



FITECH LABORATORIES

xTier™ 2.2.1 - Quickstart Guide

Copyright © Fitech Laboratories Inc.
300 Montgomery Street • Suite 621
San Francisco, CA 94104
Phone 415.371.8234 • Fax 415.371.8237

Table of Contents

Introduction	4
System Requirements	5
Required .jar Files	5
xTier™ Kernel Design.....	7
Micro-Kernel.....	7
xTier™ Micro-kernel Code Examples.....	7
Creating Your Own Micro Kernel	8
Building a Simple Application.....	9
Configuring the Application	11
Starting xTier™ for the First Time.....	14
Sending an Email.....	15
Conclusion	19

Introduction

xTier™ 2.2.1 ships with extensive Javadoc that covers not only the API, but usage examples (with code), configuration of the services, and the design rational for the way in which the services were written in addition to the way in which the services are to be utilized. This manual will not duplicate the information contained in the Javadoc, but instead, serve as an xTier™ “primer” by providing a simple example of utilizing xTier™ with J2SE. This example should serve to familiarize the developer with the concepts associated with utilizing xTier™ in their projects.

The best way to get started with xTier™ 2.2.1 is to delve into code and explore a single service. Although the way in which each individual service works is obviously different, the way in which xTier™ 2.2.1 integrates into a project, using the appropriate micro-kernel, is the same for all of the services. Additionally, the way in which a handle is obtained for each of the services is common across all services. In general there are 5 steps associated with utilizing any of the xTier™ 2.2.1 services.

- 1) Importing the proper jar files and ensuring that they are in the classpath (Note, this is only required once to use xTier™ which then allows all services to be used, it is not necessary for each service.)
- 2) Configuring xTier™ 2.2.1's and the service's XML configuration file with environment / project specific details
- 3) Starting xTier™ 2.2.1 using the micro-kernel
- 4) Acquiring a handle to the service
- 5) And finally, using the service once you have a handle

TIP: Once you know how to start xTier™ 2.2.1 and acquire a service handle for one xTier™ 2.2.1 service, you know how to acquire a handle for all of the xTier™ 2.2.1 services. (Of course, once you have a handle, using a given service will vary based on the services' API!)

Let's get started with a simple Java command line application. In this simple application we will utilize xTier™ 2.2.1, the xTier™ 2.2.1 Standard Micro-kernel and the xTier™ 2.2.1 Email Service to send an email.

System Requirements

There are many different environments and tools that can be utilized with xTier™ 2.2.1. In fact xTier™ 2.2.1 doesn't impose a particular toolset, environment or operating system beyond Java 1.4. Feel free to utilize the environment and IDE that you are most familiar with. Minimally you will need the following:

- JDK 1.4.x
- xTier™ 2.2.1
- IDE or Text Editor

TIP: Remember the installation location that you specified during the installation process. For the purposes of this example, the physical installation directory for xTier™ 2.2.1 is `c:\xtier-2.2.1` and is referred to as `%XTIER_ROOT%`.

Required .jar Files

.jar files located at `%XTIER_ROOT%/lib/ext` are required for compilation and execution of example code.

`%XTIER_ROOT%/lib/xtier` contains number of xTier™ .jar files as well as all provided micro-kernels (in separate sub-folders for each application server supported). Only **xtier.jar** or **xtier-debug.jar** is mandatory for including into classpath. **xtier-std-mk.jar** is required if standard in-process micro-kernel is used, otherwise other micro-kernels' .jar (and optionally XML configuration) files are necessary.

If you are want to run xTier™ 2.2.1 in *full* debug mode (which generates maximum console output) then **xtier-debug.jar** should be used instead of **xtier.jar**. This mode should be used only for debugging purposes.

TIP: Remember that these jar files are required for both compilation of your application as well as execution of your application.

xTier™ Kernel Design

Micro-Kernel

The xTier™ micro-kernel is a very compact software module that is responsible for starting the xTier™ kernel in the various hosting environments, such as in-process J2SE, various J2EE Application Servers, as well as any custom Java-based middleware, that xTier™ can be deployed in. The micro-kernel isolates the xTier™ Kernel services from the specifics of the hosting environment. For example, different hosting environments may dictate different startup and shutdown procedures; they may or may not provide native XA transaction monitors, logging capabilities, or a JMX server.

xTier™ is pre-packaged with micro-kernels for JBoss 3x/4x and BEA WebLogic 7x/8x application servers as well as an in-process micro-kernel for J2SE. Developers are not limited in any way to these five micro-kernels and their corresponding hosting technologies as xTier™ provides full support for developers to write their own micro-kernels and thus easily integrates into practically any Java based infrastructure.

The only responsibility of the micro-kernel is to start the xTier™ Kernel (which in its turn will load and initialize all the services), this gives the micro-kernel developer a multitude of deployment options. In fact, considering the ultra-compact size of xTier™ and its micro-kernels, it is extremely easy to create a micro-kernel, for example, for mobile applications.

xTier™ Micro-kernel Code Examples

xTier™ ships with code examples for the JBoss 3.x Application server in-process micro-kernel. You can find the Standard Micro-kernel (J2SE) in the `%XTIER_ROOT%\examples\java\com\fitechlabs\xtier\examples\microkern` directory and the JBoss micro-kernel in the

`%XTIER_ROOT%\com\fitechlabs\xtier\examples\jboss3`
directory.

Creating Your Own Micro Kernel

Creating custom micro kernels is very easy. The only contract for the micro kernel is that it should start the xTier™ kernel by calling the **XTierKernel.start(java.util.Map props)** method, where props contains an instance of **MicroKernelContext** with the key **MicroKernelContext.MICRO_KERNEL_CONTEXT**.

Notice that there are no restrictions what-so-ever on what a micro kernel is; it can be an MBean as in case of JBoss, it can be special startup class as in case of BEA WebLogic or it can be a simple class with a **main(...)** method that starts from the command line.

The bulk of the effort in micro kernel implementation is in implementing the **MicroKernelContext** interface that has to be passed to the **start(...)** method of the xTier™ kernel.

xTier™ provides detailed Javadoc for these methods as well as sample implementation for both in-process and JBoss 3 micro-kernels.

Building a Simple Application

Let's get started using xTier™ 2.2.1. We are going to create a simple Java application that sends an email utilizing the xTier™ 2.2.1 Email Service. We will start with a simple skeleton for a main class.

```
public class EmailDemo {  
  
    public static void main(String[] args) throws Exception {  
    }  
}
```

Note, the addition of **throws Exception** on the **main()** method is only to simplify the demonstration code and shouldn't be used in production code. Exceptions should be handled appropriately by the application code.

To this we need to add the lines of code that will start xTier™ 2.2.1 using the Standard Micro-kernel. The first step is to add the appropriate imports to the source file:

This line imports the xTier™ 2.2.1 Kernel.

```
import com.fitechlabs.xtier.kernel.*;
```

This line imports the xTier™ 2.2.1 Standard Micro-kernel.

```
import com.fitechlabs.xtier.microkernel.std.*;
```

This line imports the xTier™ 2.2.1 Email Service.

```
import com.fitechlabs.xtier.services.email.*;
```

This line imports the Java 1.4 util package. This is used to specify the current locale when initializing the xTier™ Kernel.

```
import java.util.*;
```

Next we will create a new method in our skeleton class called **startXtier()**.

With the introduction of xTier™ 2.2.1 the way in which the xTier™ 2.2.1 Kernel is initialized has changed. xTier™ 2.2.1 introduces the **StandardMicroKernelParams** class. This class provides a convenient way to initialize all of the parameters that must be passed to the standard micro-kernel when calling the **start()** method.

The method signature and body follows:

```
public static void startXtier() throws Exception {  
    }  
}
```

Again note that the **throws Exception** clause only serves to simplify this demonstration code, in a production system, exceptions should be handled appropriately.

The first step is to declare and create a new **StandardMicroKernelParams** object.

```
StandardMicroKernelParams params = new StandardMicroKernelParams();
```

Next set the appropriate parameters. The code to set the relevant parameters follows.

```
params.setRoot(System.getProperty("XTIER_ROOT"));  
params.setKernelRegion("emaildemo");  
params.setLocale(Locale.US);
```

The first line sets the xTier™ installation root. Note that the **XTIER_ROOT** system property is not set, then the Standard Micro-kernel attempts to discover the location of the installation root. The next line sets the "Kernel Region" which specifies the configuration region contained in the **xtier-kernel.xml** configuration file (discussed below). In this case the "emaildemo" region is specified. The third line sets the current locale. These are not all of the parameters that can be set, but they are the ones that need to be set for our example.

The next step is to start the kernel with the parameters that we have just set. The following line starts the micro-kernel.

```
// Start the micro kernel that will start the xTier kernel.
StandardMicroKernel.start(params);
```

The complete code listing for the **startXtier()** method follows.

```
public static void startXtier() throws Exception {
    // Create holder for micro kernel parameters.
    StandardMicroKernelParams params = new StandardMicroKernelParams();

    // Note that if XTIER_ROOT property is not specified (-DXTIER_ROOT=...) then the
    // standard micro kernel will attempt to auto-discover the xTier installation root.
    // If auto-discover fails micro kernel will not be able to start.
    params.setRoot(System.getProperty("XTIER_ROOT"));

    // The configuration region that was created using the management console
    params.setKernelRegion("emaildemo");
    params.setLocale(Locale.US);

    // Start the micro kernel that will start the xTier kernel.
    StandardMicroKernel.start(params);
}
```

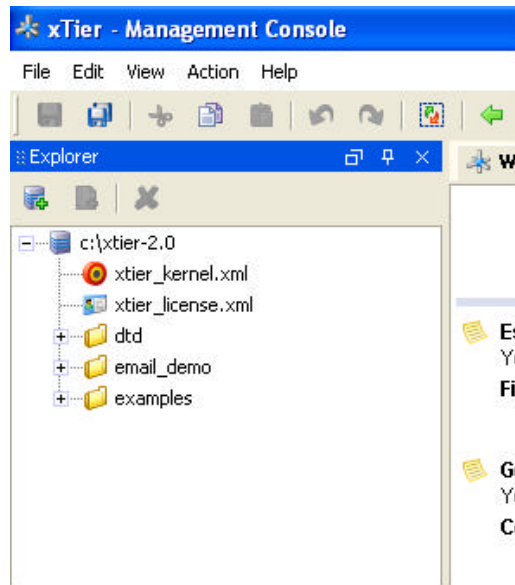
At this point you might be tempted to try to start the xTier™ Kernel, however, there are several more things that need to be configured before xTier™ will successfully start. Specifically, the “emaildemo” region specified above must be added to the **xtier-kernel.xml** file and we must customize the XML configuration files associated with the Email Service.

Configuring the Application

The next step is to create a separate set of configuration files for the “emaildemo” region.

- 1) Create the **%XTIER_ROOT%/config/email_demo** directory.
- 2) Copy **%XTIER_ROOT%/config/examples/xtier-log.xml** to **%XTIER_ROOT%/config/email_demo**
- 3) Copy **%XTIER_ROOT%/config/examples/xtier-email.xml** to **%XTIER_ROOT%/config/email_demo**

Start the xTier™ Management Console from the xTier™ 2.2.1 Windows Start group. Close the “Tip of the Day” (this feature can be turned off using the checkbox at the bottom of the dialogue box) and expand the c:\xtier-2.2.1 root in the explorer and you will be presented with the following screen. There are two files that need to be updated with the correct settings for the **emaildemo** region. Notice the **email_demo** directory that was created in the previous step.



Open the **xtier_kernel.xml** file by double-clicking it in the explorer. It will load in the tabbed editor. Examine the **xtier_kernel.xml** file. Currently there is only one region in this file and it corresponds to the examples that ship with xTier™ 2.2.1. In order for the code that was created to start the xTier™ Kernel to properly run, an **emaildemo** region must be created. Enter the following XML into **xtier_kernel.xml** after the examples region.

```
<region name="emaildemo" feature-set="eval" config-path="config/email_demo">
  <start service="jmx"/> <!-- No XML configuration. -->
  <start service="log" region="examples"/>
  <start service="email" region="emaildemo"/>
</region>
```

The first thing to examine is the region tag itself.

```
<region name="emaildemo" feature-set="eval" config-path="config/email_demo">
```

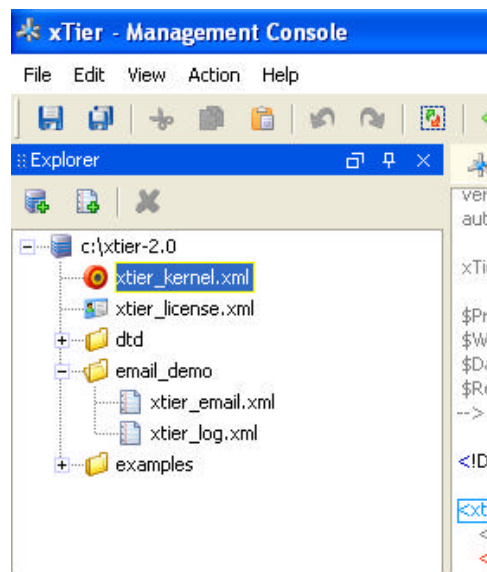
The **name** attribute specifies the name of the region. This corresponds to the parameter that was set in the **startXtier()** method defined above. The **config-path** specifies the path, *relative to %XTIER_ROOT%* where the configuration files for the individual services can be found. In this case, **config-path** is set to the directory that was created in the previous steps (**%XTIER_ROOT%/config/email_demo**).

The **<start service=""/>** tags specify which services to start - in this case, the JMX, Log and Email services. For this example using email, this is the minimal number of services that are required to start xTier™. Notice as well, that within the services tag a region is specified for the individual service configuration XML files as well. This provides a great deal of

flexibility to support multiple configurations to accommodate different environments. For instance, different regions could be specified for development, test, staging and production environments. Note, the JMX service doesn't have an xml configuration file associated with it, however, both the log and the email services do.

In this case, since the configuration files for the log and email services were copied from the examples shipped with xTier™ 2.2.1, the region for the log service is specified as "examples". Since there are no configuration changes that need to be made to the log service XML leave the region defined as examples. Notice, however, that the region for the email service is **emailedemo**. This will correspond to the region specified in the **xtier-email.xml** file.

Expand the **email_demo** folder in the xTier™ Management Console explorer.



The two configuration files that were copied earlier can be seen. Open the **xtier_email.xml** file by double-clicking the file in the explorer. There are several parameters that need to be changed so that the email service can properly send emails.

Change `<region name="examples">` to `<region name="emailedemo">`. This corresponds to the `<start service="email" region="emailedemo"/>` explained above and defined in **xtier-kernel.xml**.

The next step is to change the individual mail parameters. The complete XML definition follows:

```
<region name="emailedemo">
  <!-- Default host. -->
  <smtp-host>smtp.change.this.com</smtp-host>

  <!-- Default port, if different from 25. -->
  <smtp-port>25</smtp-port>

  <!-- Default from address. -->
  <dflt-from>change@this.com </dflt-from>

  <!-- Default Java email content encoding. -->
  <java-dflt-encoding>quoted-printable</java-dflt-encoding>

  <!-- Default .NET email content encoding. -->
  <clr-dflt-encoding>unicode</clr-dflt-encoding>
</region>
```

Change the "smtp server" tag, to a valid server that you have permission to use, and set the "dflt from" (default from) address to a *valid account*. Save both **xtier-email.xml** and **xtier-kernel.xml** and exit the xTier™ Management Console.

Starting xTier™ for the First Time

The next step is to add the code to start and stop xTier™ to the **main()** method. This will determine if the services are properly configured before adding the code to send an email. To start xTier™ just call the startXtier() method that was created earlier.

```
startXtier();
```

And in order to stop xTier™ obtain an instance of XtierKernel and call the stop() method with the following code:

```
XtierKernel.getInstance().stop();
```

The complete listing for the **main()** method follows:

```
public static void main(String[] args) throws Exception {

  // Start xTier kernel.
  startXtier();

  // Call the email example
  doEmail();
}
```

```

// Stop the xTier kernel.
XtierKernel.getInstance().stop();
}

```

Of course, this simple application doesn't use any of the kernel services yet, but it does serve as the basis to troubleshoot any configuration errors. Compile and run this application, taking care to remember the .jar files which are needed in the application's classpath.

A list of messages should appear in the console, indicating that xTier™ 2.2.1 is starting. If xTier™ has successfully started, the following line should appear in the console. (Of course, the timestamp will be different!):

```

Sun Feb 29 00:02:50 EST 2004 <trace:kernel-startup> <kernel> **** xTier kernel started ok on Sunday, February
29, 2004, 12:02:50 AM EST ****
...
Sun Feb 29 00:02:51 EST 2004 <trace:main> <kernel> **** xTier kernel stopped ok on Sunday, February 29,
2004, 12:02:51 AM EST ****

```

These lines will be among the lines providing notification that the services have started and stopped.

Most typically any errors displayed will be associated with a missing jar file. (If you see many error messages of the form **NoClassDefFound** then check to make sure that all of the jar files are included.)

Alternatively, often an error will occur because of improperly specified regions. Confirm that the regions are properly specified.

Sending an Email

Now that the service is properly configured and we have xTier™ 2.2.1 starting properly it is time to start using xTier™ 2.2.1 services. The first thing that is required to use any of the xTier™ 2.2.1 services is to acquire an instance of the email service. In the case of the email service acquiring an instance is accomplished by using the following line of code:

```
EmailService email = XtierKernel.getInstance().email();
```

The next step is to create an email session that we can utilize to send an email. xTier™ 2.2.1 uses the idea of a session to represent an email which will be sent. The session contains all of the information about the email, such as the **from address**, the **to address**, the **email type**, the **subject** and **message content** and optionally an **attachment**. This step is accomplished by using the following line of code:

```
EmailSmtpSession session = email.getSmtpSession();
```

Now that we have an email session, all that is left to do is populate the session with the email information, and then call the send method.

```
session.setFromAddress("jwebster@fitechlabs.com");  
session.addToAddress("jwebster@fitechlabs.com");  
session.setMime("text/html");  
session.setSubject("HTML email service test.");  
session.setContent("<b>This is the <font size=+1>HTML</font> email test.</b>");
```

Notice the line **session.setMime("text/html")** in the code example above. This line of code sets the type of the email to be an HTML email. (For more information on the different types permissible using the email service, see the developer's manual.) This is why in the **session.setContent(. . .)** call the message subject is formatted as HTML.

TIP: Make sure that the from address has permission to send email using the SMTP-HOST that was configured earlier! Additionally, note that a username and password can be set for the current mail session as well should authentication be required.

The final step is to send the email. Sending the email is accomplished using the following line of code.

```
session.send();
```

Now, let's add this code to our simple demonstration program in a method called **doEmail()**. The complete code listing (note again, the **throws Exception** clause is only used to simplify the code and eliminate the **try... catch** blocks for clarity) for this method follows:

```
public static void doEmail() throws Exception {  
  
    // Get the instance of 'email' service.  
    EmailService email = XtierKernel.getInstance().email();  
  
    // Get an email session  
    EmailSmtpSession session = email.getSmtpSession();  
  
    // Set the email session's parameters.  
    session.setFromAddress("jwebster@fitechlabs.com");  
    session.addToAddress("jwebster@fitechlabs.com");  
    session.setMime("text/html");  
    session.setSubject("HTML email service test.");  
  
}
```

```

        session.setContent("<b>This is the <font size=+1>HTML</font> email test.</b>");

// Send the email
session.send();

}

```

Now in order to actually send the email, add the following to the **main()** method, after that call to **startXtier()**.

```
doEmail();
```

The complete code listing follows:

```

package com.fitechlabs.xtier.quickstart;

import com.fitechlabs.xtier.microkernel.std.*;
import com.fitechlabs.xtier.kernel.*;
import com.fitechlabs.xtier.services.email.*;
import java.util.*;

public class EmailDemo {

    public static void startXtier() throws Exception {
        // Create holder for micro kernel parameters.
        StandardMicroKernelParams params = new StandardMicroKernelParams();

        // Note that if XTIER_ROOT property is not specified (-DXTIER_ROOT=...) then the
        // standard micro kernel will attempt to auto-discover the xTier installation root.
        // If auto-discover fails micro kernel will not be able to start.
        params.setRoot(System.getProperty("XTIER_ROOT"));

        // The configuration region that was created using the management console
        params.setKernelRegion("emaildemo");
        params.setLocale(Locale.US);

        // Start the micro kernel that will start the xTier kernel.
        StandardMicroKernel.start(params);
    }

    public static void doEmail() throws Exception {

        // Get the instance of 'email' service.
        EmailService email = XtierKernel.getInstance().email();

        // Get an email session
        EmailSmtpSession session = email.getSmtpSession();

        // Set the email session's parameters.
        session.setFromAddress("jwebster@fitechlabs.com");
        session.addToAddress("jwebster@fitechlabs.com");
        session.setMime("text/html");
        session.setSubject("HTML email service test.");
    }
}

```

```
        session.setContent("<b>This is the <font size=+1>HTML</font> email test.</b>");  
        session.send();  
    }  
  
    public static void main(String[] args) throws Exception {  
        // Start xTier kernel.  
        startXtier();  
  
        // Call the email example  
        doEmail();  
  
        // Stop the xTier kernel.  
        XtierKernel.getInstance().stop();  
    }  
}
```

Conclusion

You have now successfully integrated xTier™ 2.2.1 into an application, and utilized an xTier™ 2.2.1 service. The general steps for all of the services follow the same steps.

- Import the proper packages and ensure that the appropriate .jar files are on the classpath.
- Configure the service that you are going to utilize
- Start xTier™ 2.2.1 using the appropriate micro kernel
- Acquire a handle to the service
- Use the service API.

Finally, the best source for information about the API, service configuration, service examples, and design information is contained in the extensive Javadoc.