IBM Software Group

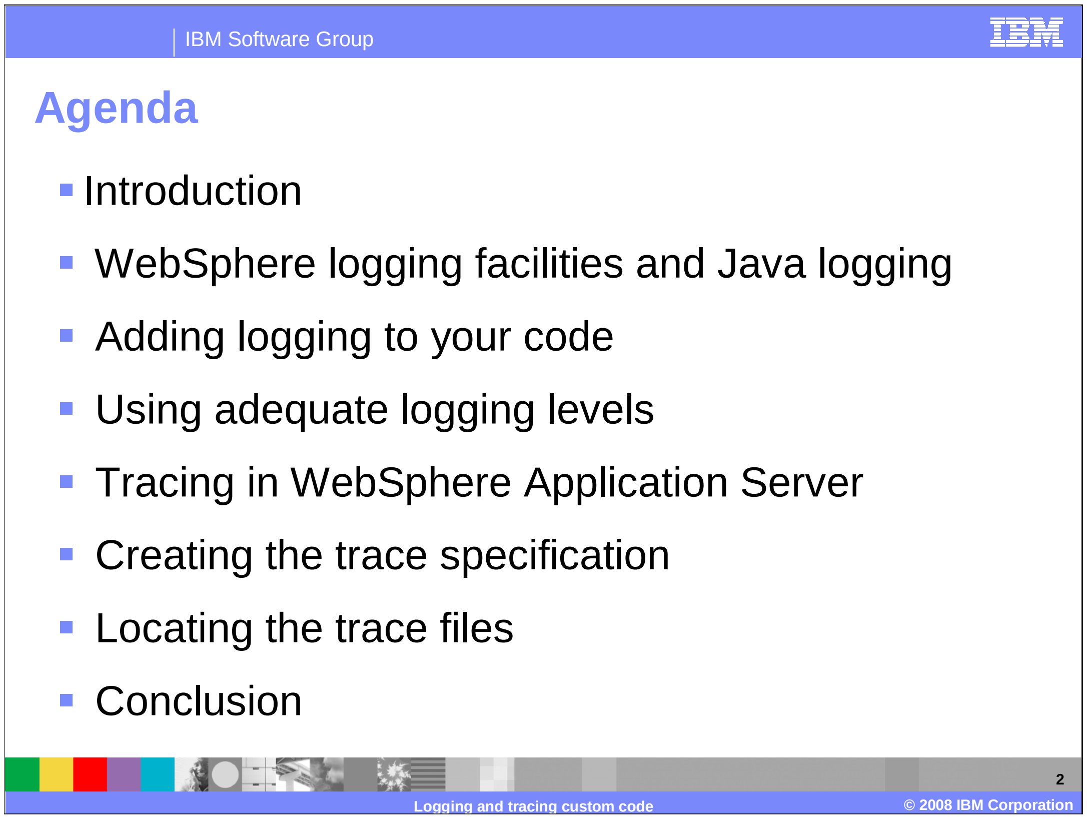
WebSphere® Commerce V6.0

Logging and tracing custom code

business on demand.

© 2008 IBM Corporation
Updated June 6, 2008

This presentation covers WebSphere Commerce version 6 logging and tracing, and provides an introductory look at how to implement logging and tracing for custom code.



Agenda

- Introduction
- WebSphere logging facilities and Java logging
- Adding logging to your code
- Using adequate logging levels
- Tracing in WebSphere Application Server
- Creating the trace specification
- Locating the trace files
- Conclusion

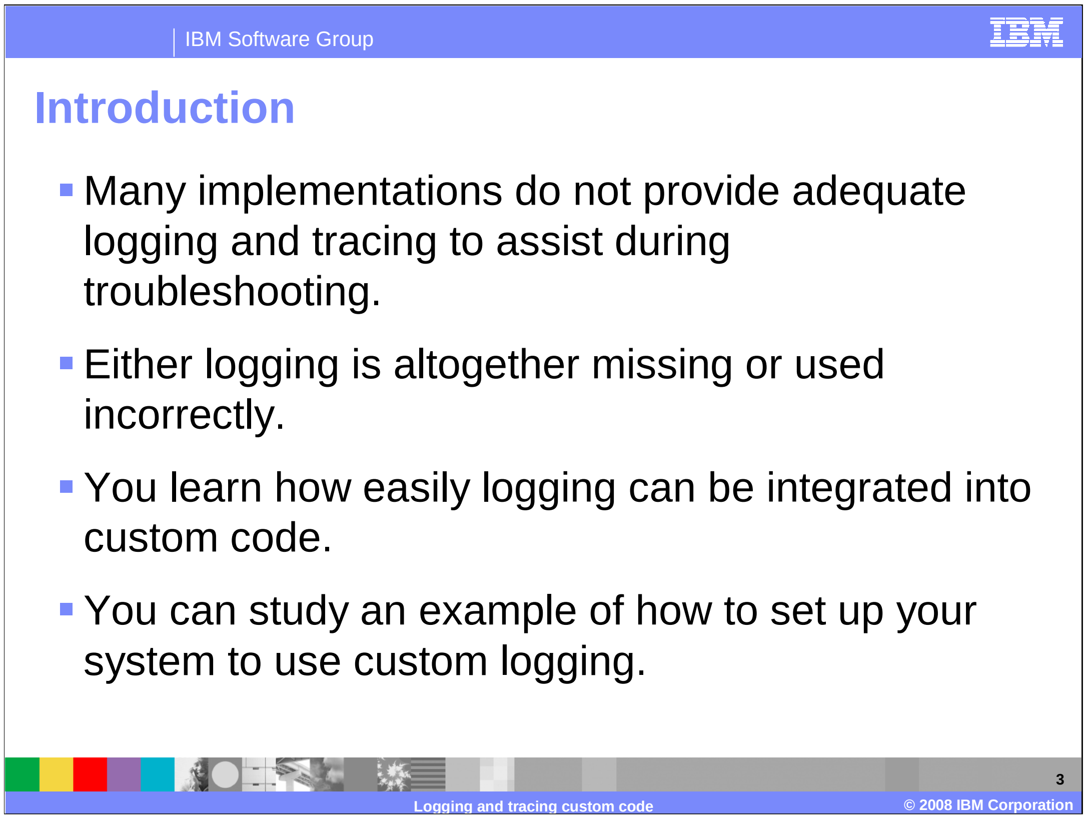
© 2008 IBM Corporation

In this presentation, these topics are discussed:

You begin by reviewing the WebSphere logging facilities and the Java logging API.

Next, you learn how you can enhance the maintainability of your code by adding logging, and see the different logging levels and recommendations.

Finally, you learn how to configure tracing using the WebSphere Application Server administrative console.



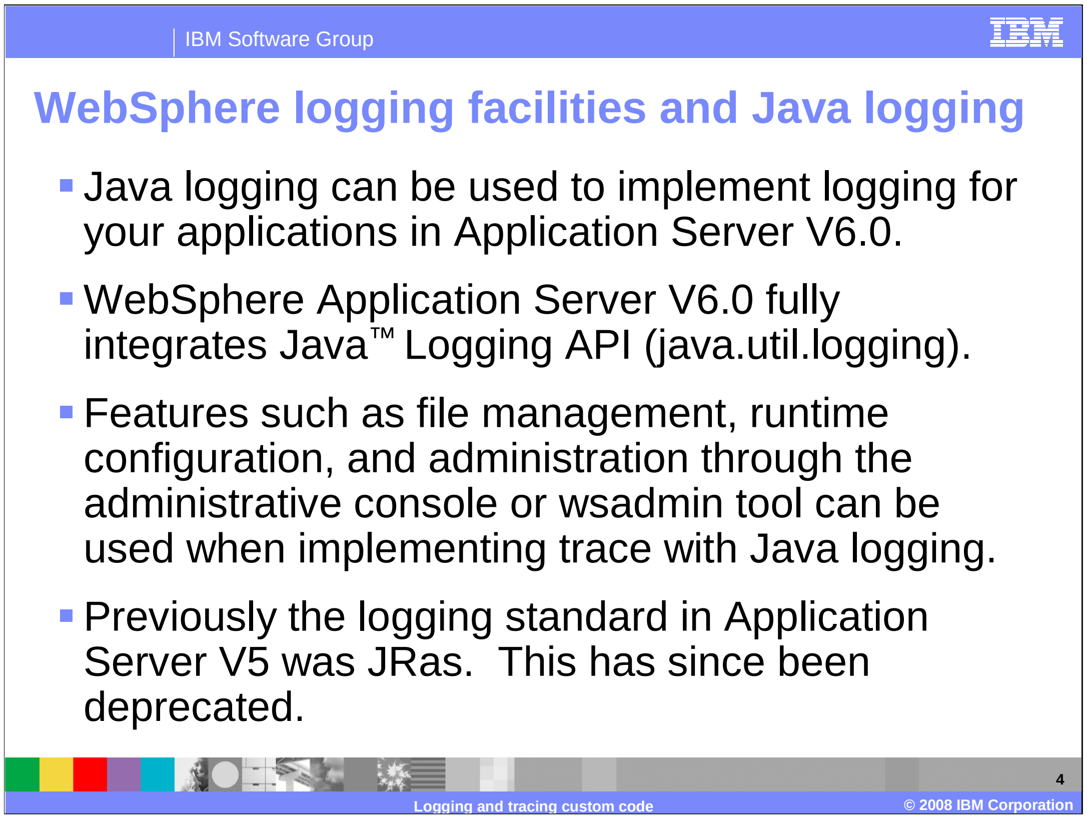
Introduction

- Many implementations do not provide adequate logging and tracing to assist during troubleshooting.
- Either logging is altogether missing or used incorrectly.
- You learn how easily logging can be integrated into custom code.
- You can study an example of how to set up your system to use custom logging.

Even though logging is a key tool for problem determination, it is not uncommon to find code that does not implement it, or it does not use it effectively.

This presentation shows how you can implement logging for your custom code.

Subsequent slides look at the logging facilities offered by WebSphere Application Server version 6, and show examples of best practices on how to make your logging more effective.


IBM Software Group

WebSphere logging facilities and Java logging

- Java logging can be used to implement logging for your applications in Application Server V6.0.
- WebSphere Application Server V6.0 fully integrates Java™ Logging API (java.util.logging).
- Features such as file management, runtime configuration, and administration through the administrative console or wsadmin tool can be used when implementing trace with Java logging.
- Previously the logging standard in Application Server V5 was JRas. This has since been deprecated.

4

Logging and tracing custom code

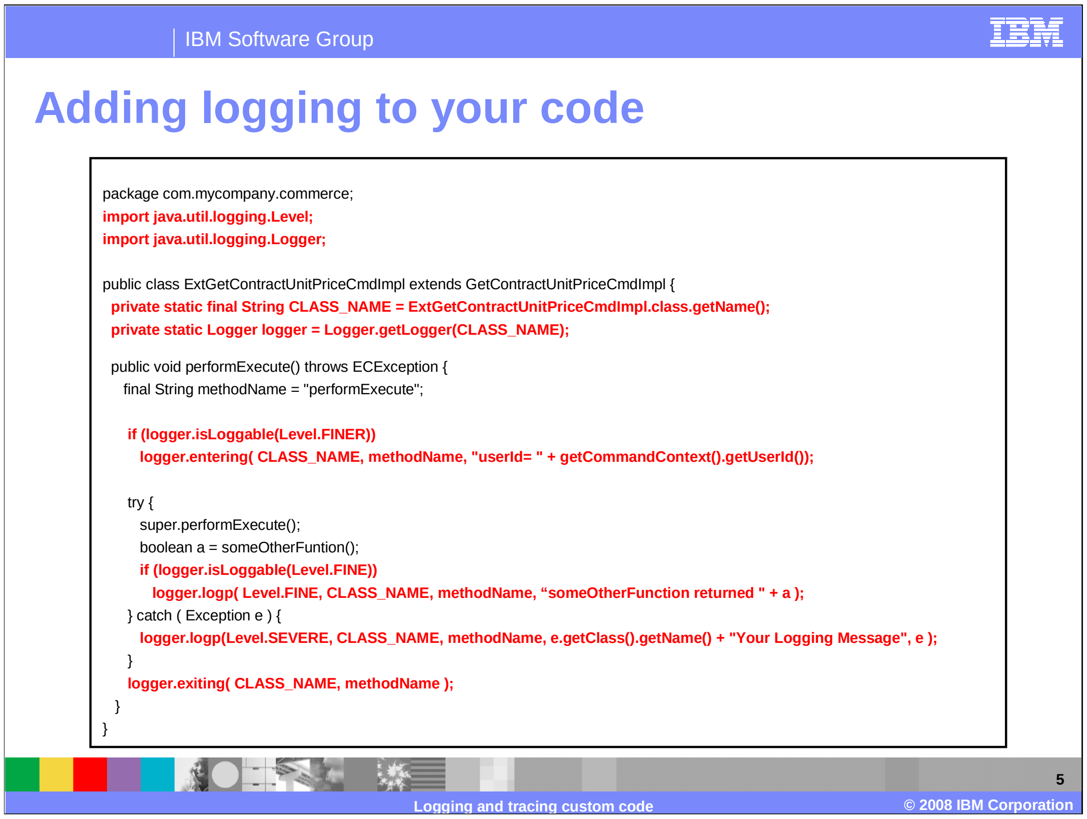
© 2008 IBM Corporation

Starting with versions 5.6.1 (fix pack 1 and higher) and 6.0, WebSphere Commerce supports WebSphere Application Server V6.0.

This version of the application server fully integrates the Java logging API, which was added in Java 1.4 to provide advanced control of informational output from applications.

In WebSphere Application Server version 6, you can implement logging for your code using the Java logging standard. This allows you to take advantage of the WebSphere logging features such as file management, runtime configuration, and administration through the administrative console or the wsadmin tool. These are described in detail in the upcoming slides.

The JRas framework, which was the logging standard in WebSphere Application Server V5, has been deprecated. If you have implemented logging using JRas, your application continues to work. However, if you face a new implementation, use Java logging instead.



Adding logging to your code

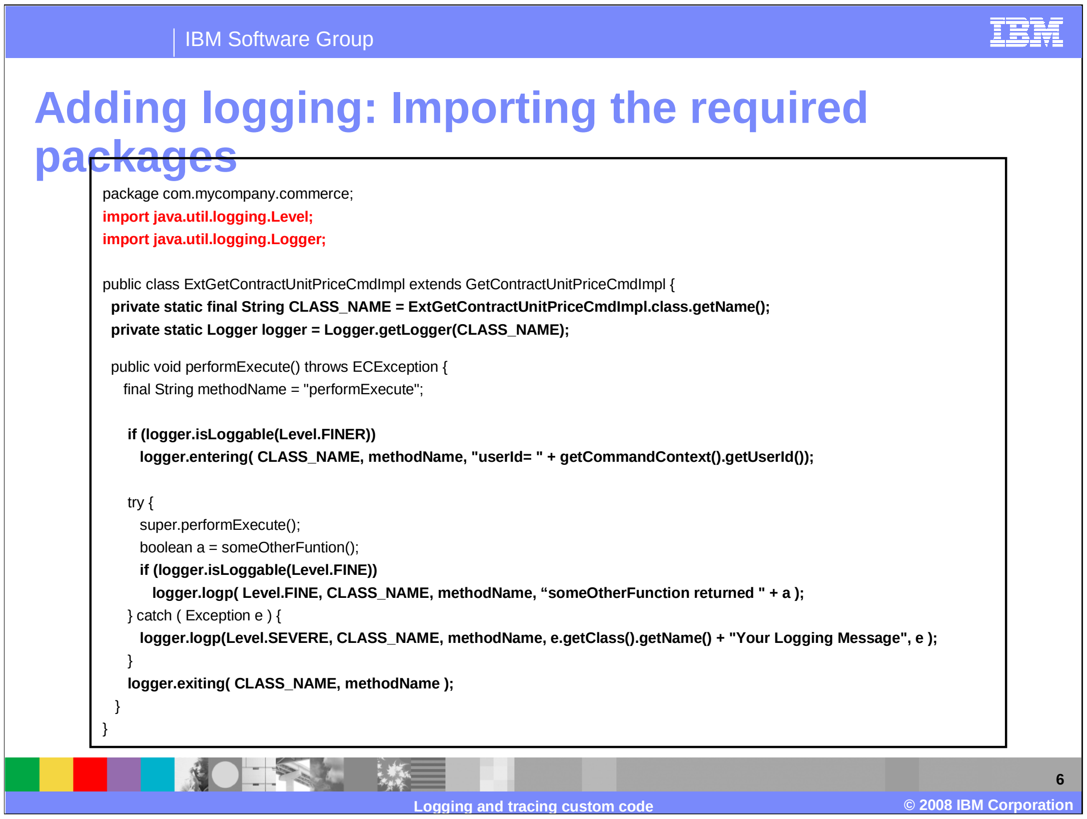
```
package com.mycompany.commerce;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ExtGetContractUnitPriceCmdImpl extends GetContractUnitPriceCmdImpl {
  private static final String CLASS_NAME = ExtGetContractUnitPriceCmdImpl.class.getName();
  private static Logger logger = Logger.getLogger(CLASS_NAME);

  public void performExecute() throws ECException {
     final String methodName = "performExecute";

     if (logger.isLoggable(Level.FINER))
        logger.entering( CLASS_NAME, methodName, "userId= " + getCommandContext().getUserId());

     try {
        super.performExecute();
        boolean a = someOtherFuntion();
        if (logger.isLoggable(Level.FINE))
           logger.logp( Level.FINE, CLASS_NAME, methodName, “someOtherFunction returned " + a );
     } catch ( Exception e ) {
        logger.logp(Level.SEVERE, CLASS_NAME, methodName, e.getClass().getName() + "Your Logging Message", e );
     }
     logger.exiting( CLASS_NAME, methodName );
  }
}
```

5

Logging and tracing custom code

© 2008 IBM Corporation

This slide provides a sample WebSphere Commerce command that implements Java Logging. The lines that have been highlighted in red correspond to the different APIs that are used for logging. In the next few slides, the code is broken down so the different sections can be explained in detail.



IBM Software Group

Adding logging: Importing the required packages

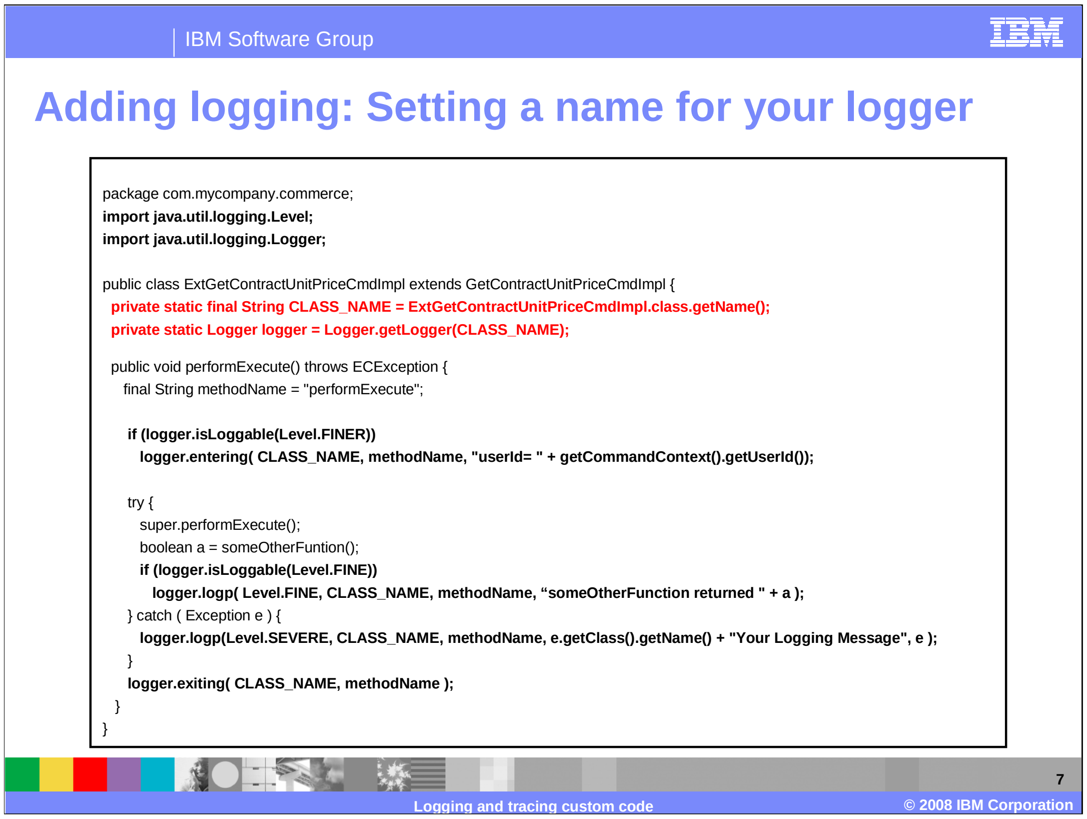
```
package com.mycompany.commerce;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ExtGetContractUnitPriceCmdImpl extends GetContractUnitPriceCmdImpl {
  private static final String CLASS_NAME = ExtGetContractUnitPriceCmdImpl.class.getName();
  private static Logger logger = Logger.getLogger(CLASS_NAME);

  public void performExecute() throws ECException {
    final String methodName = "performExecute";

    if (logger.isLoggable(Level.FINER))
      logger.entering( CLASS_NAME, methodName, "userId= " + getCommandContext().getUserId());

    try {
      super.performExecute();
      boolean a = someOtherFuntion();
      if (logger.isLoggable(Level.FINE))
        logger.logp( Level.FINE, CLASS_NAME, methodName, “someOtherFunction returned " + a );
    } catch ( Exception e ) {
      logger.logp(Level.SEVERE, CLASS_NAME, methodName, e.getClass().getName() + "Your Logging Message", e );
    }
    logger.exiting( CLASS_NAME, methodName );
  }
}
```

6

Logging and tracing custom code

© 2008 IBM Corporation

If your command uses Java logging, the first step is to import the necessary classes. In this case, the code imports the Level and Logger classes that belong to the Java util logging package.


IBM Software Group

Adding logging: Setting a name for your logger

```
package com.mycompany.commerce;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ExtGetContractUnitPriceCmdImpl extends GetContractUnitPriceCmdImpl {
  private static final String CLASS_NAME = ExtGetContractUnitPriceCmdImpl.class.getName();
  private static Logger logger = Logger.getLogger(CLASS_NAME);

  public void performExecute() throws ECException {
    final String methodName = "performExecute";

    if (logger.isLoggable(Level.FINER))
      logger.entering( CLASS_NAME, methodName, "userId= " + getCommandContext().getUserId());

    try {
      super.performExecute();
      boolean a = someOtherFuntion();
      if (logger.isLoggable(Level.FINE))
        logger.logp( Level.FINE, CLASS_NAME, methodName, “someOtherFunction returned " + a );
    } catch ( Exception e ) {
      logger.logp(Level.SEVERE, CLASS_NAME, methodName, e.getClass().getName() + "Your Logging Message", e );
    }
    logger.exiting( CLASS_NAME, methodName );
  }
}
```

7

Logging and tracing custom code © 2008 IBM Corporation

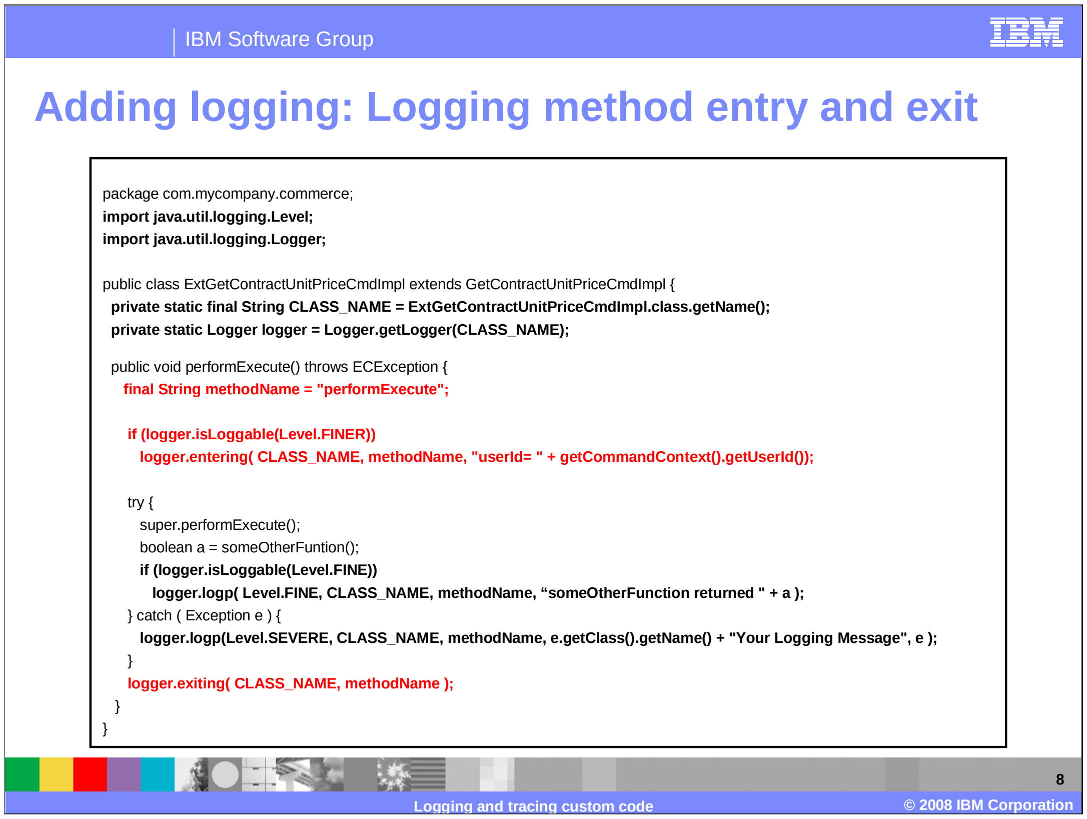
The next step is to define a logger and give it a name. The name of the logger is important because it defines an implicit hierarchy. In the example, the logger takes the name of the class where it is defined.

With this technique, you can achieve a very fine level of granularity.

By naming the logger the same as the class where it is used, the logger hierarchy matches that of the packages and classes. This allows you to enable tracing for a package and all the packages under it, or for a particular class.

Other common techniques include using the package as the logger name, or to name the logger to match the functionality they provide, such as orders, catalog, and so on.

Notice that the class name is saved as a static String. This is done because the name is reused when logging messages.


IBM Software Group

Adding logging: Logging method entry and exit

```
package com.mycompany.commerce;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ExtGetContractUnitPriceCmdImpl extends GetContractUnitPriceCmdImpl {
  private static final String CLASS_NAME = ExtGetContractUnitPriceCmdImpl.class.getName();
  private static Logger logger = Logger.getLogger(CLASS_NAME);

  public void performExecute() throws ECException {
    final String methodName = "performExecute";

    if (logger.isLoggable(Level.FINER))
      logger.entering( CLASS_NAME, methodName, "userId= " + getCommandContext().getUserId());

    try {
      super.performExecute();
      boolean a = someOtherFuntion();
      if (logger.isLoggable(Level.FINE))
        logger.logp( Level.FINE, CLASS_NAME, methodName, “someOtherFunction returned " + a );
    } catch ( Exception e ) {
      logger.logp(Level.SEVERE, CLASS_NAME, methodName, e.getClass().getName() + "Your Logging Message", e );
    }
    logger.exiting( CLASS_NAME, methodName );
  }
}
```

8

Logging and tracing custom code

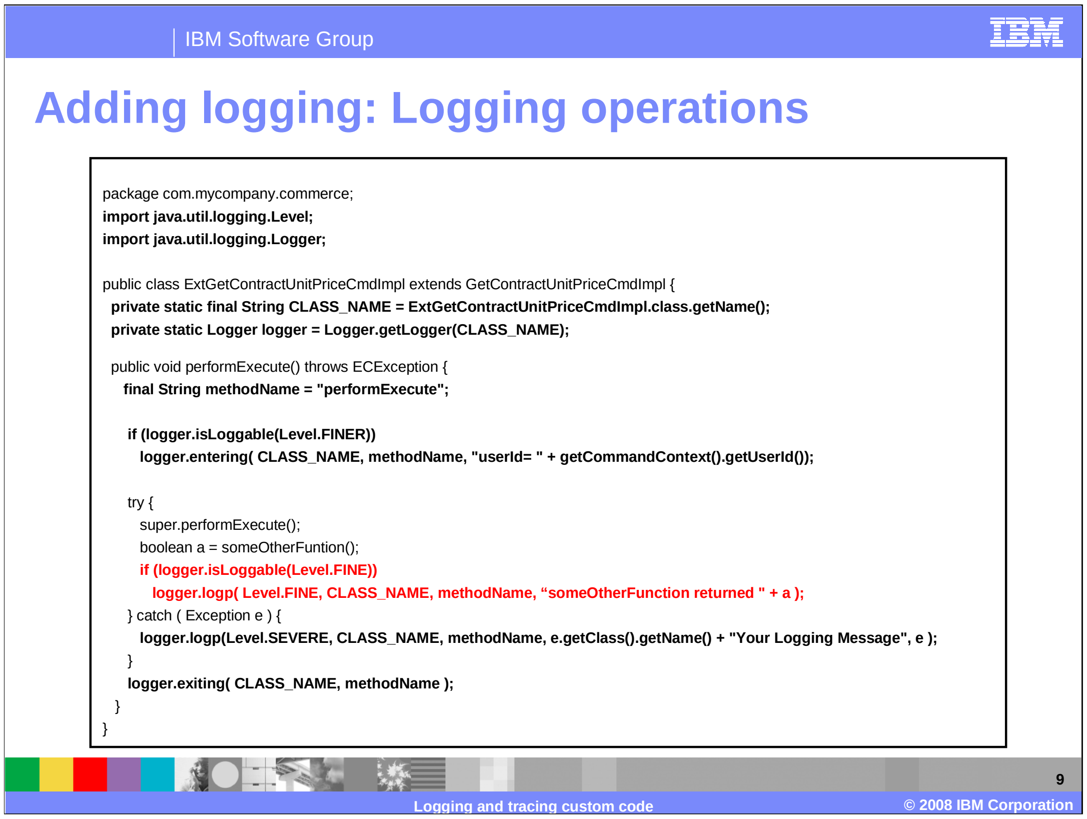
© 2008 IBM Corporation

It is a common practice to log method entry and exit for important methods. This can help you narrow down the code where the application is failing. Logger **entering** and **exiting** are convenience methods that use the FINER level.

In the example, the method name is stored in a read only string because the name is used every time you log a message.

Next, the **logger.entering** method is called if the logger is currently logging messages that use the FINER level. If you did not have this check, the code would be calling the **getCommandContext getUserId** method and the string concatenation every time the function is called, regardless of the level. Adding this check minimizes the overhead of tracing in your application.

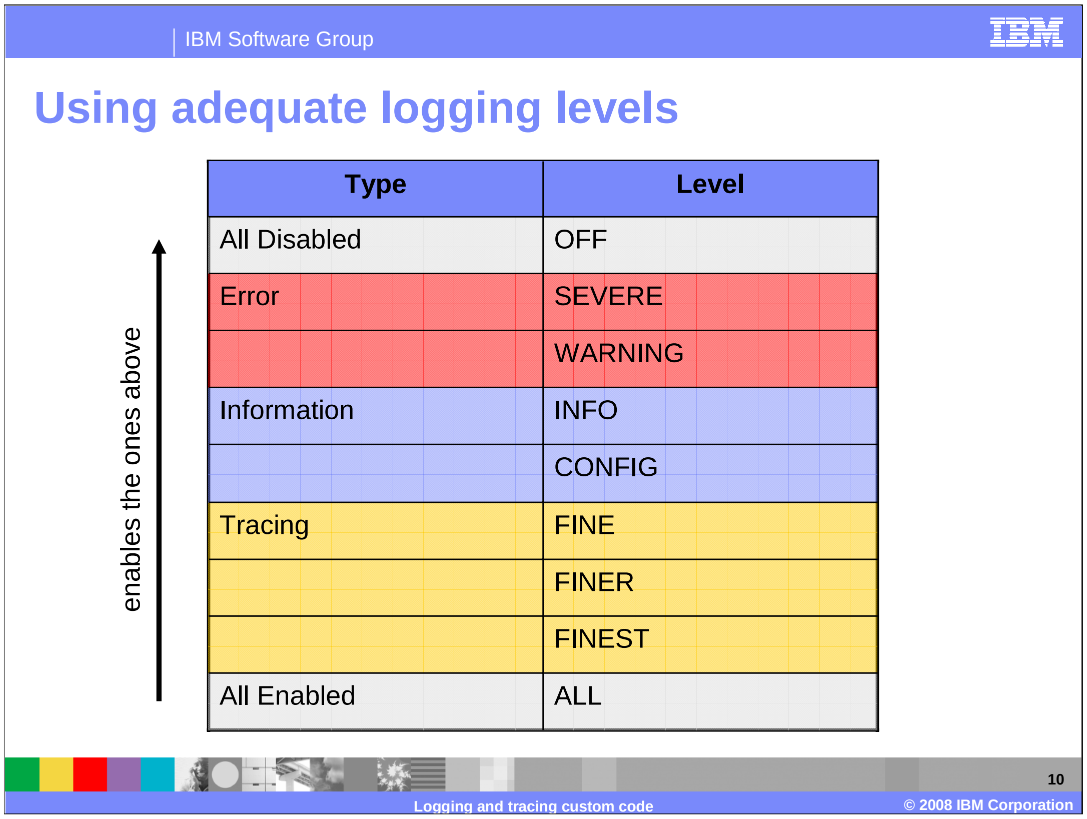
Finally, the method **exit** is logged using **logger.exiting**. Since the method is called using static parameters, the **isLoggable** API is not used.



Adding logging: Logging operations

```
package com.mycompany.commerce;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ExtGetContractUnitPriceCmdImpl extends GetContractUnitPriceCmdImpl {
  private static final String CLASS_NAME = ExtGetContractUnitPriceCmdImpl.class.getName();
  private static Logger logger = Logger.getLogger(CLASS_NAME);

  public void performExecute() throws ECException {
    final String methodName = "performExecute";

    if (logger.isLoggable(Level.FINER))
       logger.entering( CLASS_NAME, methodName, "userId= " + getCommandContext().getUserId());

    try {
       super.performExecute();
       boolean a = someOtherFuntion();
       if (logger.isLoggable(Level.FINE))
          logger.logp( Level.FINE, CLASS_NAME, methodName, “someOtherFunction returned " + a );
    } catch ( Exception e ) {
       logger.logp(Level.SEVERE, CLASS_NAME, methodName, e.getClass().getName() + "Your Logging Message", e );
    }
    logger.exiting( CLASS_NAME, methodName );
  }
}
```

This is another example of logging entries in your application. In this case, the level used is FINE. The isLoggable method is used again to prevent the string concatenation if tracing is not enabled.



Using adequate logging levels

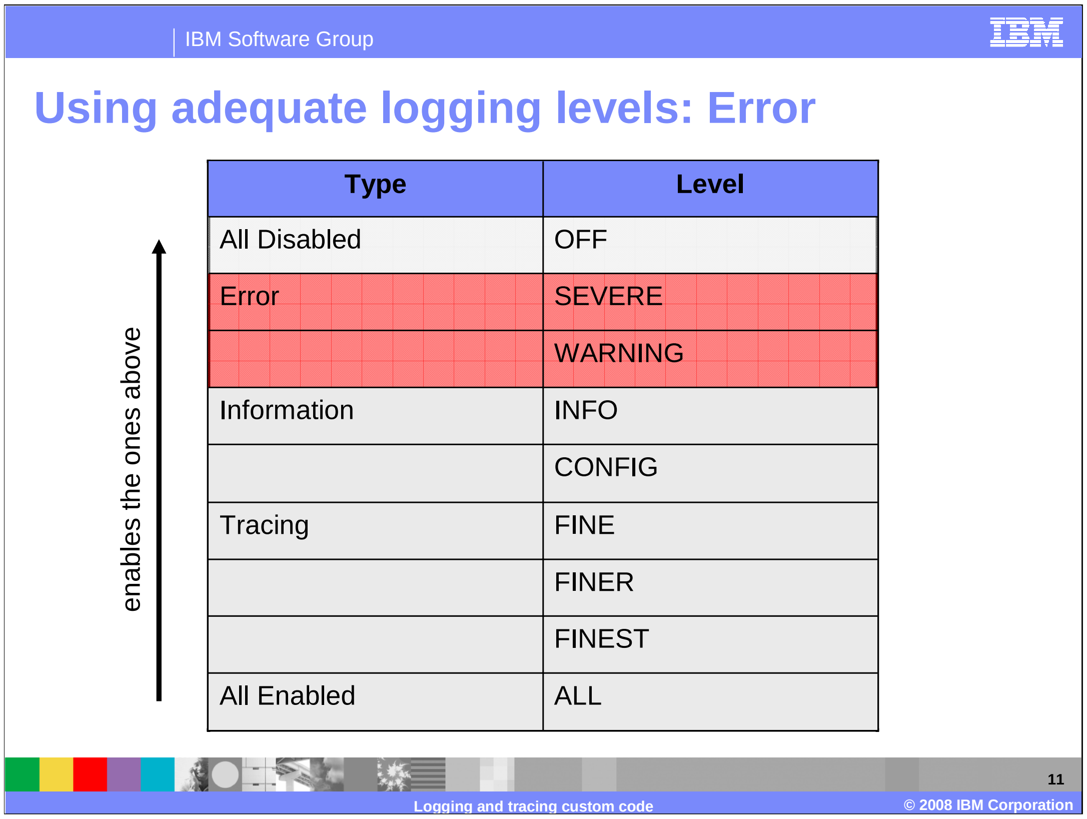
enables the ones above

Type	Level
All Disabled	OFF
Error	SEVERE
	WARNING
Information	INFO
	CONFIG
Tracing	FINE
	FINER
	FINEST
All Enabled	ALL

When logging a message, it is very important that you use the logging levels correctly. This slide shows the different levels available grouped by usage.

Logging levels can be broken down in three categories. Error and Warning, Informational and Tracing. When you enable logging at one level, you are also enabling it for all the levels above it.

This means that when you enable logging at a the FINE level, for example, all informational and error messages are also logged.



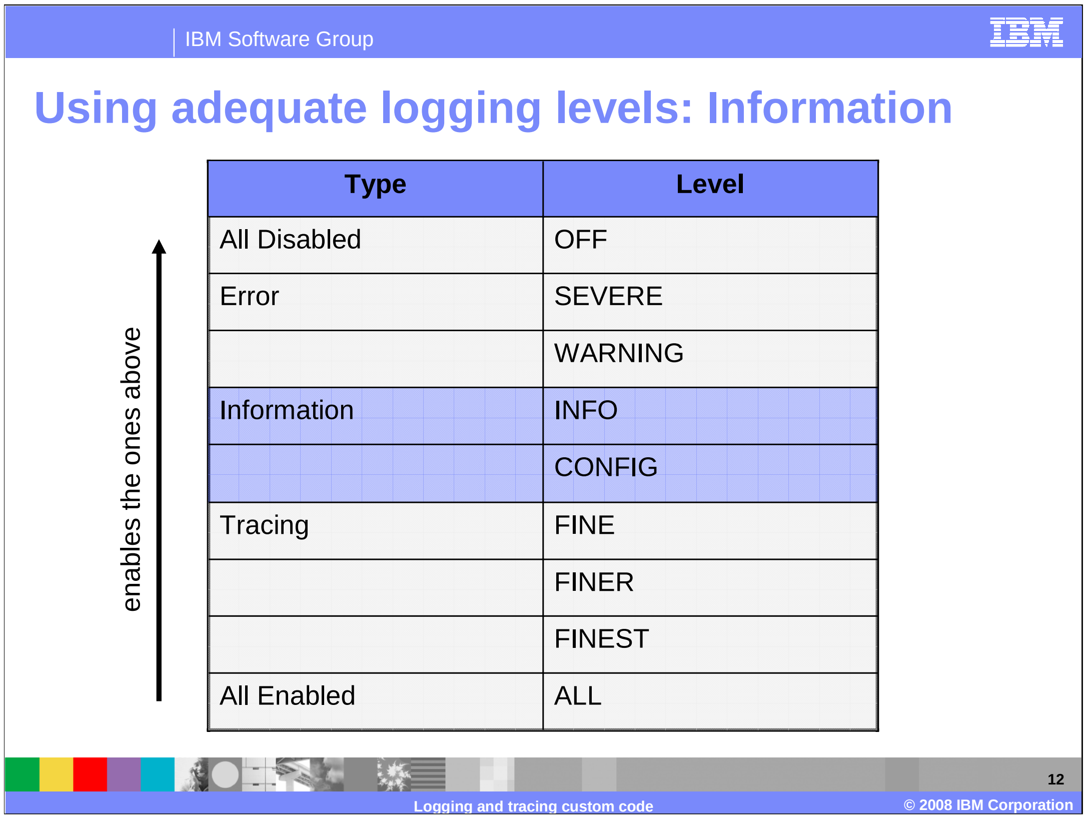
Using adequate logging levels: Error

enables the ones above

Type	Level
All Disabled	OFF
Error	SEVERE
	WARNING
Information	INFO
	CONFIG
Tracing	FINE
	FINER
	FINEST
All Enabled	ALL

Always log error conditions using SEVERE or WARNING. This ensures that exceptions are logged, even when logging is set to a minimum.

If the error condition needs to be propagated to the store front, use the WebSphere Commerce Command error handling framework,
which creates internationalized messages and allows you to specify an error view.



Using adequate logging levels: Information

Type	Level
All Disabled	OFF
Error	SEVERE
	WARNING
Information	INFO
	CONFIG
Tracing	FINE
	FINER
	FINEST
All Enabled	ALL

enables the ones above

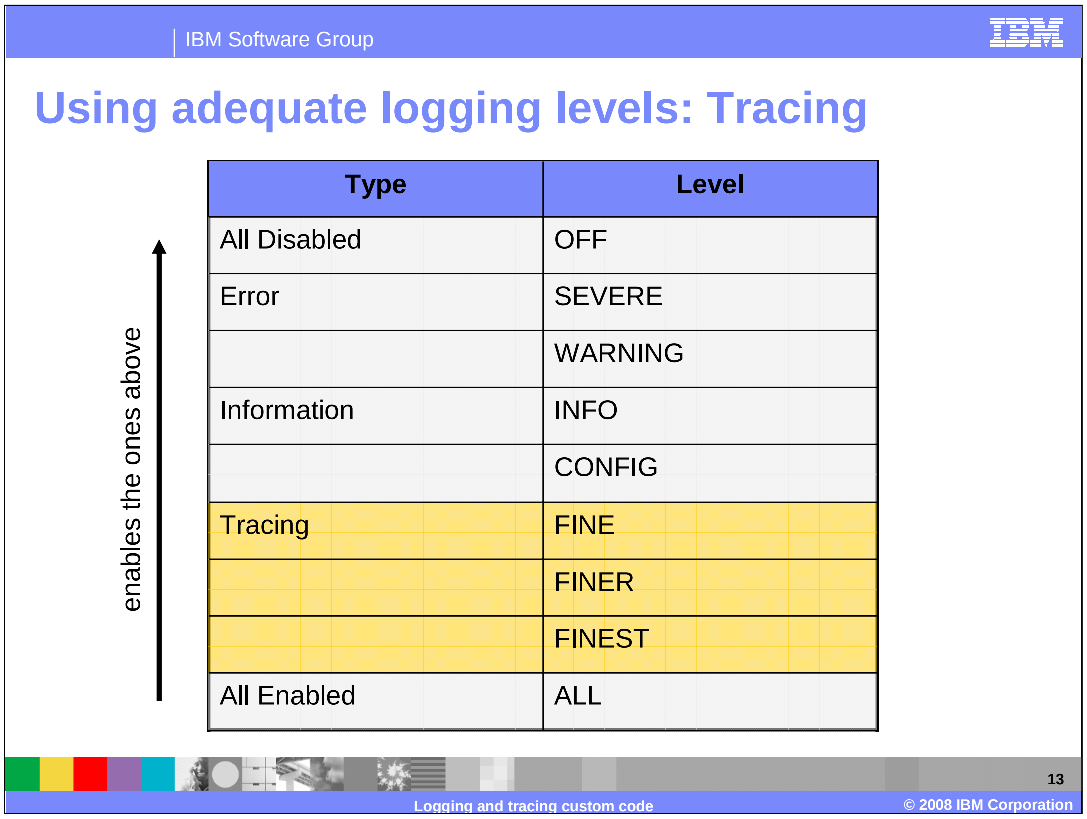
© 2008 IBM Corporation

WebSphere Application Server is configured to INFO by default, and you should keep it set at this level, because most messages are logged at this level.

Use the INFO level carefully because too much logging affects the performance of your application.

For example, do not use INFO levels for commands, such as OrderItemAdd, that are run from the store front because they can flood the logs with messages.

If the message you want to log is likely to be printed in a common operation you should consider using a tracing level instead.


IBM Software Group

Using adequate logging levels: Tracing

enables the ones above

Type	Level
All Disabled	OFF
Error	SEVERE
	WARNING
Information	INFO
	CONFIG
Tracing	FINE
	FINER
	FINEST
All Enabled	ALL

13

Logging and tracing custom code

© 2008 IBM Corporation

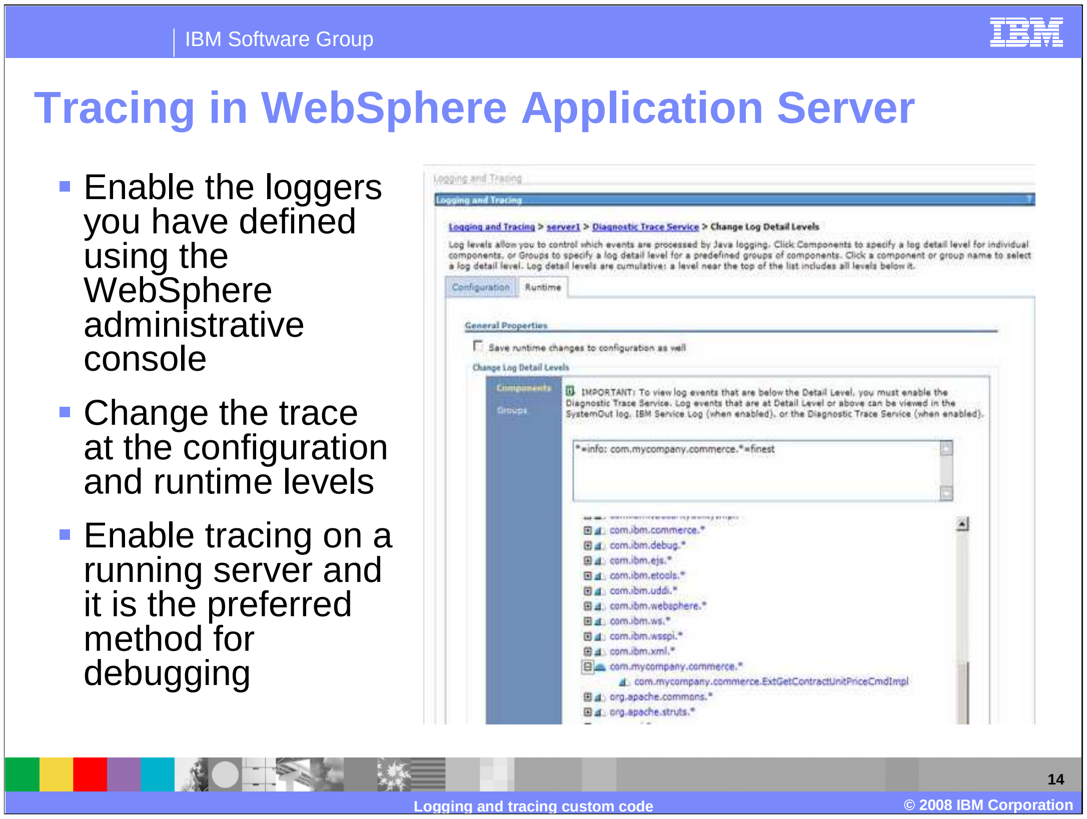
FINE, FINER and FINEST are meant to be used for tracing and debugging.

The different levels can help you prevent the “all or nothing” problem, where your application either gives too much information or no information at all.

One way of deciding which level to use is to consider their impact while the trace is enabled.

To minimize performance degradation, you can organize your messages to use the different levels. For example, use FINER or FINEST for normal messages, but use only FINE for messages where you know a large amount of trace data is generated.

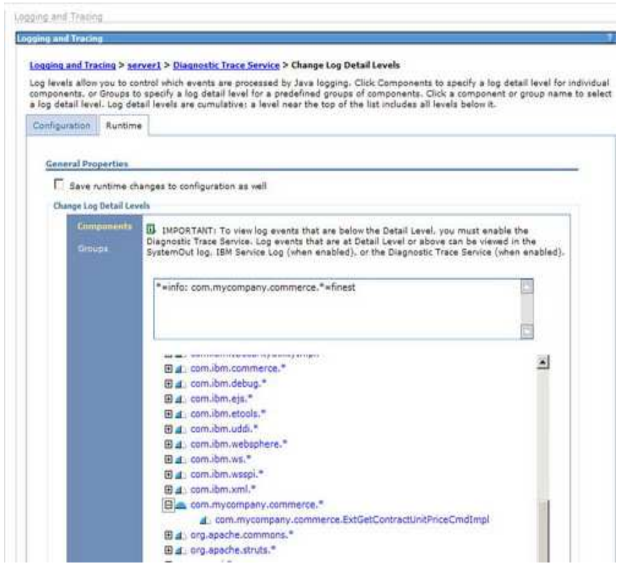
Keep in mind that you can change the trace level for a running server. Because of this, the trace levels should only be used while debugging a particular problem, and the level should be set back to INFO once the problem is resolved.



IBM Software Group

Tracing in WebSphere Application Server

- Enable the loggers you have defined using the WebSphere administrative console
- Change the trace at the configuration and runtime levels
- Enable tracing on a running server and it is the preferred method for debugging



Logging and Tracing

Logging and Tracing > server1 > Diagnostic Trace Service > Change Log Detail Levels

Log levels allow you to control which events are processed by Java logging. Click Components to specify a log detail level for individual components, or Groups to specify a log detail level for a predefined groups of components. Click a component or group name to select a log detail level. Log detail levels are cumulative: a level near the top of the list includes all levels below it.

Configuration | Runtime

General Properties

☐ Save runtime changes to configuration as well

Change Log Detail Levels

Components

Groups

IMPORTANT: To view log events that are below the Detail Level, you must enable the Diagnostic Trace Service. Log events that are at Detail Level or above can be viewed in the SystemOut log, IBM Service Log (when enabled), or the Diagnostic Trace Service (when enabled).

```
*=info: com.mycompany.commerce.*=finest
```

```
com.ibm.commerce.*
com.ibm.debug.*
com.ibm.ejs.*
com.ibm.etools.*
com.ibm.uddi.*
com.ibm.websphere.*
com.ibm.ws.*
com.ibm.wsspi.*
com.ibm.xml.*
com.mycompany.commerce.*
    com.mycompany.commerce.ExtGetContractUnitPriceCmdImpl
org.apache.commons.*
org.apache.struts.*
```

14

Logging and tracing custom code

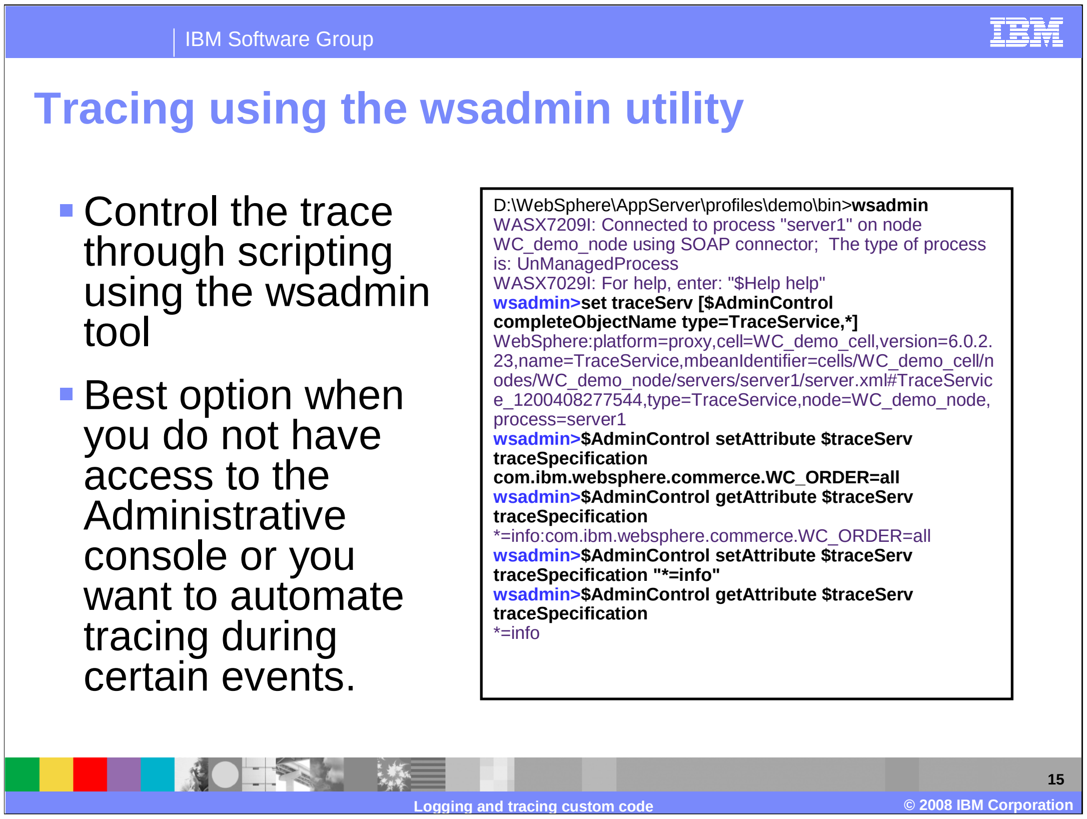
© 2008 IBM Corporation

Using the WebSphere administrative console, you can view and edit the trace specification. The console shows you two tabs, Configuration and Runtime.

Configuration trace takes effect during startup and requires a restart. Runtime trace sets a new trace on a running server.

You can change the current trace specifications without having to restart the server. This technique is the preferred mechanism for debugging.

Configuration should be used with logging levels that are permanent. You should avoid using trace levels in the configuration tab because they might have an impact on the performance of the system.



Tracing using the wsadmin utility

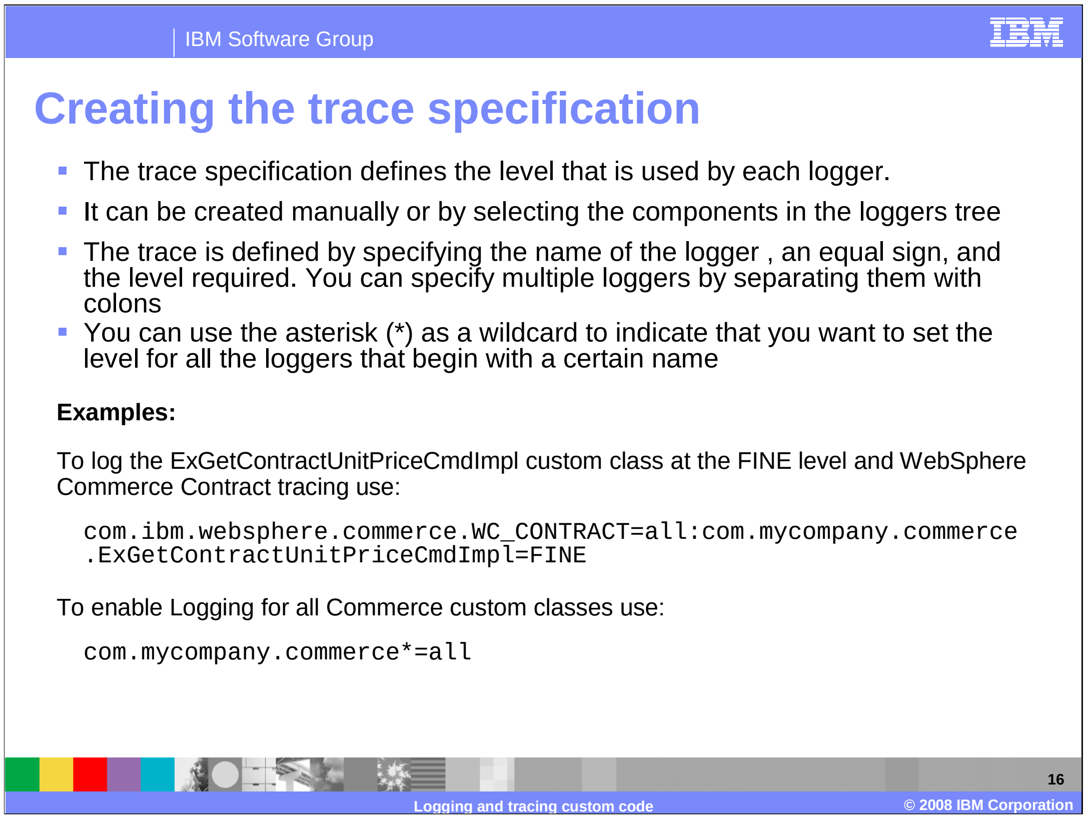
- Control the trace through scripting using the wsadmin tool
- Best option when you do not have access to the Administrative console or you want to automate tracing during certain events.

```
D:\WebSphere\AppServer\profiles\demo\bin>wsadmin
WASX7209I: Connected to process "server1" on node
WC_demo_node using SOAP connector;  The type of process
is: UnManagedProcess
WASX7029I: For help, enter: "$Help help"
wsadmin>set traceServ [$AdminControl
completeObjectName type=TraceService,*]
WebSphere:platform=proxy,cell=WC_demo_cell,version=6.0.2.
23,name=TraceService,mbeanIdentifier=cells/WC_demo_cell/n
odes/WC_demo_node/servers/server1/server.xml#TraceServic
e_1200408277544,type=TraceService,node=WC_demo_node,
process=server1
wsadmin>$AdminControl setAttribute $traceServ
traceSpecification
com.ibm.websphere.commerce.WC_ORDER=all
wsadmin>$AdminControl getAttribute $traceServ
traceSpecification
*=info:com.ibm.websphere.commerce.WC_ORDER=all
wsadmin>$AdminControl setAttribute $traceServ
traceSpecification "*=info"
wsadmin>$AdminControl getAttribute $traceServ
traceSpecification
*=info
```

WebSphere also allows you to control the trace through scripting using the wsadmin tool. This is the best option when you do not have access to the administrative console or you want to automate tracing during certain events.

You can find more details in the developerWorks article **Using logging and tracing in the WebSphere Commerce custom code.**

See the References slide at the end of the presentation.



Creating the trace specification

- The trace specification defines the level that is used by each logger.
- It can be created manually or by selecting the components in the loggers tree
- The trace is defined by specifying the name of the logger , an equal sign, and the level required. You can specify multiple loggers by separating them with colons
- You can use the asterisk (*) as a wildcard to indicate that you want to set the level for all the loggers that begin with a certain name

Examples:

To log the ExGetContractUnitPriceCmdImpl custom class at the FINE level and WebSphere Commerce Contract tracing use:

```
com.ibm.websphere.commerce.WC_CONTRACT=all:com.mycompany.commerce
.ExGetContractUnitPriceCmdImpl=FINE
```

To enable Logging for all Commerce custom classes use:

```
com.mycompany.commerce*=all
```

To enable or disable logging, update the trace specification that defines the levels that are used for each logger.

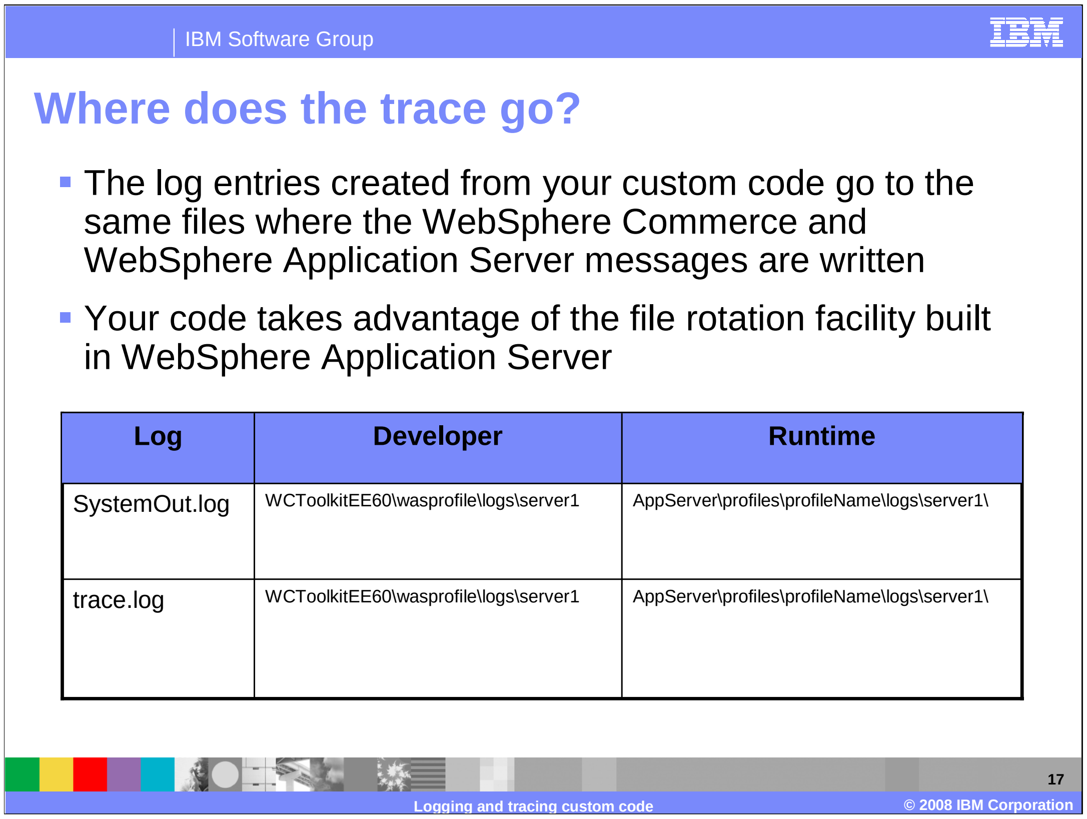
The trace is defined by specifying the name of the logger , an equal sign, and the level required. You can specify multiple loggers by separating them with colons. You can use the asterisk to indicate that you want to set the level for all the loggers that begin with a certain name.

This slide shows two examples.

The first example shows how to enable trace specifications for WebSphere Commerce.

The second example shows you how to specify a trace using the asterisk wildcard notation.

Assuming that you followed the convention and named the loggers using the class names, this should enable tracing for the all of your custom code.


IBM Software Group

Where does the trace go?

- The log entries created from your custom code go to the same files where the WebSphere Commerce and WebSphere Application Server messages are written
- Your code takes advantage of the file rotation facility built in WebSphere Application Server

Log	Developer	Runtime
SystemOut.log	WCToolkitEE60\wasprofile\logs\server1	AppServer\profiles\profileName\logs\server1\
trace.log	WCToolkitEE60\wasprofile\logs\server1	AppServer\profiles\profileName\logs\server1\

17

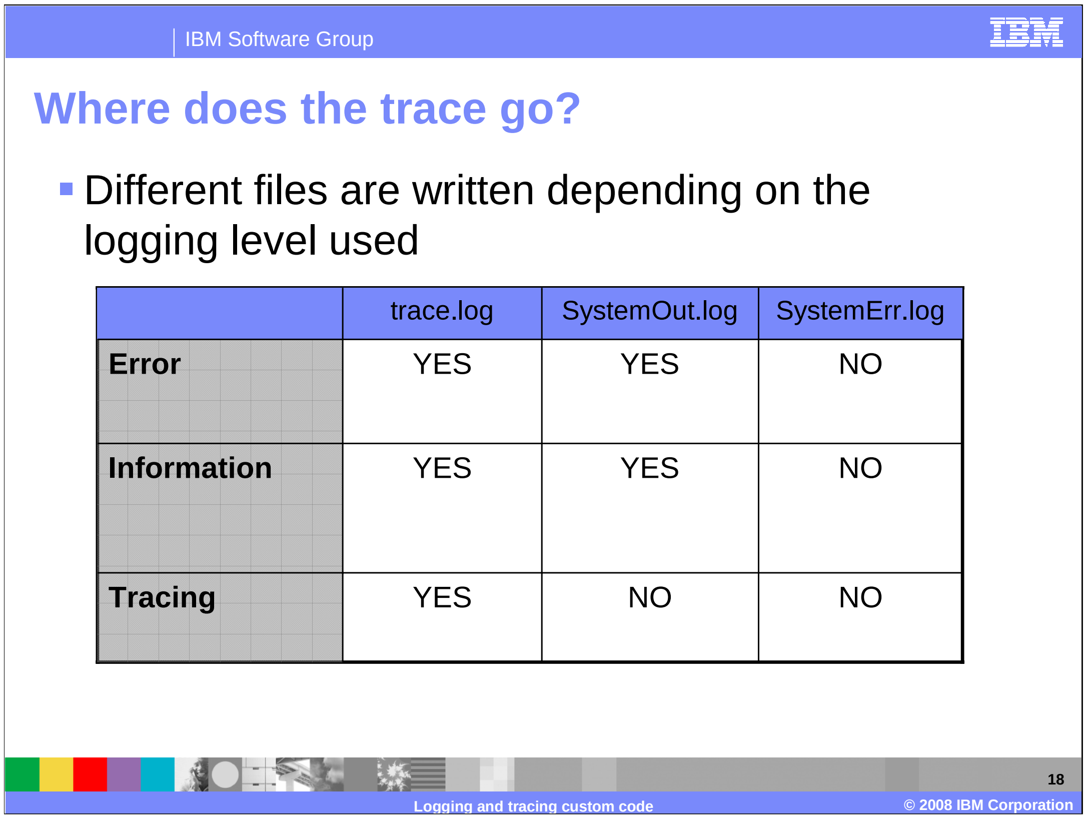
Logging and tracing custom code

© 2008 IBM Corporation

When you enable tracing for your custom code, the log entries are written to the files you are already familiar with, namely SystemOut.log and trace.log.

Both custom and WebSphere tracing go to the same file, making it easier to debug a particular problem that requires multiple traces to be enabled.

Also, by using the WebSphere files, your trace is taking advantage of the file rotation facility offered by the WebSphere Application Server.



Where does the trace go?

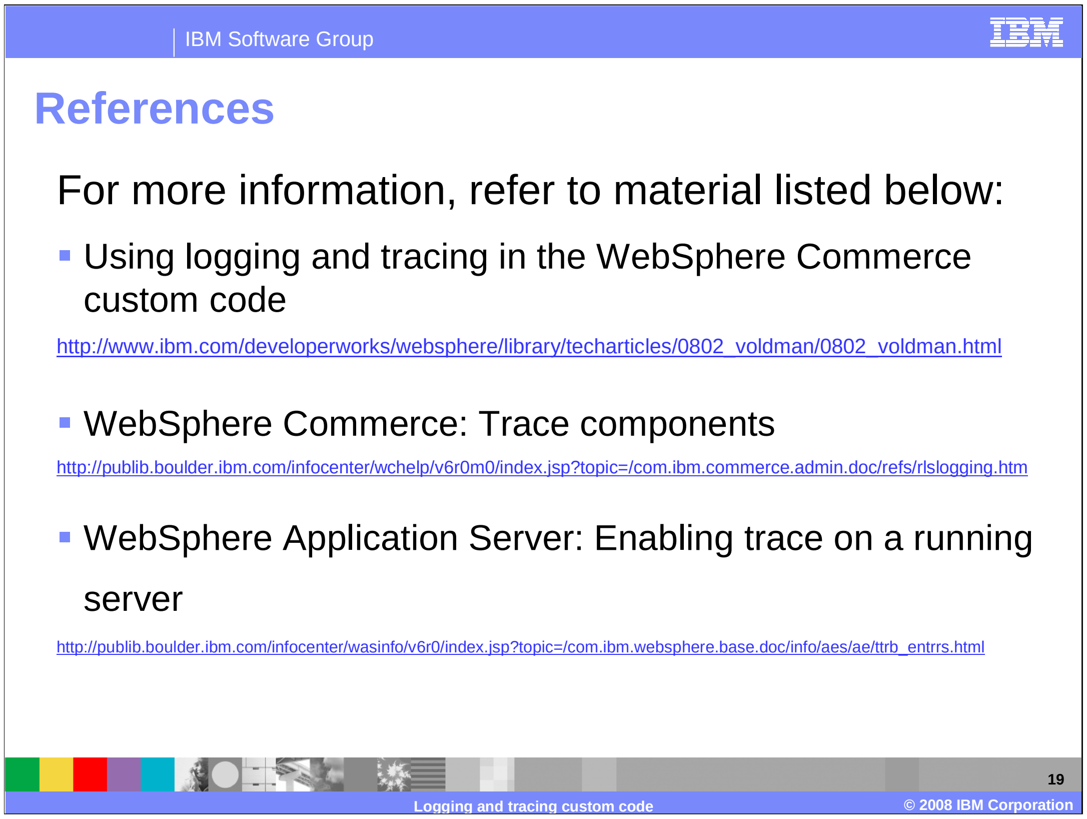
- Different files are written depending on the logging level used

	trace.log	SystemOut.log	SystemErr.log
Error	YES	YES	NO
Information	YES	YES	NO
Tracing	YES	NO	NO

Depending on the logging level used by your messages, the entries go to different files

Error and Warning, and Information messages are logged in both, trace.log and SystemOut.log.

Trace messages are only printed in trace.log but not in SystemOut.log



References

For more information, refer to material listed below:

- Using logging and tracing in the WebSphere Commerce custom code

http://www.ibm.com/developerworks/websphere/library/techarticles/0802_voldman/0802_voldman.html

- WebSphere Commerce: Trace components

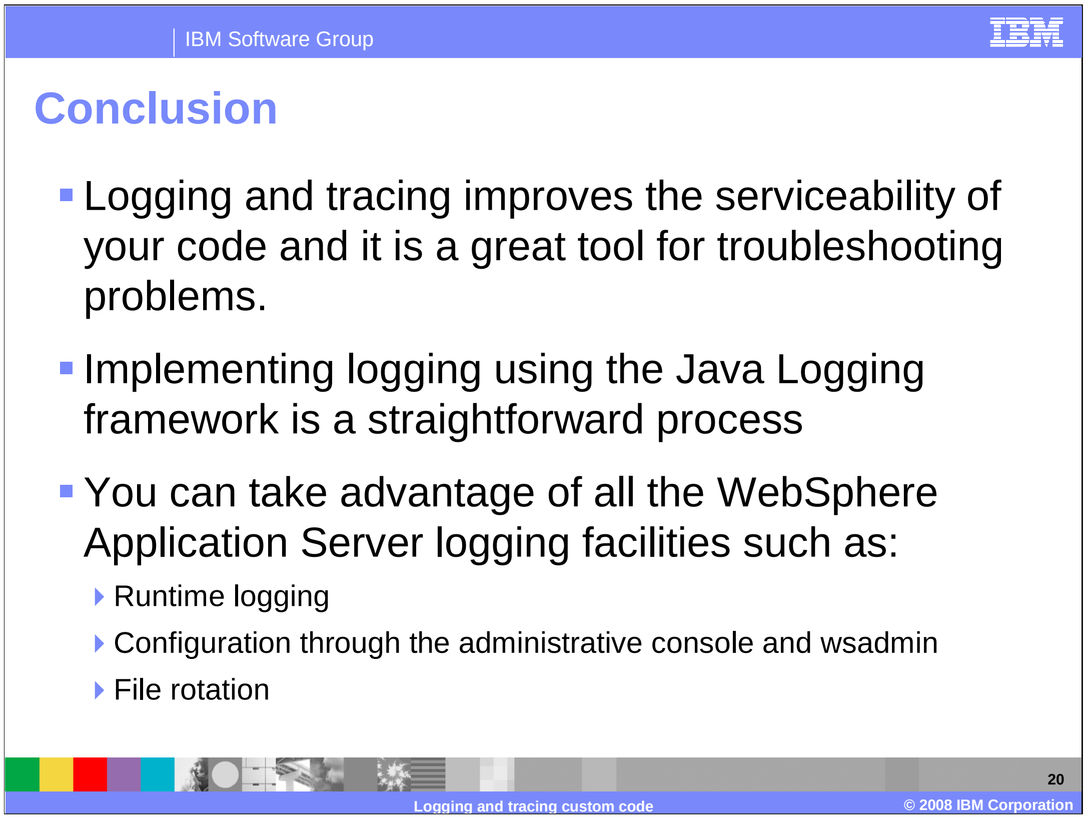
http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=/com.ibm.commerce.admin.doc/refs/rlslogging.htm

- WebSphere Application Server: Enabling trace on a running server

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/ttrb_entrrs.html

This slide contains useful links from the WebSphere Commerce and WebSphere Application Server Information Centers.

The first reference "Using logging and tracing in the WebSphere Commerce custom code" is a developerWorks article that this presentation was based upon. Refer to this article for more detailed information on this topic.

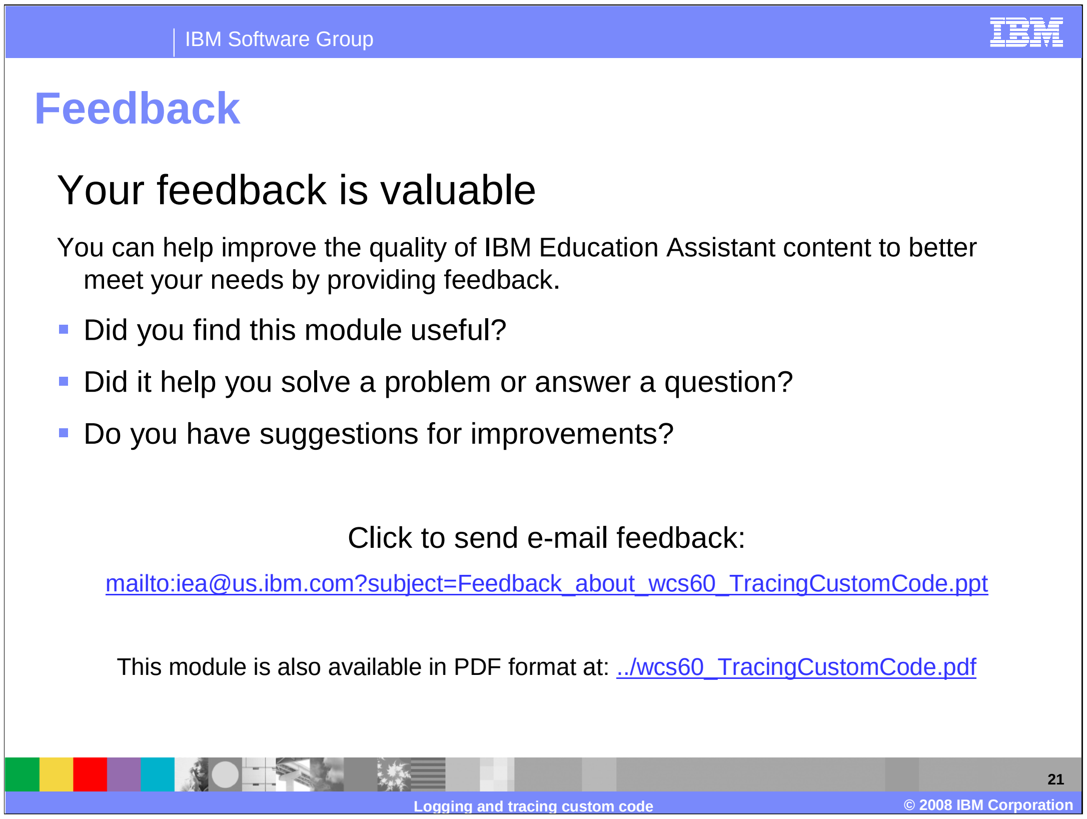


Conclusion

- Logging and tracing improves the serviceability of your code and it is a great tool for troubleshooting problems.
- Implementing logging using the Java Logging framework is a straightforward process
- You can take advantage of all the WebSphere Application Server logging facilities such as:
  - Runtime logging
  - Configuration through the administrative console and wsadmin
  - File rotation

Implementing logging and tracing in WebSphere Application Server V6 using the Java Logging framework is a straightforward process.

This presentation showed, by implementing Java logging and following a few best practices, you can improve the serviceability of your code and your ability to troubleshoot problems.


IBM Software Group

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_wcs60_TracingCustomCode.ppt

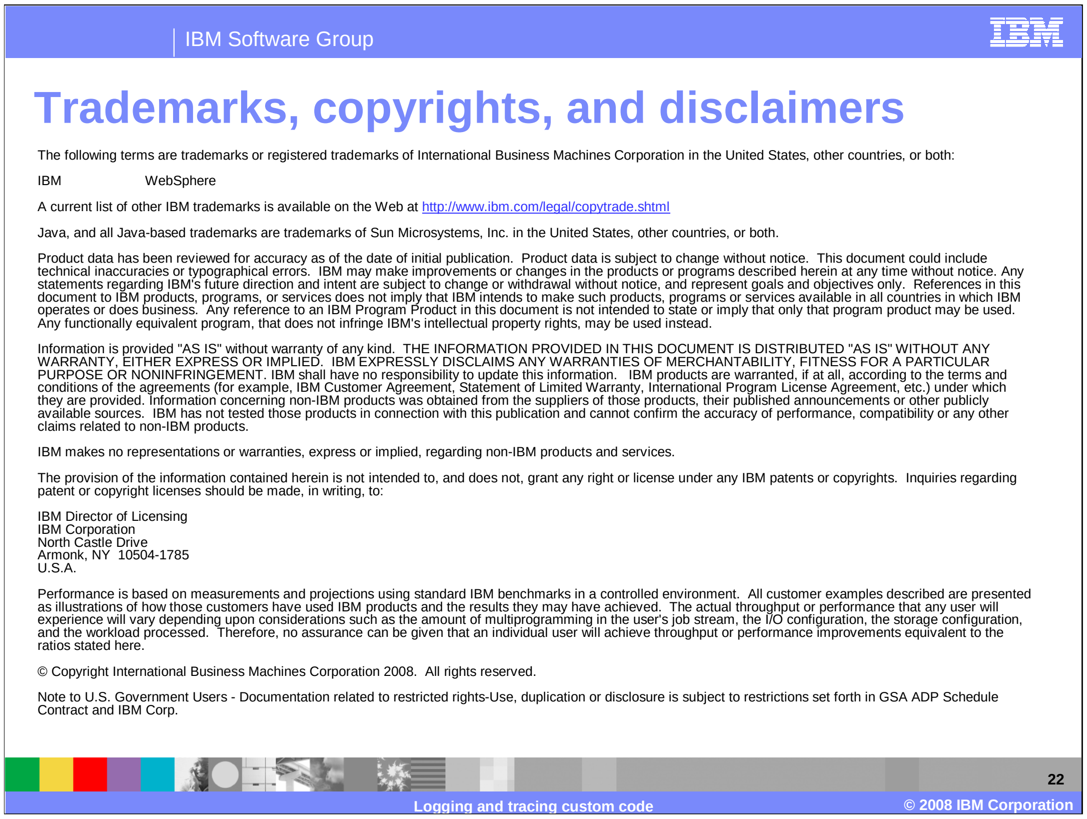
This module is also available in PDF format at: ../wcs60_TracingCustomCode.pdf

21

Logging and tracing custom code

© 2008 IBM Corporation

You can help improve the quality of IBM Education Assistant content by providing feedback.



Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2008. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.