An Introduction to LogBox

www.coldbox.org

Covers up to version 1.0.0

Contents

What is LogBox	1
Installing LogBox	1
Configuration	2
API Reference	2
Log Levels	2
Categories	2
Accessing Loggers inside ColdBox applications	3
Customizing Your Logging Categories	3
Custom Log Formats	3
Custom Appenders	3

LogBox is an enterprise ColdFusion (CFML) logging library designed to give you flexibility, simplicity and power when logging or tracing is needed in your applications.

CFLog is fine if you have incredibly simple logging requirements, but LogBox is designed to be flexible to the needs of complex applications and dynamically configurable to control what gets logged and where without changing the code in your app. LogBox comes bundled with the ColdBox Platform, but also can be used as a standalone library in any CFML application or framework.

LogBox is built around the abstraction that each message in your application gets emitted as a "log event" with a given severity (INFO, WARN, ERROR, etc). LogBox helps you filter and redirect those messages to any location (or appender) that you wish. That means your log events can find their way to text files, databases, E-mail, Message Queues, or the good old CF logs.

You have the power to turn off or restrict logs to a specific appender location based on what part of your application the event is coming from and the severity of the message. You can also use environment overrides to alter what's logged (and where) on your development server as opposed to production all via configuration. Your code stays the same on every server!

INSTALLING LOGBOX

ColdBox-Bundled

If you are using the ColdBox Platform, you don't need to do anything--LogBox is included for you and the framework creates it for you. The default config is a CFC called LogBox.cfc and located inside of the "config" folder.

`/config/LogBox.cfc`

To download the ColdBox MVC platform (with bundled LogBox), visit `http://www.coldbox.org/download` and snag the latest version.

Standalone

If you would like to use LogBox on its own, visit the download page `http://www.coldbox.org/download` and download the standalone LogBox library. After unzipping the library, the easiest way to install it is to place the "logbox" folder in your webroot. A more secure method is to leave that folder outside the webroot and create a "logbox" ColdFusion mapping that points to it.

When your application starts up, create and store an instance of logbox. `system.logging.LogBox` and pass in your created configuration file:

```
LogBoxConfig = new logbox.system.logging.config
    .LogBoxConfig( CFCConfigPath="path.to.config" );

application.LogBox = new logbox.system.logging
    .LogBox( LogBoxConfig );
```

Change any paths with `logbox.system` to `coldbox.system` if you are using LogBox inside of the ColdBox MVC Platform.

CONFIGURATION

Appenders

One of the most core concepts of LogBox is appenders. These are components that store your log messages somewhere. The API for an appender is pretty basic-- it receives a message, and as long as that message falls within the range of severities the appender has been configured to handle, the message is stored. Appenders are named from the concept that they originally would "append" the message to a text file, but they can do so much more than text files. There are a collection of out-of-the box appenders, plus you are can write your own!

- **AsyncDBAppender** - Inserts records into a table asynchronously
- **AsyncFileAppender** - Inserts records into a file asynchronously
- **AsyncRollingFileAppender** - Inserts records into a file asynchronously with archiving
- **CFAppender** - Appends to CF's logs. Similar to cflog tag
- **ColdBoxTracerAppender** - Appends to ColdBox's Debugger
- **ConsoleAppender** - Send message to Java's standard output
- **DBAppender** - Insert records into a table
- **DummyAppender** - Inserts nowhere
- **EmailAppender** - Send messages to an E-mail address
- **FileAppender** - Appends to text file
- **RollingFileAppender** - Appends to text file with rotations
- **ScopeAppender** - Append messages to an in-memory array
- **SocketAppender** - Uses Java to send the message to any host:port
- **TracerAppender** - Output messages in ColdFusion cftracer debugging

Before LogBox will emit your messages, you must configure one or more appenders and assign them to logging categories (See categories section). You can assign a list of appender names, or use an asterisk (*) to assign all appenders. This is easy as the configuration is declared as a struct literal like so:

/config/LogBox.cfc

```
function configure() {
    logBox = {
        // Register Appenders
        appenders = {
            MyAsyncFile = {
                class='logbox.system.logging.appenders.
AsyncRollingFileAppender',
                properties={
                    filePath=expandPath( '/logs' )
                }
            },
            myEmail = {
                class='logbox.system.logging.appenders.
EmailAppender',
                properties = {
                    from='admin@example.com',
                    to='alerts@example.com'
                }
            }
        },
        // Root Logger. Assign all appenders
        root = { appenders='*' }
    };
}
```

API REFERENCE

Now that we have the LogBox library created and configured with appenders, we can use it by asking for a logger. This will get the default logger, or **root** logger from LogBox and log three messages of different severities. Each message is a string and can be accompanied by a second parameter of extra info of your choice.

Note: The `extraInfo` parameter can be complex such as a struct of data.

```
logger = application.LogBox.getRootLogger();

logger.info( 'Application Initialized.' );
logger.warn( 'This method is deprecated.' );
logger.error( 'Error placing order', cfcatch);
```

LOG LEVELS

Every log message has a severity, or level associated with it. Here are the possible log levels. An integer number is associated with each level to establish a hierarchy of importance.

Severity	Integer Level
Off	-1
Fatal	0
Error	1
Warn	2
Info	3
Debug	4

We can change our root logger from above to ignore unimportant messages and only log ones with a level of Error or Fatal.

```
root = { levelMin='FATAL', levelMax='ERROR', appenders='*' }
```

CATEGORIES

LogBox has a default logger called the "root" logger, but you can create as many named loggers as you want. This allows you to section off log messages by categories to give you fine-grain control of your logging. Logger names, or categories, can be anything you want but the common convention is to name them after the CFC file that they are logging from. The easiest way to do this is to pass the "**this**" reference into the `getLogger()` method.

Here is a code sample that shows a simple component that accepts LogBox as a constructor argument and creates and stores logger named after itself. The logger would be named "`com.foo.myService`".

/com/foo/myService.cfc

```
component {
    function init( LogBox ) {
        variables.log = arguments.LogBox.getLogger( this );
        return this;
    }
}
```

It is also valid to make up your own logger names which could be useful in a legacy app that doesn't use many CFCs.

```
log = application.LogBox.getLogger( 'SecurityLogger' );
```

ACCESSING LOGGERS INSIDE COLDBOX APPLICATIONS

If you are using LogBox from inside of ColdBox, every handler, interceptor, and plugin automatically has a reference to `variables.logBox`, and `variables.log` - a logger object created and ready to go for you.

For models inside of ColdBox or any application using WireBox, you can autowire your logger in easily like so:

myService.cfc

```
component {
    property name='log' inject='logbox:logger:{this}';

    function doSomething() {
        log.info( 'Doing something!' );
    }
}
```

CUSTOMIZING YOUR LOGGING CATEGORIES

Earlier, we configured the appenders and log levels for our root logger, which all loggers inherit from, but we also have the ability to assign different appenders and log levels to each logging category as well to truly customize which messages get logged from which parts of our application.

This snippet of config will turn off debugging messages from the ColdBox framework and will log message from any component in the `model.security` package to the myEmail appender only.

```
// Root Logger
root={ levelMin='FATAL', levelMax='DEBUG', appenders='*' },

// Granular Categories
categories = {
    'coldbox.system'={ levelMin='FATAL', levelMax='INFO',
                       appenders='*' },
    'model.security'={ levelMax='DEBUG', appenders='myEmail' }
}
```

Note: LogBox treats logging categories as a hierarchy delimited by periods, so if there is config for `model`, `model.security`, and `model.security.UserService`, the most specific match will be used. All logging categories inherit from the "root" logger.

CUSTOM LOG FORMATS

What if you like the file appender, but want to change the log file format? You can override the layout of most log appenders with a simple CFC that contains a `format()` method. Just specify the path to your layout CFC in the LogBox config for that appender. Every time a log event is sent, the appender will ask the layout CFC's `format()` method to produce a string out of the logEvent object.

myLayout.cfc

```
component {
    string function format( logEvent ) {
        return 'Your server just called and said,
                            "#logEvent.getMessage()#".';
    }
}
```

In your config:

```
MyFile = {
    class='logbox.system.logging.appenders.FileAppender',
    layout='path.to.myLayout'
}
```

More Info: http://wiki.coldbox.org/wiki/LogBox.cfm#Creating_a_Custom_Layout

CUSTOM APPENDERS

If you need more flexibility than what is provided with a custom layout, or you want to persist log messages to a completely new storage medium, you can write a custom appender of your own. Create a CFC that extends `logbox.system.logging.AbstractAppender`. Minimally your component needs to implement a `logMessage()` method that receives logEvent object and is responsible for persisting it.

myAppender.cfc

```
component extends='coldbox.system.logging.AbstractAppender' {
    function logMessage( logEvent ) {
        writeLog( logEvent.getMessage() );
    }
}
```

Note: ForgeBox has a category for sharing appenders with the community. Check there before writing one to see if someone else has already done the work.

`http://www.coldbox.org/forgebox/type/logbox`

More Info: http://wiki.coldbox.org/wiki/LogBox.cfm#Creating_Custom_Appenders

The LogBox framework is completely documented in our online wiki located at http://wiki.coldbox.org/wiki/LogBox.cfm

For API docs to see class definitions and method signatures, please visit the API docs located at http://www.coldbox.org/api

We have an active Google Group with hundreds of subscriber located at

http://groups.google.com/group/coldbox

Our official code repository is on GitHub. Please favorite us and feel free to fork and submit pull requests.

https://github.com/ColdBox/coldbox-platform

LOGBOX
By ColdBox

An Introduction to LogBox

Ortus Solutions
A Software Revolution

LogBox is professional open source backed by Ortus Solutions, who provides training, support & mentoring, and custom development.

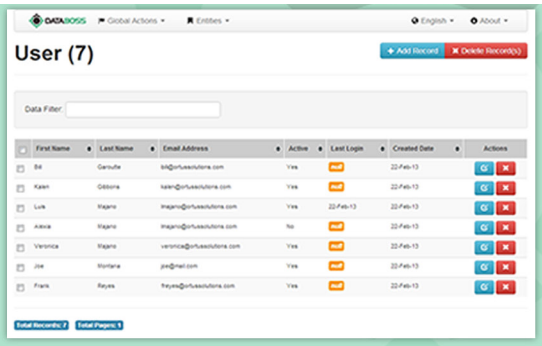
Support Program

The Ortus Support Program offers you a variety of Support Plans so that you can choose the one that best suit your needs. Subscribe to your plan now and join the Ortus Family!

With all of our plans you will profit not only from discounted rates but also receive an entire suite of support and development services. We can assist you with sanity checks, code analysis, architectural reviews, mentoring, professional support for all of our Ortus products, custom development and much more!

For more information visit
www.ortussolutions.com/services/support

Our Plans	M1	Entry	Standard	Premium	Enterprise
Price	**$199**	**$2099**	**$5699**	**$14099**	**$24599**
Support Hours	2 4 tickets	10 20 tickets	30 60 tickets	80 160 tickets	150 300 tickets
Discounted Rate	**$185/hr**	**$180/hr**	**$175/hr**	**$170/hr**	**$160/hr**
Renewal Price	**$199/month**	**$1800/year**	**$5250/year**	**$13600/year**	**$2400/year**
Phone/Online Appointments	✔	✔	✔	✔	✔
Web Ticketing System	✔	✔	✔	✔	✔
Architectural Reviews	✔	✔	✔	✔	✔
Hour Rollover		✔	✔	✔	✔
Custom Development		✔	✔	✔	✔
Custom Builds & Patches			✔	✔	✔
Priority Training Registration			✔	✔	✔
Development & Ticket Priority			✔	✔	✔
Response Times	1-5 B.D.	1-3 B.D.	1-2 B.D.	< 24 hr	< 12 hr
Books, Trainings, Product Discounts	0%	5%	10%	15%	20%
Free Books	0	0	1	3	5



"Build dynamic ORM administrators in seconds"

DATABOSS
Dynamic Administrator

www.data-boss.com

www.coldbox.org | www.ortussolutions.com

First Edition - March 2014