Advanced ColdFusion: Error Handling

Mosh Teitelbaum
mosh.teitelbaum@evoch.com
evoch, LLC

Why Handle Errors?

- Professional

  Handling errors prevents users from seeing ugly and unprofessional error messages.

- Recoverability

  Handling errors allows you to gracefully recover and/or redirect users to other resources.

- Auditing

  Handling errors allows you to be immediately informed of site errors and to log errors in a useful way.

What Are Errors?

- Unforeseen

  Errors are results produced by the code that are generally unforeseen, unanticipated, and/or unplanned for by the developer.

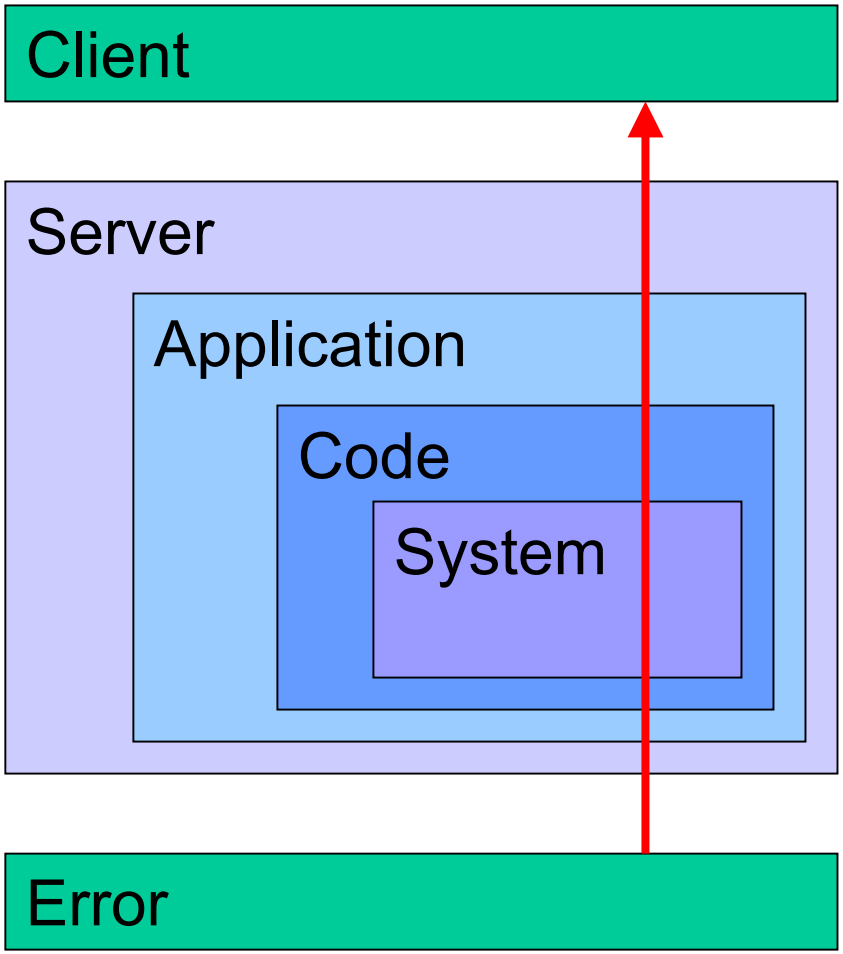
- A Fact Of Life

  Any sizable amount of code will contain bugs. The larger the codebase, the larger the amount of bugs.

- Manageable

  Errors can be managed and can even be used to our advantage.

How Are Errors Handled?



From the ground up:

1st – At the code level

2nd – At the application level

3rd – At the server level

If left unhandled, the user sees the raw (ugly) error message.

Error Classifications

Classification	Description
Logic	Faulty design and/or logic within your code.
Syntax	Syntactical errors including misspellings, invalid or missing parameters, etc.
Runtime	Errors that occur due to runtime conditions (faulty input data, invalid configuration, etc.)
Validation	A form of runtime error. Results from server-side form validation.
System	Errors that occur as a result of system problems (database is down, unresponsive remote host, etc.)
Request	Errors that occur as a result of invalid requests (404 errors, invalid HTTP headers, etc.)

Logic Errors

Logic errors usually do not result in an error message or a thrown exception, but cause the program to malfunction nonetheless. Logic errors are usually only caught through thorough testing.

Example:

```
<CFSET counter = 1>
<CFLOOP CONDITION = “counter LTE 5">
        <CFOUTPUT>#counter#<BR></CFOUTPUT>
</CFLOOP>
```

Syntax Errors

Syntactical errors occur as a result of misspellings, invalid parameters, invalid data types, and other problems with language syntax. These errors can be caught via application-level and/or server-level error handlers.

Example:

```
<CFOUTPUT QUERY="someQuery">
        <CFOUTPUT>#someQuery.someField#<BR></CFOUTPUT>
</CFOUTPUT>
```

Runtime Errors

Runtime errors, or *Exceptions*, are those that occur after syntax checking and are a result of some unforeseen condition such as server-side validation errors, data type mismatches, out of scope data, etc. These errors can be caught via code-level, application-level, and/or server-level error handlers.

Example:

```
<CFPARAM NAME="age" DEFAULT="Mosh">

<CFIF Int(age) LT 18>
        You are too young to enter this website.
</CFIF>
```

Validation Errors

Runtime errors that occur when ColdFusion's server-side form validation catches a problem with submitted form data. These errors can only be caught via application-level error handlers.

Example:

```
<FORM ACTION="action.cfm" METHOD="Post">
        <INPUT TYPE="Text" NAME="Age" VALUE="thirty">
        <INPUT TYPE="Hidden" NAME="Age_integer" VALUE="Err Msg">
</FORM>
```

System Errors

System errors occur as a result of some sort of system (not code) failure including invalidly configured servers, inaccessible databases, unavailable web resources, and file system errors. These errors can be caught via code-level, application-level, and/or server-level error handlers.

Example:

```
<CFHTTP URL="http://www.nonExistentServer.com/" METHOD="GET">
</CFHTTP>
```

Request Errors

Request errors occur as a of a client error, not a server or code error. These client errors include requests for invalid resources (404 errors) and invalid HTTP headers. These errors can only be caught via server-wide error handlers.

Example:

http://www.validServer.com/invalidResource.cfm

ColdFusion Error Handling

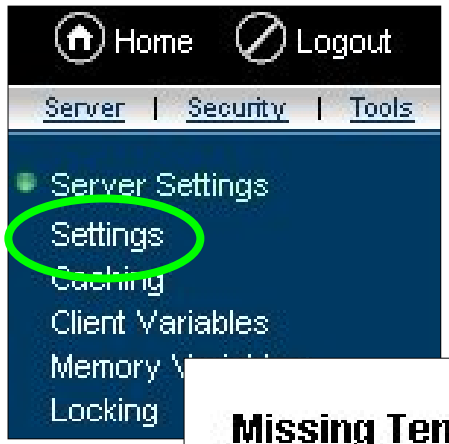
- Server-Wide Error Handler
  Template that is executed whenever an error is not handled at the application or code level.

- Server-Wide Missing Template Handler
  Template that is executed whenever ColdFusion cannot find a requested template.

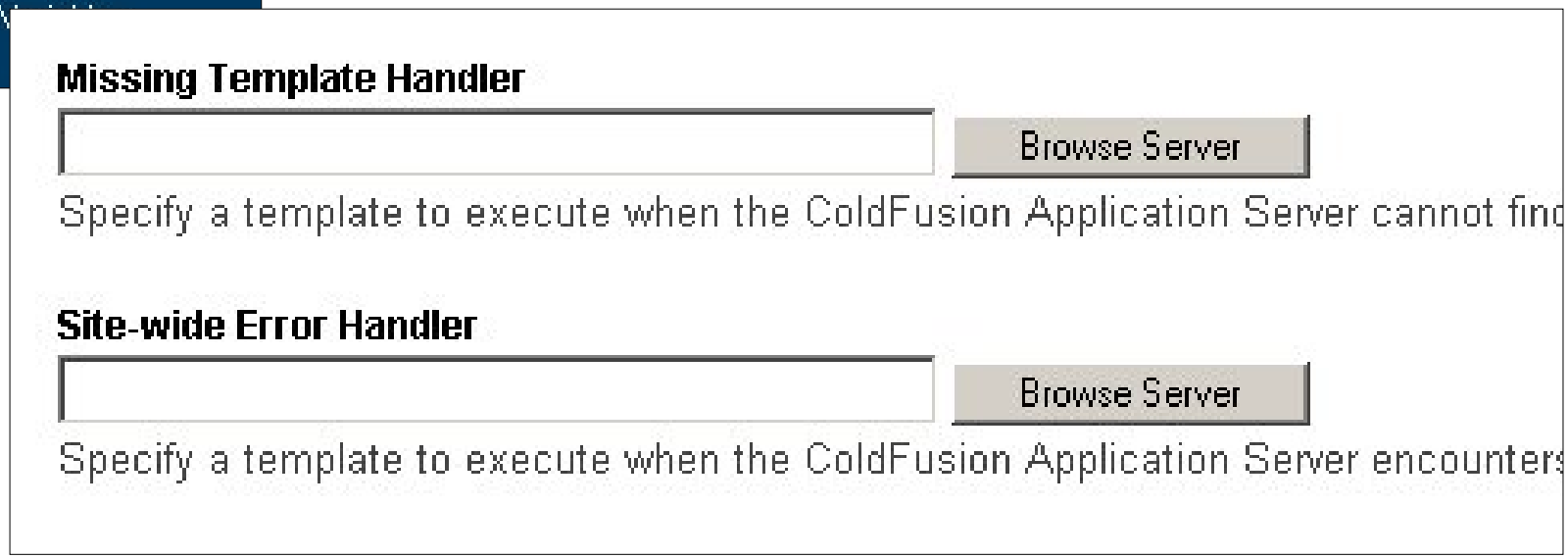
- CFERROR
  Allows specification of application-wide error handling templates.

- CFTRY / CFCATCH / CFTHROW / CFRETHROW
  Allows for handling of errors at the code level.

Server-Wide Error Handler Templates



1. Click on “Server Settings | Settings” in the ColdFusion Administrator
2. Complete the input fields at the bottom of the page



<CFERROR>

Allows specification of application-wide error handling templates. Templates have access to a special scope, ERROR, that contains information about the error.

Attribute	Description
Type	Required. The type of error that the custom error page handles. "request" or "validation" or "monitor" or "exception"
Template	Required. The relative path to the custom error page.
MailTo	Optional. The e-mail address of the administrator to notify of the error. The value is available to your custom error page in the MailTo property of the error object.
Exception	Required. Type of exception. Required if type = "exception" or "monitor".

<CFERROR TYPE="Request">

<CFERROR TYPE="Request" TEMPLATE="err.cfm" MAILTO="Email">

- Catches all errors except server-side validation errors
- Specified template ignores all CFML code – only ERROR variables can be dynamically displayed. ERROR variables do not need to be enclosed by <CFOUTPUT> tags
- Used to catch errors caused by any other application-wide error handlers
- Allows you to personalize your error message

TYPE="Request" Error Variables

Variable	Description
Browser	Client's User-Agent
DateTime	Date and time when the error occurred
Diagnostics	Detailed error diagnostics
GeneratedContent	The content generated before the error occurred
HTTPReferer	URL from which this page was accessed
MailTo	Value of the <CFERROR> MailTo attribute
QueryString	URL's query string
RemoteAddress	IP Address of the client
Template	Template being executed when the error occurred

<CFERROR TYPE="Validation">

<CFERROR TYPE="Validation" TEMPLATE="err.cfm" MAILTO="Email">

- Catches all server-side validation errors
- Specified template ignores all CFML code – only ERROR variables can be dynamically displayed. ERROR variables do not need to be enclosed by <CFOUTPUT> tags
- Must be specified in Application.cfm
- Allows you to personalize your form validation error messages.

TYPE="Validation" Error Variables

Variable	Description
InvalidFields	An unordered list of validation errors that occurred
MailTo	Value of the <CFERROR> MailTo attribute
ValidationHeader	Default ColdFusion text used for header of validation error messages
ValidationFooter	Default ColdFusion text used for footer of validation error messages

<CFERROR TYPE="Exception">

<CFERROR TYPE="Exception" TEMPLATE="err.cfm" EXCEPTION="Type" MAILTO="Email">

- Catches all runtime errors, except server-side validation errors, that are not caught at the code level
- Like TYPE="Request" but can contain CFML code
- Can create new errors/exceptions
- Can specify multiple such <CFERROR> tags to personalize error messages by exception type.

TYPE="Exception" Error Variables

Same as TYPE=“Request” plus the following:

Variable	Description
Detail	Value of the <CFTHROW> Detail attribute
ErrorCode	Value of the <CFTHROW> ErrorCode attribute
ExtendedInfo	Value of the <CFTHROW> ExtendedInfo attribute

<CFERROR TYPE="Monitor">

<CFERROR TYPE="Monitor" TEMPLATE="err.cfm" MAILTO="Email">

- Executed whenever a runtime error, caught at the code level, occurs. This template is processed and then control returns to the page where the exception occurred.
- Can contain CFML code
- Can create new errors/exceptions
- Used to log exceptions or to perform any other action that should occur whenever an exception is caught and handled.

TYPE="Monitor" Error Variables

Same as TYPE=“Exception” plus the following:

Variable	Description
Message	Value of the <CFTHROW> Message attribute
Type	Value of the <CFTHROW> Type attribute

<CFTRY>

Allows for code-level exception handling. Contains code that may cause exceptions and one or more <CFCATCH> blocks that are used to handle those exceptions.

Attribute	Description

<CFCATCH>

Specifies the type of exception that it catches and contains code used to handle the caught exception. Nested within <CFTRY>.

Attribute	Description
Type	Optional. Specifies the type of exception to be handled by the cfcatch block. The following lists basic exception types you can use: • Application • Database • Template • Security • Object • MissingInclude • Expression • Lock • Custom_type • Any (default) • SearchEngine (new in CFMX)

<CFTRY> / <CFCATCH>

```
<CFTRY>
        Code that may throw an exception

        <CFCATCH TYPE="Exception">
                Code that handles specified exception type
        </CFCATCH>

        <CFCATCH TYPE="Any">
                Code that handles any other exception type
        </CFCATCH>
</CFTRY>
```

CFCATCH Variables

<CFCATCH> blocks have access to a special scope, CFCATCH, that contains information about the caught exception. All exception types share the following CFCATCH variables. Other variables are defined only for specific exception types.

Variable	Description
Detail	Detailed message about the error
ErrorCode	Value of the <CFTHROW> ErrorCode attribute
ExtendedInfo	Value of the <CFTHROW> ExtendedInfo attribute
Message	Simple message about the error
Type	Type of error

<CFTHROW>

Allows you to throw custom exception types. This is usually done when consolidating your exception handling in one or more files specifically created for handling exceptions.

Variable	Description
Detail	Optional. Detailed information about the custom exception
ErrorCode	Optional. An error code identifying the custom exception
ExtendedInfo	Optional. Extended information about the custom exception
Message	Optional. A message about the custom exception
Type	Optional. A name for the custom exception
Object	Optional. Throws a Java exception. New in CFMX.

Throwing Custom Exception Types

```
<CFTHROW TYPE="com.evoch.myException" MESSAGE="Oops!">
```

- If TYPE is not specified, it defaults to "Application"
- If custom type is a dot-notation series of strings, ColdFusion will try to match the complete string and, if unable to, will keep generalizing the string until it finds a match or runs out of strings. I.e., ColdFusion will search in this order:
  - com.evoch.myException
  - com.evoch
  - com
  - Any
- This allows for the definition of a rich exception model

Throwing Java Exception Types

```
<CFTHROW OBJECT="#javaExceptionObject#">
```

- Before throwing a Java exception, you must first instantiate an instance of a valid Java exception class. For example:

```
<CFOBJECT TYPE="java" ACTION="create"
        CLASS="jException" NAME="javaExceptionObject">
```

- Cannot be used with any other attributes of the <CFTHROW> tag

<CFRETHROW>

Allows you to re-throw the current exception from within a CFCATCH block. This is usually done when an exception is caught within a CustomTag or CFC to allow the calling template to catch and handle the exception.

Variable	Description

Trick: Accessing ERROR from CFCATCH

Normally, the ERROR scope is only available in error handling templates. CFCATCH blocks are limited to the variables in the CFCATCH scope. But, if you specify a <CFERROR TYPE="Monitor"> in your Application.cfm file, you can access the ERROR scope from your CFCATCH blocks.

```
<CFERROR TYPE="Monitor" TEMPLATE="blank.cfm">


<CFTRY>
        <CFCATCH TYPE="Any">
                <CFOUTPUT>#Error.Diagnostics#</CFOUTPUT>
        </CFCATCH>
</CFTRY>
```

Resources

- Macromedia LiveDocs – http://livedocs.macromedia.com/coldfusion/6.1/htmldocs/errors.htm#wp1096654
- Exception Handling With CFML – http://www.how2cf.com/files/papers/exceptions.pdf
- Just about any Ben Forta book on ColdFusion – http://www.forta.com/books/

Closing

- Questions?

- Contact Info

Mosh Teitelbaum
evoch, LLC
mosh.teitelbaum@evoch.com
http://www.evoch.com/

- Extras
  - ERROR Variables
  - Advanced Exception Types

Extras: ERROR Variables

Browser	Client's User-Agent
DateTime	Date and time when the error occurred
Detail	Value of the <CFTHROW> Detail attribute
Diagnostics	Detailed error diagnostics
ErrorCode	Value of the <CFTHROW> ErrorCode attribute
ExtendedInfo	Value of the <CFTHROW> ExtendedInfo attribute
GeneratedContent	The content generated before the error occurred
HTTPReferer	URL from which this page was accessed
InvalidFields	An unordered list of validation errors that occurred
MailTo	Value of the <CFERROR> MailTo attribute
Message	Value of the <CFTHROW> Message attribute
QueryString	URL's query string
RemoteAddress	IP Address of the client
RootCause	Java exception reported by the JVM as the cause of the "root cause" of the exception. New in CFMX
TagContext	Array of structures containing information for each tag on the stack (currently open tags).
Type	Value of the <CFTHROW> Type attribute
ValidationHeader	Default ColdFusion text used for header of validation error messages
ValidationFooter	Default ColdFusion text used for footer of validation error messages

Extras: Advanced Exception Types

All advanced exception types are preceded with "COM.Allaire.ColdFusion."

- CFEXECUTE.OutputError
- CFEXECUTE.Timeout
- FileException
- HTTPAccepted
- HTTPAuthFailure
- HTTPBadGateway
- HTTPBadRequest
- HTTPCFHTTPRequestEntityTooLarge
- HTTPCGIValueNotPassed
- HTTPConflict
- HTTPContentLengthRequired
- HTTPContinue
- HTTPCookieValueNotPassed
- HTTPCreated
- HTTPFailure
- HTTPFileInvalidPath
- HTTPFileNotFound
- HTTPFileNotPassed
- HTTPFileNotRenderable
- HTTPForbidden
- HTTPGatewayTimeout
- HTTPGone
- HTTPMethodNotAllowed
- HTTPMovedPermanently
- HTTPMovedTemporarily
- HTTPMultipleChoices
- HTTPNoContent
- HTTPNonAuthoritativeInfo
- HTTPNotAcceptable
- HTTPNotFound
- HTTPNotImplemented
- HTTPNotModified
- HTTPPartialContent
- HTTPPaymentRequired
- HTTPPreconditionFailed
- HTTPProxyAuthenticationRequired
- HTTPRequestURITooLarge
- HTTPResetContent
- HTTPSeeOther
- HTTPServerError
- HTTPServiceUnavailable
- HTTPSwitchingProtocols
- HTTPUnsupportedMediaType
- HTTPUrlValueNotPassed
- HTTPUseProxy
- HTTPVersionNotSupported
- POPAuthFailure
- POPConnectionFailure
- POPDeleteError
- Request.Timeout
- SERVLETJRunError
- HTTPConnectionTimeout