

[<< Back to Dashboard](#)

Request Context Guide

Covers up to version 3.5.0

Overview

On every request to a ColdBox event, the framework creates an object that models the incoming request. This object is called the **Request Context** object

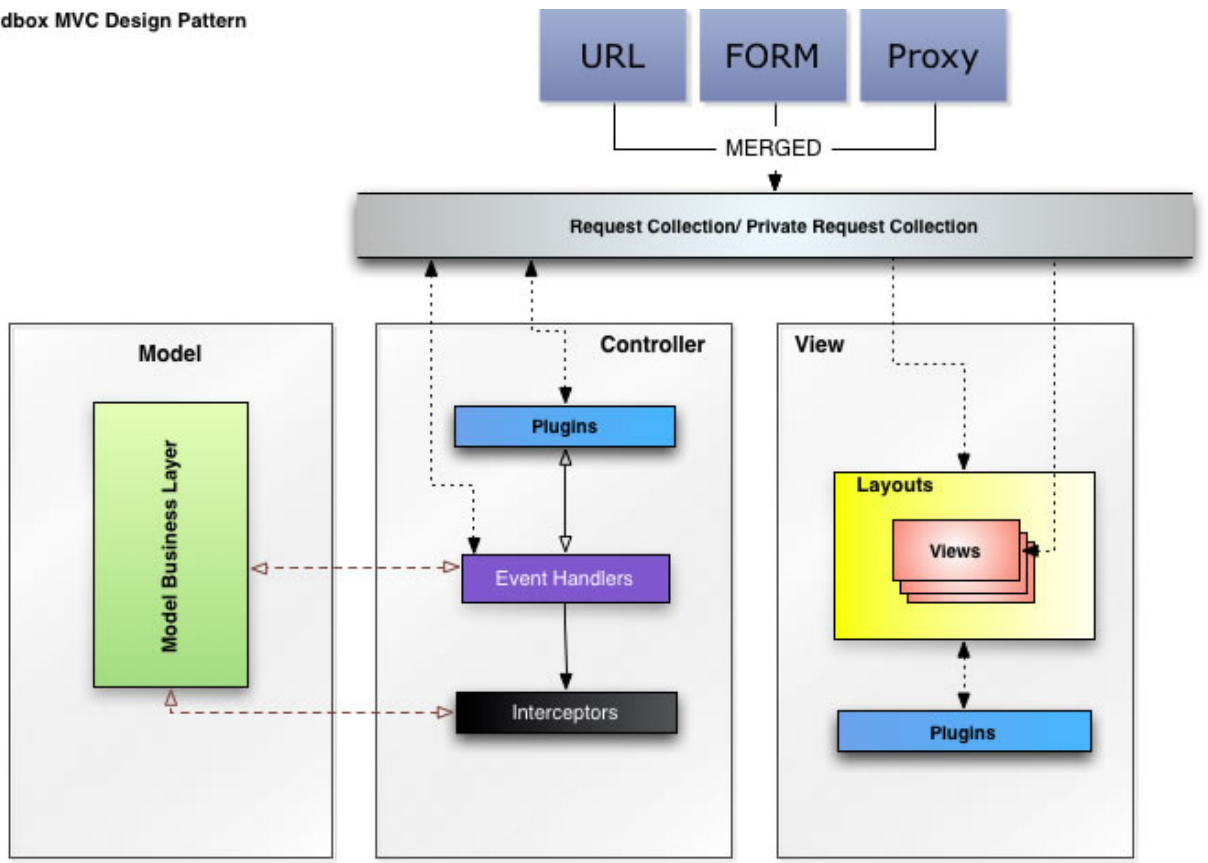
(*coldbox.system.web.context.RequestContext*)

and it contains the incoming FORM/URL/REMOTE variables the client sent in and the object lives in the ColdFusion **request** scope. You will use this object in the controller and view layer of your application to get/set values, get metadata about the request, marshall data for RESTful request, and so much more.

Contents

- [Request Context Guide](#)
 - [Overview](#)
 - [How does it work?](#)
 - [Event Handlers](#)
 - [Views](#)
 - [Interceptors](#)
 - [Plugins](#)
 - [What can I do with it?](#)
 - [Most Commonly Used Methods](#)
 - [Request Metadata Methods](#)
 - [Extending The Request Context](#)
 - [Summary](#)

The Coldbox MVC Design Pattern



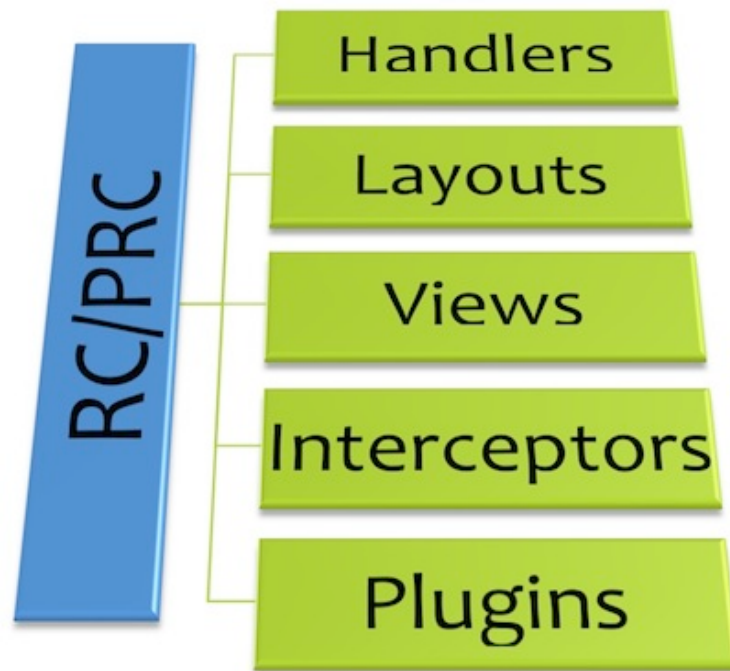
How does it work?

The framework will merge the incoming URL/FORM/REMOTE variables into a single

structure called the **request collection** structure that will live inside the request context object. We also internally create a second collection called the **private request collection** that is useful to store data and objects that have no outside effect.

As best practice, store data in the private collection and leave the request collection intact with the client's request data.

The request collection lives in the request scope and can be accessed from every single part of the framework's life cycle, except the Model layer of course. Therefore, there is one contract and way on how to handle FORM/URL/REMOTE and any other kind of variables. You can consider the request collection to be sort of a super information highway that is unique per request. You will interact with it to get/set values that any part of a request's lifecycle will interact with it. The ColdBox debugger even traces for you how this data structure changes as events are fired during a request.



The request context objects makes it much easier for you to have access to variables, since you will no longer be scoping them. It also **encapsulates** any other scopes you would like to merge into the collection. Another important aspect of the request context is that it also contains several metadata and utility methods that you can use to build your applications. Some of these methods are covered below, but the best place to learn about all of its methods is to see the [API Docs](#).

Note: FORM variables take precedence

Event Handlers

All event handlers receive 3 arguments whenever you declare actions in them:

- **event** - The request context object
- **rc** - A reference to the request collection structure
- **prc** - A reference to the private request collection structure

```
function list(event,rc,prc){}
```

Receiving the references makes a huge performance difference as you can interact with the structures instead of methods in the **event** object. So as a tip, try to interact with the structures as much as you can.

Views

All layouts and views have the following variables declared for you in the **variables** scope:

- **event** - The request context object
- **rc** - A reference to the request collection structure
- **prc** - A reference to the private request collection structure

So there is no need to know where stuff comes from, it is there already, just use it. Again, try to interact with the structures as much as you can so you can increase performance.

Interceptors

All interceptor events receive the following arguments:

- **event** - The request context object
- **interceptData** - The interception data

They do not receive the two references, so if you need them in your events, you will need to reference them manually:

```
function preProcess(event, interceptData){
  var rc = event.getCollection();
  var prc = event.getCollection( private=true );
}
```

Plugins

Plugins do not receive references to the collections or context object, they must request them via the following methods:

- **getRequestContext()** - Get access to the context reference
- **getRequestCollection(boolean private=false)** - Get access to the private or public request collection

What can I do with it?

The event object has a plethora of methods to help you deal with a request:

- Getting a reference to the collection
- Appending structures to the collection
- Removing values from the collection
- Paraming values in the collection (Similar to cfparam)
- Getting values from the collection
- Setting values into the collection
- Getting metadata about a request, such as the current executing event, handler, action, layouts, view, and much more.

- Determining a coldbox proxy request
- Determining if you are in SES mode
- Building links for you
- So much more.

Most Commonly Used Methods

Below you can see a listing of the mostly used methods in the request context object:

- **buildLink(string linkto, [boolean translate], [boolean ssl], [string baseURL])** : Build a link in SES or non SES mode.
- **clearCollection()** : Clears the entire collection
- **collectionAppend(struct, overwrite)** : Append a collection overwriting or not
- **getCollection()** : Get a reference to the collection
- **getEventName()** : The event name in use in the application (e.g. do, event, fa)
- **getSelf()** : Returns index.cfm?event=
- **getValue(keyname, defaultValue)** : get a value
- **getTrimValue(keyname, defaultValue)** : get a value trimmed
- **getSESBaseURL()** : Get the base ses url string.
- **isProxyRequest()** : flag if the request is an incoming proxy request
- **isSES()** : flag if ses is turned on
- **isAjax()** : Is this request ajax based or not
- **noRender(boolean)** : flag that tells the framework to not render any html, just process and silently stop.
- **overrideEvent()** : Override the event in the collection
- **paramValue()** : param a value in the collection
- **removeValue()** : remove a value
- **setValue(keyname, value)** : set a value
- **setLayout(layout)** : Set the layout to use for this request
- **setView(name, nolayout, cache, cachetimeout, cachelastaccesstimeout)** : Used to set a view to render
- **showDebugPanel(boolean)** : Sets whether the ColdBox debugging panel will be rendered or not.
- **valueExists()** : Checks if a value exists in the collection.
- **renderData()** : Marshall data to JSON, JSONP, XML, WDDX, PDF, HTML, etc.

Some Samples:

```
//Get a reference
<cfset var rc = event.getCollection() >

//test if this is an MVC request or a remote request
<cfif event.isProxyRequest() >
  <cfset event.setValue( 'message' , 'We are in proxy mode right now' )>
</cfif>

//param a variable called page
<cfset event.paramValue( 'page' ,1)>
//then just use it
<cfset event.setValue( 'link' , 'index.cfm?page=#rc.page#' )>

//get a value with a default value
<cfset event.setValue( 'link' , 'index.cfm?page=#event.getValue( ' page ',1)#' )>

//Set the view to render
<cfset event.setView( 'homepage' )>

//Set the view to render with no layout
<cfset event.setView( 'homepage' , true )>
```

```

//set the view to render with caching stuff
<cfset event.setView(name= 'homepage' ,cache= 'true' ,cacheTimeout= '30' )>

//override a layout
<cfset event.setLayout( 'Layout.Ajax' )>

//check if a value does not exists
<cfif not event.valueExists( 'username' )>

</cfif>

//Don't show the debug panel for this request
<cfset event.showDebugPanel( false )>

//Tell the framework to stop processing gracefully, no renderings
<cfset event.noRender() >

//Build a link
<form action= "#event.buildLink('user.save')#" method= "post" >
</form>

```

Request Metadata Methods

- **getCurrentAction()** : Get the current execution action (method)
- **getCurrentEvent()** : Get's the current incoming event, full syntax.
- **getCurrentHandler()** : Get the handler or handler/package path.
- **getCurrentLayout()** : Get the current set layout for the view to render.
- **getCurrentView()** : Get the current set view
- **getCurrentModule()** : The name of the current executing module
- **getDebugpanelFlag()** : Get's the boolean flag if the ColdBox debugger panel will be rendered.
- **getDefaultLayout()** : Get the name of the default layout.
- **getDefaultView()** : Get the name of the default view.

Extending The Request Context

One key feature about ColdBox is its flexibility. Every application is different and unique. Therefore, the request context might not do exactly what you need and if you needed to modify it, well you would have to modify the framework core files and that is **NOT** a good idea. Therefore, the decorator pattern came to our rescue and you can decorate the request context by using a **request context decorator** object. With this object you can decorate some or all of the methods in the request context provided to you by the framework. You can even add more functionality to it. How, well, for that you need to read the [Request Context Decorator Guide](#).

Summary

You will be surprised at how many useful methods the event object contains. So I highly suggest you read the [CFC API](#) and study it. You will find tons of methods that will make your development life easier. Don't only read this guide and play with it, read also the CFC docs, they provide a wealth of information also.