

[<< Back to Dashboard](#)

Model-View-Controller Contents

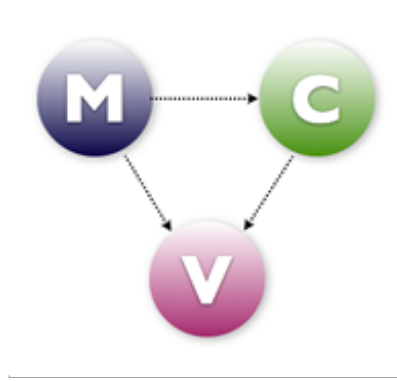
Introduction

ColdBox is based on the Model-View-Controller development pattern for web applications. **MVC** is a software approach that separates application logic from presentation. Read below as wikipedia states it:

- [Model-View-Controller](#)
 - [Introduction](#)
 - [MVC Layers](#)
 - [Summary](#)
 - [Resources](#)

"A developer often wishes to separate *data (model)* and *user interface (view)* concerns, so that changes to the user interface will not affect data handling, and that the data can be reorganized without changing the user interface. The model-view-controller solves this problem by decoupling data access and business logic from data presentation and user interaction, by introducing an intermediate component: *the controller*." [Wikipedia](#)

In its most basic form, a ColdBox application can be layered/divided into three layers: Model, View, Controller. The most important lesson in creating sustainable and object oriented applications using the MVC design pattern, is understanding that there are separations of concerns; separation of code, business logic, data and presentation data (or more). This separation is key to a succesful transition into an object oriented framework and applying object oriented design & analysis. This separation brings in turn a decoupling or interdependency between layers/objects that help in creating more maintainable software. Your layers/objects become in fancy terms **cohesive** or tight, they do one thing and one thing right. I hope you are not shivering with fear by this point. Relax, object oriented analysis and design takes years to master and the best recipe is to learn and practice. So I will repeat this again, separation is key. You don't have to understand all the concepts like design patterns, MVC, domain model or all the fancy words all at once, I encourage you to keep reading, practicing and enjoying how to build better software. That is why you are here, to take it a step further.



MVC Layers

- **The Model** represents your data structures and business logic. The *domain-specific* representation of the information that the application operates on. Many applications also use a persistent storage mechanism (such as a database) to store and retrieve data. MVC does not specifically mention the data access layer because it is understood to be underneath or encapsulated by the Model. This is the most important part of your application and it is usually modeled by coldfusion components. You can even do the entire model in another language or physical location (web services). All you need to understand is that its the layer that runs the logic show!
- **The View** is the information that is being presented to a user, in our case it can be HTML/XML/Raw data. It can be used to build various views, composite views, layouts-view combinations, etc. ColdBox even has a layout manager to help you create easy to use applicaton/website layouts and render single or composite views. This layer uses the model data to present itself or change itself.
- **The Controller** serves as an intermediary between the Model, the View, and any other resources needed to process a request. In ColdBox, it processes and responds to events, typically user actions, and may invoke changes on the model or any other external feature of the web application. In ColdBox, a Controller is called an **Event Handler**

Summary


An important but unknown fact about these definitions, is the fact that data gets passed around from each of the three layers. How? Frameworks or patters never explain this, it is just assumed. Well, this data object in ColdBox terms is called the **Request Collection**. It is a data structure that lives in the request scope, so it is unique per request, and it models a user's request. It is available to all parts of the running framework application and it is contained inside of an object called the **request context**. This object has tons of utilities and methods that you can use to get/set values that are shared amongst all layers. You will learn more about the layers in all the provided guides in this wiki.

If you don't need or understand the added separation of business logic into a

model layer or find that maintaining models requires more complexity than you want or have experienced before, you can ignore them and build your application minimally using Event Handlers and Views. However, you should know that this is **highly discouraged** and if you will be mastering object oriented web applications, you should develop and separate logic into a domain model. The choice is yours.

Resources

- http://en.wikipedia.org/wiki/Domain_model
- [Domain Drive Design](#)
- [Martin Fowler's Domain Model](#)

 Categories:

- [theory](#)