Magento®

eCommerce Platform for Growth

# Certified Developer Plus Exam

## *Study Guide*

# Table of Contents



 

# Introduction

This Study Guide is intended to help you prepare for the Magento® Certified Developer Plus Exam. For each content area of the exam, the Study Guide details the objectives (skills) that are tested by the exam. For each objective, it provides questions that you should ask yourself as you study the Magento code base, along with code references that you can use as entry points to find answers to those questions.

This Study Guide is continually being revised and improved. When preparing for the exam, remember to check the website for the latest version of this Study Guide.

# 1- Basics

This topic comprises approximately 5% of the Standard form of the exam. Questions are drawn randomly from the following topics and objectives:

## High-level Magento architecture

- Describe Magento codepools
- Describe typical Magento module structure
- Describe Magento templates and layout files location
- Describe Magento skin and JavaScript files location
- Identify and explain the main Magento design areas (adminhtml and frontend)
- Explain class naming conventions and their relationship with the autoloader
- Describe methods for resolving module conflicts.

  To verify your understanding of these objectives, ask yourself the following questions:

  - How does the framework interact with the various codepools?
  - What constitutes a namespace and a module?
  - What does the structure of a complete theme look like?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Core_Model_App`
  - `Mage_Core_Model_Config`
  - `Varien_Autoload`

## Magento configuration

- Explain how Magento loads and manipulates configuration information
- Describe class group configuration and use in factory methods
- Describe the process and configuration of class overrides in Magento
- Register an Observer

- Identify the function and proper use of automatically available events, including `*_load_after`, etc.
- Set up a `cron` job

  Since Magento is an XML DOM configuration-based framework, a thorough understanding of how Magento loads and assembles its configuration XML DOM is a core competency.

  - How does the framework discover active modules and their configuration?
  - What are the common methods with which the framework accesses its configuration values and areas?
  - How are per-store configuration values established in the XML DOM?
  - By what process do the factory methods and autoloader enable class instantiation?
  - Which class types have configured prefixes, and how does this relate to class overrides?
  - Which class types and files have explicit paths?
  - What configuration parameters are available for event observers?
  - What are the interface and configuration options for automatically fired events?
  - What is the structure of event observers, and how are properties accessed therein?
  - What configuration parameters are available for `cron` jobs?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Core_Model_App_Area`
  - `Mage_Core_Model_Config`
  - `Mage_Core_Model_Store`
  - `Varien_Event_Observer`

## Internationalization

- Describe how to plan for internationalization of a Magento site
- Describe the use of Magento translate classes and translate files
- Describe the advantages and disadvantages of using subdomains and subdirectories in internationalization

To verify your understanding of these objectives, ask yourself the following questions:

- Which method is used for translating strings, and on which types of objects is it generally available?
- In what way does the developer mode influence how Magento handles translations?
- How many options exist to add a custom translation for any given string?
- What is the priority of translation options?
- How are translation conflicts (when two modules translate the same string) processed by Magento?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Model_Translate::init()`
- `Mage_Core_Model_Locale::emulate()`

# 2- Request Flow

This topic comprises approximately 6% of the exam. Questions are drawn randomly from the following topics and objectives:

## Application initialization

- Describe the steps for application initialization
- Describe the role of the system entrypoint, `index.php`

  Starting with the `index.php`, including `Mage.php`, follow through the steps Magento takes to set up the run time environment.

  - How and when is the include path set up and the auto loader registered?
  - How and when does Magento load the base configuration, the module configuration, and the database configuration?
  - How and when are the two main types of setup script executed?
  - When does Magento decide which store view to use, and when is the current locale set?
  - Which ways exist in Magento to specify the current store view?
  - When are the request and response objects initialized?

  These code references can be used as an entry point to find answers to the questions above:

  - `index.php`
  - `Mage_Core_Model_App::run()`
  - `Mage_Core_Model_Config::loadBase()` and `init()`

## Front Controller

- Describe the role of the front controller
- Identify uses for events fired in the front controller

  Follow the flow of control through front controller initialization until an action controller is dispatched.

  - Which ways exist in Magento to add router classes?
  - What are the differences between the various ways to add routers?
  - Think of possible uses for each of the events fired in the front controller

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Controller_Varien_Front::init()` and `dispatch()`

## URL rewrites

- Describe URL structure/processing in Magento
- Describe the URL rewrite process

Focus on the internals of database-based URL rewrites:

- What is the purpose of each of the fields in the `core_url_rewrite` table?
- When does Magento created the rewrite records for categories and products?
- How and where does Magento find a matching record for the current request?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Controller_Varien_Front::dispatch()`
- `Mage_Core_Model_Url_Rewrite::rewrite()`

## Request routing

- Describe request routing/request flow in Magento
- Describe how Magento determines which controller to use and how to customize route-to-controller resolution

Starting with the front controller delegating the process of mapping a request to a controller action, study the steps that occur until a controller action is dispatched.

- Which routers exist in a native Magento implementation?
- How does the standard router map a request to a controller class?
- How does the standard router build the filesystem path to a file that might contain a matching action controller?
- How does Magento process requests that cannot be mapped?
- After a matching action controller is found, what steps occur before the action method is executed?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Controller_Varien_Front::init()`
- `Mage_Core_Controller_Varien_Router_Standard::collectRoutes()` and `match()`

## Module initialization

- Describe the steps needed to create and register a new module
- Describe the effect of module dependencies
- Describe different types of configuration files and the priorities of their loading

This objective covers how Magento loads modules and how modules interact with each other:

- What does "Magento loads modules" mean?
- In which order are Magento modules loaded?
- Which core class loads modules?
- What are the consequences of one module depending on another module?
- During the initialization of Magento, when are modules loaded in?
- Why is the load order important?
- What is the difference regarding module loading between `Mage::run()` and `Mage::app()`?

These code references can be used as an entry point to find answers to the questions above:

- `Mage::run()` and `Mage::app()`
- `Mage_Core_Model_App::run()` and `init()`

## Design and layout initialization

- Identify the steps in the request flow in which:
  - Design data is populated
  - Layout configuration files are parsed
  - Layout is compiled
  - Output is rendered

The design configuration is part of Magento's view implementation. This objective covers the processing of these XML instructions:

- Which ways exist to specify the layout update handles that will be processed during a request?
- Which classes are responsible for the layout being loaded?
- How are layout xml directives processed?
- Which configuration adds a file containing layout xml updates to a module?
- Why is the load order of layout xml files important?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Controller_Varien_Action::loadLayout()`
- `Mage_Core_Model_Layout::__construct()`
- `Mage_Core_Model_Layout_Update::load()`

## Flushing data (output)

- Describe how and when Magento renders content to the browser
- Describe how and when Magento flushes output variables using the front controller

This objective covers the response object as well as combining JavaScript and CSS files.

- Which events are associated with sending output?
- Which class is responsible for sending output?
- What are possible issues when this output is not sent to the browser using the typical output mechanism, but is instead sent to the browser directly?
- How are redirects handled?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Controller_Varien_Front::dispatch()`
- `Mage_Core_Controller_Response_Http and super classes`
- `Mage_Page_Block_Html_Head::getCssJsHtml()`

# 3- Rendering

This topic comprises approximately 6% of the exam. Questions are drawn randomly from the following topics and objectives:

## Themes in Magento

- Define and describe the use of themes in Magento:
    - How you can use themes to customize core functionality?
    - How can you implement different designs for different stores using Magento themes?
    - In which two ways you can register custom theme?

  These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Core_Model_Layout`
    - `Mage_Core_Model_Layout_Update`
    - `Mage_Core_Model_Design`
    - `Mage_Core_Model_Design_Package`

- Define and describe the use of design packages:
    - What is the difference between package and theme?
    - What happens if the requested file is missed in your theme/package?

  These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Core_Model_Layout`
    - `Mage_Core_Model_Layout_Update`
    - `Mage_Core_Model_Design`
    - `Mage_Core_Model_Design_Package`
      `Mage_Core_Block_Template`

- Describe the process of defining template file paths:
    - Which kind of paths (absolute or relative) does Magento use for templates and layout files?
    - How exactly can Magento define which physical file correspond to certain template/layout to use?
    - Which classes and methods need to be rewritten in order to add additional directories to the fallback list?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Model_Design`
- `Mage_Core_Model_Design_Package`
- `Mage_Core_Block_Template`

## Blocks

- Describe the programmatic structure of blocks:
  - What are blocks used for in Magento?
  - What is the parent block for all Magento blocks?
  - Which class does each block that uses a template extend?
  - In which way does a template block store information about its template file? Does it store an absolute or a relative path to the template?
  - What is the role of the `Mage_Core_Block_Abstract` class?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Core_Model_Layout`
  - `Mage_Core_Model_Layout_Update`
  - `Mage_Core_Block_Template`
  - `Mage_Core_Block_Abstract`
  - `Mage_Adminhtml_Block_Abstract`
- Describe the relationship between templates and blocks:
  - Can any block in Magento use a template file?
  - How does the `$this` variable work inside the template file?
  - Is it possible to render a template without a block in Magento?
  - Is it possible to have a block without a template in Magento?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Core_Block_Template`
  - `Mage_Core_Block_Abstract`
- Describe the stages in the lifecycle of a block:
  - Which class is responsible for creating an instance of the block?
  - Which class is responsible for figuring out which blocks should be created for certain pages?

- How is the tree of blocks typically rendered?
- Is it possible to create an instance of the block and render it on the page without using the Magento layout?
- Is it possible to create an instance of the block and add it to the current layout manually?
- How are a block's children rendered? Once you added a child to the block, can you expect it will be rendered automatically?
- What is a difference in rendering process for different types of blocks?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Model_Layout`
- `Mage_Core_Model_Layout_Update`
- `Mage_Core_Block_Template`
- `Mage_Core_Block_Abstract`
- `Mage_Core_Block_Text`
- `Mage_Core_Block_Text_List`

- Describe events fired in blocks:
  - How can block output be caught using an observer?
  - What events do `Mage_Core_Block_Abstract` and `Mage_Core_Block_Template` fire?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Block_Abstract`

- Identify different types of blocks:
  - What is the purpose of each of the following block types:
    - Mage_Core_Block_Template
    - Mage_Core_Block_Text_List
    - Mage_Core_Block_Text
  - Which block type renders its children automatically?
  - Which block type is usually used for a "content" block on Magento pages?

These code references can be used as an entry point to find answers to the questions above:

- `Mage/Core/Block/*`
- `Mage_Core_Block_Abstract`

- `Mage_Core_Block_Template`
- `Mage_Core_Block_Text`
- `Mage_Core_Block_Text_List`
- `Mage_Page_Block_Html_Head`

- Describe block instantiation:
  - How can a template's block instance be accessed inside the template file, and how can other block instances be accessed?
  - How can block instances be accessed from the controller?
  - How can block instances be accessed inside install scripts or other model class instances?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Core_Model_Layout`
  - `Mage_Core_Model_Layout_Update`
  - `Mage_Core_Controller_Varien_Action`

- Explain different mechanisms for disabling block output:
  - In which ways can block output be disabled in Magento?
  - Which method can be overridden to control block output?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Core_Model_Layout`

- Describe how a typical block is rendered:
  - Which class performs rendering of the template?
  - Which classes are responsible for figuring out the absolute path for including the template file?
  - In which method are templates rendered?
  - How can output buffering be enabled/disabled when templates are rendered?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Core_Model_Layout`
  - `Mage_Core_Model_Layout_Update`
  - `Mage_Core_Block_Template`
  - `Mage_Core_Block_Abstract`

## Design layout, XML schema, and CMS content directives

- Describe the elements of Magento's layout XML schema, including the major layout directives:
  - How are `<update />`, `<block />`, and `<action />` used in Magento layout?
  - Which classes and methods determine which nodes from layout XML correspond to certain URLs?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Core_Model_Layout`
  - `Mage_Core_Model_Layout_Update`
  - `Mage_Core_Block_Abstract`

- Register layout XML files:
  - How can layout XML files be registered for the frontend and adminhtml areas?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Core_Model_Layout_Update`

- Create and add code to pages:
  - How can code be modified or added to Magento pages using the following methods?
    - Template customizations
    - Layout customizations
    - Overriding block classes
    - Registering observers on general block events
  - In which circumstances are each of the above methods more or less appropriate than others?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Core_Controller_Varien_Action`
  - `Mage/Core/Block/*`
  - `Mage_Core_Block_Abstract`
  - `Mage_Core_Block_Template`

- `Mage_Core_Block_Text`
- `Mage_Core_Block_Text_List`
- `Mage_Page_Block_Html_Head`

- Explain how variables can be passed to block instances via layout XML:
  - How can variables be passed to the block using the following methods?
    - From layout xml file
    - From controller
    - From one block to another
    - From an arbitrary location (for example, install/upgrade scripts, models)
  - In which circumstances are each of the above methods more or less appropriate than others?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Model_Layout`
- `Mage_Core_Controller_Varien_Action`
- `Mage/Core/Block/*`
- `Mage_Core_Block_Abstract`
- `Mage_Core_Block_Template`
- `Mage_Core_Block_Text`
- `Mage_Core_Block_Text_List`
- `Mage_Page_Block_Html_Head`

- Describe various ways to add and customize JavaScript to specific request scopes:
  - Which block is responsible for rendering JavaScript in Magento?
  - Which modes of including JavaScript does Magento support?
  - Which classes and files should be checked if a link to a custom JavaScript file isn't being rendered on a page?

These code references can be used as an entry point to find answers to the questions above:

- `Mage/Core/Block/*`
- `Mage_Core_Block_Abstract`
- `Mage_Core_Block_Template`
- `Mage_Core_Block_Text`
- `Mage_Core_Block_Text_List`
- `Mage_Page_Block_Html_Head`

## General References for this section:

`Mage_Core_Model_Layout`

`Mage_Core_Model_Layout_Update`

`Mage_Core_Model_Design`

`Mage_Core_Model_Design_Package`

`Mage_Core_Controller_Varien_Action`

# 4- Working with Database in Magento

This topic comprises approximately 11% of the exam. Questions are drawn randomly from the following topics and objectives:

## Models, resource models, and collections

- Describe the basic concepts of models, resource models, and collections, and the relationship they have to one another
- Configure a database connection
- Describe how Magento works with database tables
- Describe the load-and-save process for a regular entity
- Describe group save operations
- Describe the role of `Zend_Db_Select` in Magento
- Describe the collection interface (filtering/sorting/grouping)
- Describe the hierarchy of database-related classes in Magento
- Describe the role and hierarchy of setup objects in Magento

These objectives fall into two broad areas. The first is about how models work with resource models and collections in order to access the database storage layer; the second is about how to work with the database directly by using the adapter classes and the `Zend_Db_Select` object to create queries.

- Which methods exist to access the table of a resource model?
- Which methods exist to create joins between tables on collections and on select instances?
- How does Magento support different RDBMSs?
- How do table name lookups work, and what is the purpose of making table names configurable?
- Which events are fired automatically during CRUD operations?
- How does Magento figure out if a `save()` call needs to create an INSERT or an UPDATE query?
- How many ways exist to specify filters on a flat table collection?
- Which methods exist to influence the ordering of the result set for flat table collections? How do the methods differ?

- Why and how does Magento differentiate between setup, read, and write database resources?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Model_Abstract`
- `Mage_Core_Model_Resource_Db_Abstract`
- `Mage_Core_Model_Resource_Db_Collection_Abstract`
- `Mage_Core_Model_Resource::getTableName()`
- `Zend_Db_Select`

## Install/upgrade scripts

- Describe the install/upgrade workflow
- Write install and upgrade scripts using set-up resources
- Identify how to use the DDL class in setup scripts

Each module can encapsulate the preparation and upgrade of the database table it requires via setup scripts.

- Under which circumstances are setup scripts executed?
- What is the difference between the different classes used to execute setup scripts?
- Which is the base setup class for flat table entities, and which one the base for EAV entities?
- Which methods are generally available in setup scripts to manipulate database tables and indexes?
- What is the difference between `addAttribute()` and `updateAttribute()` in EAV setup scripts?
- How can you implement a rollback in Magento?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Core_Model_App::run()` and `_initModules()`
- `Mage_Core_Model_Resource_Setup::applyAllUpdates()` and `applyAllDataUpdates()`
- `Mage_Eav_Model_Entity_Setup::addAttribute()` and `updateAttribute()`

# 5- Entity-Attribute-Value (EAV) Model

This topic comprises approximately 8% of the exam. Questions are drawn randomly from the following topics and objectives:

## EAV model concepts

- Define basic EAV concepts and class hierarchy
- Describe the database schema for EAV entities
- Describe the EAV entity structure and its difference from the standard core resource model
- Describe the EAV load-and-save process and its differences from the regular load-and-save process

  This objective covers understanding how EAV entity values are stored in the database, how the involved tables relate, how the EAV resource models differ from the flat table resource models and how the EAV models process CRUD operations.

  - Which classes in `Mage_Eav` are used as resource models and which are used as regular models?
  - How do flat and EAV resource models differ?
  - Which entities in a native Magento installation use EAV resource models and why?
  - What are the advantages and disadvantages of EAV over flat table resource models?
  - How are store and website scope attribute values implemented in the Magneto EAV system?
  - How does the model distinguish between insert and update operations?
  - How do load and save processes for EAV entities differ from those for flat table entities? What parts are identical?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Eav_Model_Config`
  - `Mage_Eav_Model_Entity_Abstract`
  - `Mage_Eav_Model_Entity_Collection_Abstract`
  - `Mage_Eav_Model_Entity_Abstract::load()` and `save()`
  - `Mage_Core_Model_Abstract::load()` and `save()`
  - `Mage_Eav_Model_Entity_Collection_Abstract::load()`

- `Mage_Core_Model_Resource_Db_Collection_Abstract::load()`

## Attributes management

- Identify the purpose of attribute frontend, source, and backend models
- Describe how to implement the interface of attribute frontend, source, and backend models:
  - How do attribute models, attribute source models, attribute backend models and attribute frontend models relate to each other?
  - Which methods have to be implemented in a custom source model (or frontend model or backend model)?
  - Can adminhtml system configuration source models also be used for EAV attributes?
  - What is the default frontend model (and source and backend model) for EAV attributes?
  - Does every attribute use a source model?
  - Which classes and methods are related to updating the EAV attribute values in the flat catalog tables? What factors allow for attributes to be added to flat catalog tables?
  - How are store-scoped entity attribute values stored when catalog flat storage is enabled for that entity type?
  - Which frontend, source, and backend models are available in a stock Magento installation?
  - How do multi-lingual options for attributes work in Magento?
  - How do you get a list of all options for an attribute for a specified store view in addition to the admin scope?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Eav_Model_Entity_Attribute_Abstract`
  - `Mage_Eav_Model_Entity_Attribute_Backend_Abstrace`
  - `Mage_Eav_Model_Entity_Attribute_Frontend_Abstract`
  - `Mage_Eav_Model_Entity_Attribute_Source_Abstract`
  - `Mage_Eav_Model_Entity_Attribute_Source_Table`
  - `Mage_Eav_Model_Resource_Entity_Attribute_Option_Collection::load()`
- Describe how to create and customize attributes.

  Besides simply using the stock EAV attributes that come with Magento, one of the most common operations for developers is to modify or create attributes.

- Which setup methods are available to work with EAV entities?
- How can an EAV setup class be instantiated in a setup script if not specified in the XML `<class>` configuration for a setup resource?
- What is the difference between `addAttribute()` and `updateAttribute()`?
- What are the advantages of using a custom setup class for manipulating EAV attributes in a custom module?

This code reference can be used as an entry point to find answers to the questions above:

- `Mage_Eav_Model_Entity_Setup`

# 6- Adminhtml

This topic comprises approximately 6% of the exam. Items are drawn randomly from the following topics and objectives:

## Common structure/architecture

- Describe the similarities and differences between adminhtml and frontend interface and routing:
    - Which areas in configuration are only loaded for the admin area?
    - What is the difference between admin and frontend controllers?
    - When does Magento figure out which area to use on the current page?
    - How you can make your controller work under the `/admin` route?

    These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Adminhtml_Controller_Action`
    - `Mage_Core_Model_Layout`
    - `Mage_Core_Model_Layout_Update`
    - `Mage_Core_Controller_Varien_Router_Standard`
- Describe the components and types of cache clearing using the adminhtml interface:
    - At which moment does Magento check if a user is logged in or not?
    - Which class do most Magento adminhtml blocks extend?
    - What are the roles of adminhtml config?
    - What are the differences between the different cache types on the admin cache cleaning page?
    - What is the difference between “Flush storage” and “Flush Magento Cache”?
    - How you can clear the cache without using the UI?

    These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Adminhtml_Model_Config`
    - `Mage_Adminhtml_Model_Config_Data`
    - `Mage_Admin_Model_Observer`
    - `Mage_Core_Model_Cache`

## Forms in Magento

- Define form structure, form templates, grids in Magento, and grid containers and elements:
    - Which block does a standard Magento form extend?
    - What is the default template for a Magento form?
    - Describe the role of a form container and its template.
    - Describe the concept of Form elements, and list system elements implemented in Magento.
    - Describe the concept of Fieldsets.
    - How can you render an element with a custom template?

  These code references can be used as an entry point to find answers to the questions above:

    - `lib/Varien/Data/Form/*`
    - `Mage_Adminhtml_Block_Widget_Form`
    - `Mage_Adminhtml_Block_Widget_Form_Container`

## Grids in Magento

- Create a simple form and grid for a custom entity
- Describe how to implement advanced Adminhtml Grids and Forms, including editable cells, mass actions, totals, reports, custom filters and renderers, multiple grids on one page, combining grids with forms, and adding custom JavaScript to an admin form:
    - Which block class do Magento grid classes typically extend?
    - What is the default template for Magento grid instances?
    - How can grid filters be customized?
    - How does Magento actually perform sorting/paging/filtering operations?
    - What protected methods are specific to adminhtml grids, and how are they used?
    - What is the standard column class in a grid, and what is its role?
    - What are column renderers used for in Magento?
    - How can JavaScript used for a Magento grid be customized?
    - What is the role of the grid container class and its template?
    - What is the programmatic structure of mass actions?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Adminhtml_Block_Widget_Grid`
- `Mage_Adminhtml_Block_Widget_Grid_Column`
- `Mage_Adminhtml_Block_Widget_Grid_Column_Renderer/*`
- `Mage_Adminhtml_Block_Widget_Grid_Column_Filter/*`
- `Mage_Adminhtml_Block_Widget_Grid_Container`
- `app/design/adminhtml/default/default/template/widget/grid.phtml`
- `app/design/adminhtml/default/default/template/widget/grid/container.phtml`

## System configuration

- Define the basic terms, elements, and structure of system configuration XML:
  - How can elements in system configuration be rendered with a custom template?
  - How does the structure of `system.xml` relate to the rendered elements in the System Configuration view?
  - How can the CSS class of system configuration elements be changed?
  - What is the syntax for specifying the options in dropdowns and multiselects?
  - Which classes are used to parse and render system configuration XML?
  - What is the syntax to specify a custom renderer for a field in system configuration?
  - How does Magento store data for system configuration?
  - What is the difference between `Mage::getStoreConfig(...)` and `Mage::getConfig()->getNode(...)`?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage/Adminhtml/Model/System/Config/*`
  - `Mage/Adminhtml/Block/System/Config/*`

- Describe system configuration scopes:
  - How do different scopes (global, website, store) work in Magento system configuration?
  - How does Magento store information about option values and their scopes?

These code references can be used as an entry point to find answers to the questions above:

- `core_config_data` table
- `Mage_Core_Model_Core_Config_Data`
- `Mage_Core_Model_Resource_Config_Data`
- `Mage_Core_Model_Resource_Config_Data_Collection`

## Access Control Lists (ACL) and permissions in Magento

- Define/identify basic terms and elements of ACL
- Use ACL to:
  - Set up a menu item
  - Create appropriate permissions for users
  - Check for permissions in permissions management tree structures

To verify your understanding, ask yourself these questions:

- For what purpose is the `_isAllowed()` method used and which class types implement it?
- What is the XML syntax for adding new menu element?
- What is `adminhtml.xml` used for? Which class parses it, and which class applies it?
- Where is the code located that processes the ACL XML and where is the code that applies it?
- What is the relationship between Magento and `Zend_Acl`?
- How is ACL information stored in the database?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Admin_Model_Acl`
- `Mage_Admin_Model_Acl_Resource`
- `Mage_Admin_Model_Acl_Role`
- `Mage_Admin_Model_Resource_Acl`
- `Mage_Admin_Model_Resource_Role`
- `Mage_Admin_Model_Resource_Roles`
- `Mage_Admin_Model_Resource_Rules`

## Working with extensions in Magento

- Describe how to enable and configure extensions
- Define Magento extensions and describe the different types of extension available (Community, Core, Commercial)

  Questions to ask yourself:

  - In which folders are Magento extensions files located?
  - Which files are necessary to make custom modules work?
  - How can module dependencies be manipulated?
  - What is the role of the downloader?
  - How can modules be installed through Magento Connect?

# 7- Catalog

This topic comprises approximately 8% of the exam. Items are drawn randomly from the following topics and objectives:

## Product Types

- Identify and describe standard product types (simple, configurable, bundled, etc.)
- Create custom product types from scratch or modify existing product types
- Identify how custom product types interact with indexing, SQL, and underlying data structures.

  In addition to allowing customization of existing product types, the framework provided by the Magento catalog module lets you create completely new ones.

  - Which product types exist in Magento?
  - Which product types are implemented as part of the `Mage_Catalog` module, and which are not?
  - What steps need to be taken in order to implement a custom product type?
  - How do the different product types handle calculation?
  - Which indexing processes does the product type influence?
  - Which product types implement a parent-child relationship between product entities?
  - Which database tables are shared between product types, and which ones are specific to one product type?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage_Catalog_Model_Product_Type`
  - `Mage_Catalog_Model_Product_Type_Abstract`
  - `Mage_Catalog_Model_Product_Type_Simple`
  - `Mage_Catalog_Model_Resource_Product_Type_Configurable`
  - `Mage_Bundle_Model_Product_Type`

## Price Generation

- Identify basic concepts of price generation in Magento
- Modify and adjust price generation for products (for example, during integration of third-party software):
    - Under what circumstances are product prices read from the index tables?
    - From which modules do classes participate in price calculation?
    - Which ways exist to specify custom prices during runtime?
    - How do custom product options influence price calculation?
    - How are product tier prices implemented and displayed?
    - What is the priority of the different prices that can be specified for products (price, special price, group price, tier price, etc.)?

  These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Catalog_Model_Product::getPrice()` and `getFinalPrice()`
    - `Mage_Catalog_Model_Product_Type_Price ::getTierPrice()`
    - `Mage_Catalog_Model_Product_Indexer_Price`
    - `Mage_Catalog_Model_Product_Type_Configurable_Price`

## Category Structure

- Describe the Category Hierarchy Tree Structure implementation (the internal structure inside the database), including:
    - The meaning of `parent_id 0`
    - The construction of paths
    - The attributes required to display a new category in the store

  Questions to ask yourself:

    - How is the category hierarchy reflected in the database? Does it differ when multiple root categories are present?

- How is a catalog tree read from the database tables, with and without flat catalog tables enabled?
- How does working with categories differ if the flat catalog is enabled on a model level?
- How is the category parent id path set on new categories?
- Which methods exist to read category children and how do they differ?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Catalog_Model_Category`
- `Mage_Catalog_Model_Resource_Category`
- `Mage_Catalog_Model_Resource_Category_Collection`
- `Mage_Catalog_Model_Resource_Category_Tree`

## Catalog Price Rules

- Identify how catalog price rules are implemented in Magento:
  - How are the catalog price rules related to the product prices?
  - How are the catalog price rules stored in the database?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_CatalogRule_Model_Rule`
- `Mage_CatalogRule_Model_Observer`
- `Mage_CatalogRule_Model_Rule_Product_Price`

## Other Skills

- Choose optimal catalog structure (EAV vs. Flat) for a given implementation
- Implement, troubleshoot, and modify Magento tax rules
- Modify, extend, and troubleshoot the Magento layered ("filter") navigation
- Troubleshoot and customize Magento indexes
- Describe custom product options in Magento

The Magento catalog module doesn't only consist of categories and products: a lot of additional catalog functionality is implemented, partly within the `Mage_Catalog` module, partly in other modules.

- When and how are the catalog flat tables updated when a product is modified?
- Which factors are used by the `Mage_Tax` module to apply the correct tax rate (or rates) to a product price?
- How can attributes with custom source models be integrated into layered navigation filtering?
- Which classes are responsible for rendering the layered navigation?
- Which indexes are used for the layered navigation?
- Which steps are needed to integrate a custom indexer into the framework offered by the `Mage_Index` module?
- How are custom product options stored on quote and order items?
- How can you specify custom product options on-the-fly on quote items?
- How are custom product options copied from quote to order items?
- How are custom product options processed when a product is added to the cart?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Catalog_Model_Product_Indexer_Flat`
- `Mage_Catalog_Model_Category_Indexer_Flat`
- `Mage_Catalog_Model_Product_Indexer_Eav`
- `Mage_Catalog_Model_Resource_Product_Indexer_Eav`
- `Mage_Tax_Helper_Data:: getPrice()`
- `Mage_Catalog_Block_Layer_State`
- `Mage_Catalog_Block_Layer_View`
- `Mage_Catalog_Model_Layer`

# 8- Checkout

This topic comprises approximately 13% of the exam. Items are drawn randomly from the following topics and objectives:

## Checkout components

- Describe how to modify and effectively customize the quote object, the quote item object, and the address object:
    - What is the quote model used for in Magento?
    - What is the shopping cart model used for in Magento?
    - How does Magento store information about the shopping cart?
    - How are different product types processed when added to the cart?
    - What is the difference between shipping and billing address objects in Magento? How is each used in the quote object?
    - What is the difference in processing quote items for onepage and multishipping checkout in Magento?
    - How does Magento process additional information about products being added to the shopping cart (custom options, components of configurable products, etc.)?
    - How do products in the shopping cart affect the checkout process?
    - How can the billing and shipping addresses affect the checkout process?
    - When exactly does inventory decrementing occur?
    - When exactly does card authorization and capturing occur?

    These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Sales_Model_Quote`
    - `Mage_Sales_Model_Quote_Address`
    - `Mage_Sales_Model_Quote_Item`
    - `Mage_Sales_Model_Quote_Address_Item`
    - `Mage/CatalogInventory/etc/config.xml`
    - `Mage_CatalogInventory_Model_Observer`
- Explain the database schema for total models:
    - What are total models responsible for in Magento?
    - How you can customize total models?
    - How can the individual total models be identified for a given checkout process?
    - How can the priority of total model execution be customized?

- To which objects do total models have access in Magento, and which objects do they usually manipulate?
- Which class runs total models processing?
- What is the flow of total model execution?
- At which moment(s) are total models executed

These code references can be used as an entry point to find answers to the questions above:

- `Mage/Sales/etc/config.xml`
- `Mage/Tax/etc/config.xml`
- `Mage_Sales_Model_Quote_Address`
- `Mage_Sales_Model_Quote_Address_Item`
- `Mage_Sales_Model_Quote_Address_Total_Abstract`
- `Mage_Sales_Model_Quote_Address_Total_Collector`
- `Mage/Sales/Model/Quote/Address/Total/*`
- `Mage/SalesRule/etc/config.xml`
- `Mage_SalesRule_Model_Validator`

## Shopping Cart price rules

- Describe how shopping cart price rules work and how they can be customized:
  - Which module is responsible for shopping cart price rules?
  - What is the difference between shopping cart and catalog price rules?
  - What are the limitations of Magento shopping cart rules?

  These code references can be used as an entry point to find answers to the questions above:

  - `Mage/SalesRule/etc/config.xml`
  - `Mage/SalesRule/Model/*`

## Shipping and payment methods in Magento

- Describe the programmatic structure of shipping methods, how to customize existing methods, and how to implement new methods
- Describe the shipping rates calculation process:
  - How does Magento calculate shipping rates?
  - What is the hierarchy of shipping information in Magento?
  - How does the `TableRate` shipping method work?
  - How do US shipping methods (FedEX, UPS, USPS) work?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Shipping_Model_Carrier_Abstract`
- `Mage_Shipping_Model_Rate_Abstract`
- `Mage_Shipping_Model_Rate_Request`
- `Mage_Shipping_Model_Rate_Result`
- `Mage/Shipping/Model/Rate/Result/*`
- `Mage_Shipping_Model_Info`
- `Mage/Shipping/Model/Carrier/*`
- `Mage/Shipping/Model/Resource/Carrier/*`

- Describe the programmatic structure of payment methods and how to implement new methods:
  - How can payment method behavior be customized (for example: whether to charge or authorize a credit card; controlling URL redirects; etc.)?
  - Which class is the basic class in the payment method hierarchy?
  - How can the stored data of payment methods be customized (credit card numbers, for example)?
  - What is the difference between payment method and payment classes (such as `order_payment`, `quote_payment`, etc.)?
  - What is the typical structure of the payment method module?
  - How do payment modules support billing agreements?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Payment_Model_Method_Abstract`
- `Mage_Payment_Model_Method_Cc`
- `Mage_Payment_Model_Info`
- `Mage/Paypal/*`
- `Mage/PaypalUk/*`

## Magento multishipping implementation

- Describe how to extend the Magento multishipping implementation
- Identify limitations of the multishipping implementation:
  - How does the storage of quotes for multishipping and onepage checkouts differ?
  - Which quotes in a multishipping checkout flow will be virtual?
  - What is the difference in the multishipping processing for a quote with virtual products in it?

- How can different product types be split among multiple addresses when using multishipping in Magento?
- How many times are total models executed on a multishipping checkout in Magento?
- Which model is responsible for multishipping checkout in Magento?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Checkout_Model_Type_Multishipping`
- `Mage_Sales_Model_Quote_Address`
- `Mage_Sales_Model_Quote`

# 9- Sales and Customers

This topic comprises approximately 9% of the exam. Items are drawn randomly from the following topics and objectives:

## Sales

- Describe order creation in the admin
- Describe the differences in order creation between the frontend and the admin:
    - Which classes are involved in order creation in the admin? What are their roles (especially the role of adminhtml classes)?
    - How does Magento calculate price when an order is created from the admin?
    - Which steps are necessary in order to create an order from the admin?
    - What happens when existing orders are edited in the admin?
    - What is the difference between order status and order state?

  These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Sales_Model_Order`
    - `Mage_Sales_Model_Order_Address`
    - `Mage_Adminhtml_controllers_Order_CreateController`
    - `Mage_Adminhtml_Model_Sales_Order_Create`
- Card operations (capturing and authorization):
    - Which classes and methods are responsible for credit card operations (for example authorization or capturing)?
    - What is the difference between “pay” and “capture” operations?
    - What are the roles of the `order`, `order_payment`, `invoice`, and `payment` methods in the process of charging a card?
    - What are the roles of the total models in the context of the invoice object?
    - How does Magento store information about invoices?

  These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Sales_Model_Order_Invoice`
    - `Mage/Sales/Model/Order/Invoice/*`

        - `Mage_Sales_Model_Order_Payment`
        - `Mage_Payment_Model_Method_Info`
        - `Mage_Payment_Model_Method_Abstract`

- Describe the order shipment structure and process:

    - How shipment templates be customized?
    - How can different items from a single order be shipped to multiple addresses? Is it possible at all?
    - How does Magento store shipping and tracking information?

    These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Sales_Model_Order_Shipment`
    - `Mage/Sales/Model/Order/Shipment/*`

- Describe the architecture and processing of refunds:

    - Which classes are involved, and which tables are used to store refund information?
    - How does Magento process taxes when refunding an order?
    - How does Magento process shipping fees when refunding an order?
    - What is the difference between online and offline refunding?
    - What is the role of the credit memo total models in Magento?

    These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Sales_Model_Order_Creditmemo`
    - `Mage/Sales/Model/Order/Creditmemo/*`

- Describe the implementation of the three partial order operations (partial invoice, partial shipping, and partial refund):

    - How do partial order operations affect order state?
    - How is data for partial operations stored?

    These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Sales_Model_Order_Invoice`
    - `Mage_Sales_Model_Order_Payment`
    - `Mage_Sales_Model_Order_Shipment`
    - `Mage_Sales_Model_Order_Creditmemo`
    - `Mage_Payment_Model_Method_Abstract`

- Describe cancel operations:
    - What cancel operations are available for the various order entities in Magento (order, order item, shipment, invoice, credit memo)? Do all of them support cancellation?
    - How are taxes processed during cancel operations?

  These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Adminhtml_Sales_OrderController`
    - `Mage_Sales_Model_Order_Invoice`
    - `Mage_Sales_Model_Order_Payment`
    - `Mage_Sales_Model_Order_Shipment`
    - `Mage_Sales_Model_Order_Creditmemo`

## Customer

- Describe the architecture of the customer module
- Describe the role of customer addresses
- Describe how to add, modify, and display customer attributes:
    - What is the structure of tables in which customer information is stored?
    - What is the customer resource model?
    - How is customer information validated?
    - How can customer-related email templates be manipulated?
    - What is the difference between shipping and billing addresses for a customer?
    - How does the number of shipping and billing address entities affect the frontend interface for customers?
    - How does customer information affect prices and taxes?
    - How can attributes be added to a customer address? How are custom address attributes you converted to an order address?
    - Can a customer be added to two customer groups at the same time?

  These code references can be used as an entry point to find answers to the questions above:

    - `Mage/Customer/etc/config.xml`
    - `Mage_Customer_Model_Customer`
    - `Mage_Customer_Model_Resource_Customer`
    - `Mage_Customer_Model_Customer_Address`

# 10- Advanced features

This topic comprises approximately 11% of the exam. Items are drawn randomly from the following topics and objectives:

## Widgets

- Create frontend widgets and describe widget architecture:
    - What classes are typically involved in Magento widget architecture?
    - How are admin-configurable parameters and their options specified?

    These code references can be used as an entry point to find answers to the questions above:

    - `Mage_Widget_Model_Widget`

## API

- Use the Magento API to implement third party integrations
- Extend the existing Magento API to allow for deeper integrations into third party products
- Describe the different Web Service APIs available within the Magento Core
- Describe the advantages and disadvantages of the available Web Service APIs in Magento
- Identify the configuration files used for the v2 SOAP API
- Describe the purpose of the configuration files related to the API

    Ask yourself these questions:

    - What are the syntactical differences between API versions?
    - How is the final WSDL composed? How can it be customized?
    - How are existing methods overwritten? How can additional methods be added?

These code references can be used as an entry point to find answers to the questions above:

- `Mage_Api_Model_Wsdl_Config`
- `Mage_Api_Model_Server`

## Other Skills

- Integrate Google features (Google Wallet, Checkout, AdWords, Analytics) into Magento implementation

# 11- Enterprise Edition

This topic comprises approximately 9% of the exam. Questions are drawn randomly from the following objectives:

- Describe how to customize, extend, and troubleshoot Enterprise Edition catalog target rules:
    - What additional possibilities does the Enterprise target rule module provide over the `Mage_CatalogRule`?
    - How does the module store the rules in the database?
    - Which indexer processes the rules, and how is the index used to realize faster reads?

    These code references can be used as an entry point to find answers to the questions above:

    - `Enterprise_TargetRule_Model_Rule`
    - `Enterprise_TargetRule_Model_Resource_Index`
    - `Enterprise_TargetRule_Model_Observer`
- Describe how to customize, extend, and troubleshoot the Enterprise Edition reward point system:
    - How do the features offered by the reward point system hook into other Magento modules?
    - Under which conditions may reward points be assigned?
    - Which steps are required to add new custom options to assign reward points?

    These code references can be used as an entry point to find answers to the questions above:

    - `Enterprise_Reward_Model_Observer`
    - `Enterprise_Reward_Model_Reward_Rate::calculateToCurrency()` and `calculateToPoints()`
    - `Enterprise_Reward_Helper_Data::getRateFromRatesArray()`
- Describe how to implement, customize, and troubleshoot Enterprise Edition website restrictions:
    - How does the website restrictions module determine which pages are accessible without authentication?

- How does the module prevent visitors from accessing restricted website content?

These code references can be used as an entry point to find answers to the questions above:

- `Enterprise_WebsiteRestriction_Model_Observer`
- `Enterprise_WebsiteRestriction_Block_Cms_Stub`

- Identify the elements and functioning of Enterprise Edition Full Page Cache:
  - How does the full page cache module hook into the request flow and serve page requests from the cache?
  - What is the difference between the `applyWithoutApp()` and `applyInApp()` methods of the abstract?
  - What is the role of each node in the `cache.xml` configuration file?
  - Which steps are necessary to implement a dynamically rendered block inside an otherwise fully cached page?

These code references can be used as an entry point to find answers to the questions above:

- `Enterprise_PageCache_Model_Observer`
- `Enterprise_PageCache_Model_Container_Abstract`
- `Enterprise_PageCache_Model_Processor_Default`

- Describe the Payment Bridge:
  - How is the Payment Bridge implemented and configured?

This code reference can be used as an entry point to find answers to the question above:

- `Enterprise/Pbridge/*`

# 12- Challenge Questions

Approximately 8% of the Magento Certified Developer Plus exam consists of Challenge questions, randomly drawn from the objectives covered in the individual sections of this Study Guide. The Challenge questions go into more detail or cover more complex areas of the topic than the questions in the other sections of the Magento Certified Developer Plus exam.

# Sample Questions

These questions are representative of the types of question on the exam. Answers are given on the page following the last question.

1- In which of the following methods would you log event names?

A. `Mage::addObserver()`

B. `Varien_Event::dispatchEvent()`

C. `Mage::logEvent()`

D `Mage_Core_Model_App::dispatchEvent()`

2- In which order are translations loaded?

| |
|---|
| 1- Module translation files under `app/locale/[locale]/*.xml` |
| 2- Translations stored in the DB table `core_translate` |
| 3- Theme translations in the `translate.csv` file located in the theme `locale/[locale]/` directory |

A. 1, 2, 3

B. 1 ,3, 2

C. 2, 1, 3

D. 2, 3, 1

3 - When specifying a custom connection resource for a module that handles the DB reads, which of the following is a possible resource name?

A. `<modulename_readonly>`

B. `<modulename_read>`

C. `<modulename_default>`

D. `<modulename_setup>`

E. `<modulename_connection_read>`

4 - Assuming a setup class of `Mage_Sales_Model_Resource_Setup`, which of the following will correctly add an attribute to the sales/order entity?

A.
```
$installer->addAttribute('sales_order_entity', 'foo', array
('type'=> 'int'));
```

B.
```
$installer->addAttribute('sales/order', 'foo', array('type'
=>'int'));
```

C.
```
$installer->addAttribute('order', 'foo', array('type' => 'int'));
```

D.
```
$installer->addAttribute('sales_flat_order', 'foo', array('type'
=>'int'));
```

# Answers to Sample Questions

1-D

2-B

3-B

4-C