**Chapter X Xblocks-the component architecture of Open edX**

**X.1 What is a XBlock**

The XBlock specification is a component architecture designed to make it easier to create new online educational experiences..An XBlock is the basic building block of any edX course. It controls its own data structure (**model**), its own HTML rendering (**view**), and its own business logic (**controller**). An XBlock executes in a **runtime**, which provides common services/utilities, user/request context, and storage. While the course structure data is primarily stored in a database infrastucture called **modulestoreB**, an Xblock's **persistence** is configured on a field-by-field basis by designating the scope or each of its **fields**. Some scopes support user-specific data, which results in different XBlock content for different target users.

An XBlock has a unique identifier known as **block-id**. Its instantiation within a context of a course is identified by its **usage-id**. A common class of xBlocks have the same defined **type** (or **category**) that indicate its common behavior and interface. Some examples of XBlock types are: "course\_info" (shared by Handouts and Announcements), "problem" (shared by all CAPA-based assessments), "video", "chapter" (a.k.a Section), and "sequential" (a.k.a. Subsection).

**X.2 Xock concepts**

XBlocks are build such that course teams use them to create independent course components that work seamlessly with other components in an online course. These include the following

*Xblock Fields* ***:-*** These are used  to store state data for your Xblock.

*Xblock Methods* ***:-*** These are used in the XBlock Python file to define the behavior of your XBlock.

*Xblock Fragments* ***:-*** A fragment is a part of a web page returned by an XBlock view method. A fragment typically contains all the resources needed to display the XBlock in a web page, including HTML content, JavaScript, and CSS resources.

*Xblock children****:-*** An XBlock can have child XBlocks.

*Xblock runtime* ***:-*** An XBlock runtime is the application that hosts XBlock.

**X.3 XBLOCK & EDX PLATFORM**

**X.3.1 EdX Studio**

EdX Studio is the application in the edX Platform that instructors use to build courseware. Because instructors use Studio to add and configure XBlocks, Studio is also an Xblock runtime application.

**X.3.2 EdX LMS**

The edX Learning Management System (LMS) is the application in the edX Platform that learners use to view and interact with courseware. Because it presents XBlocks to learners and records their interactions, the LMS is also an Xblock runtime application.

## **[X.4](http://edx.readthedocs.io/projects/xblock-tutorial/en/latest/concepts/children.html" \l "id10) XBlock Tree Structure**

An XBlock does not refer directly to its children. Instead, the structure of a tree of XBlocks is maintained by the runtime application, and is made available to the XBlock through a runtime service.

This allows the runtime to store, access, and modify the structure of a course without incurring the overhead of the XBlock code itself.

XBlock children are not implicitly available to their parents. The runtime provides the parent XBlock with a list of child XBlock IDs. The child XBlock can then be loaded with the **get\_child()**function. Therefore the runtime can defer loading child XBlocks until they are actually required.

**X.5 DATA ACCESS & PERSISTENCE**

**Data Access** refers to software and activities related to storing, retrieving, or acting on data housed in a database or other repository. Two fundamental types of data access exist:

1. sequential access
2. random access

**Persistence** refers to object and process characteristics that continue to exist even after the process that created it ceases or the machine it is running on is powered off.

**X.5.1 Data access and persistence in edx platform**

This part describes how the edx-platform does CRUD (Create, Read, Update, Delete) operations on xblocks and other models and backs those operations with persistence. Currently open-edx uses Django's ORM backed by a SQL DB as the persistence layer for user-relative data. The user history data is moved to a non-SQL db(Mongo db).

**X.5.2 XMLModuleStore**

* The original course data is in xml which is stored on a file system.
* When the server launches, it scans the file system and loads every such course into memory.
* It then serves the courseware from memory.
* The ***XMLModuleStore*** is the data access layer which finds, loads, and serves up the course data to the application. It has only Read access. No Create, Update, nor Delete.

**X.5.3 MongoModuleStore**

* When Studio is launched, a read-write storage mechanism and data access layer is needed so, ***MongoModuleStore***with CRUD methods is added.
* Here the persistence technology (Mongo) with the DAO CRUD functionality has been convoluted
* ***MongoModuleStore*** stores each ***Xmodule*** as a separate document in a non-SQL (Mongo) database.
* The db connection and pymongo usages are needed to be abstracted out. Hence it uses the xblock's old style location (tag, org, courseid, type, blockid, draft or None) as the key.
* All changes (add, remove, move child) immediately impact the published course.
* ***MongoModuleStore*** handles inheritance by loading the whole course on each access.

**X.5.4 SplitMongoModuleStore**

* It is created to fix the expense and fragility of inheritance.
* It offers full versioning of all edits, the ability to share the same content and settings between courses, as many named branches of a course or named-subcourse structure as one needs.
* Once again the persistence technology (Mongo) with the DAO functionality (CRUD operations on xblocks) is being convoluted.