Docentron Application Engine Architecture

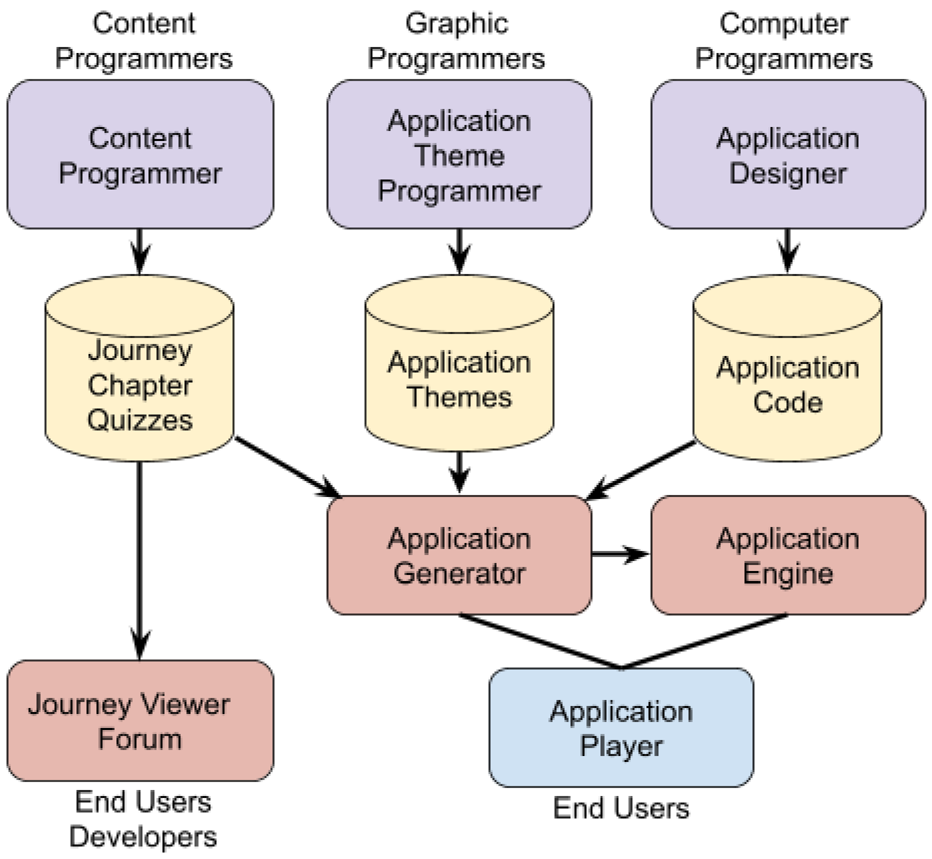
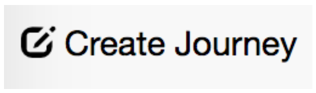
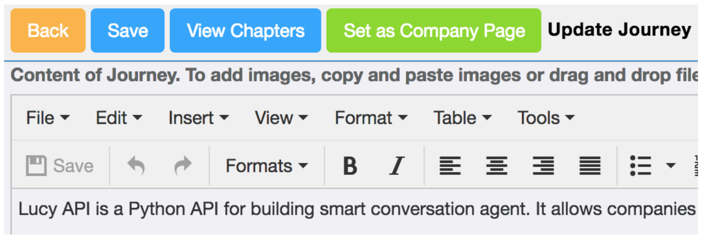


Figure 1. Docentron Application Engine Architecture

Docentron application platform provides online tools for creating mobile and online applications, user interface, and rich contents for the applications. The platform allows developers to quickly build new applications by allowing developers to share codes, designs, assets, and contents. Figure 1 shows how the tools (content programmer, graphic programmer, and application designer) generate rich content and application codes that can be customised and packaged by AI, developers, and even end-users. Application player allows end-users to play their choice of contents on their choice of games and applications. Journey viewer allows developers to publish applications and forums for sharing and to get user feedback.

Online Content Editor and Content Programmer with DC cloud file storage allows developers to quickly publish contents, themes, and applications. It allows forums for end-users feedback and purchase paid applications and services.

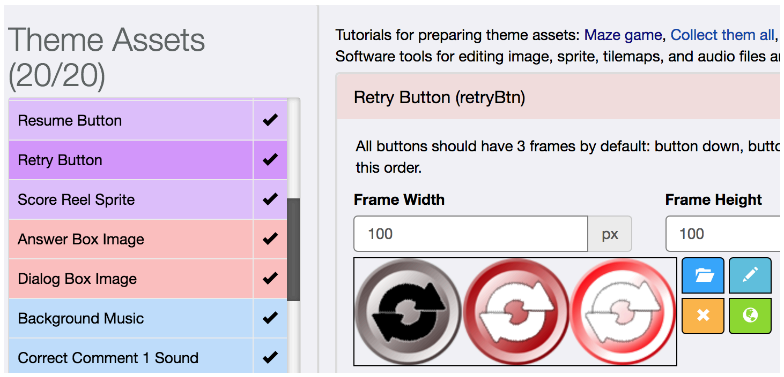




Content of Journey. To add images, copy and paste images or drag and drop file

Lucy API is a Python API for building smart conversation agent. It allows companies

Application Theme Programmer allows graphic designers to customise user interfaces of existing applications to provide customised user experiences for different content types.

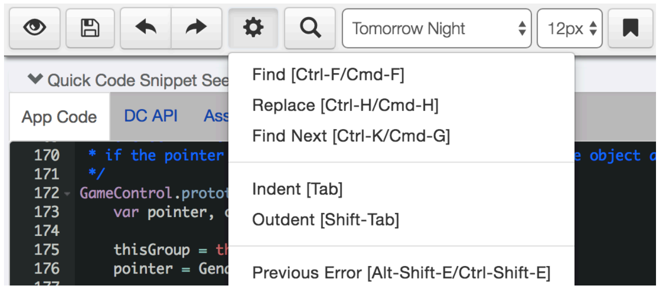


Tutorials for preparing theme assets: Maze game, Collect them all,
Software tools for editing image, sprite, tilemaps, and audio files a

All buttons should have 3 frames by default: button down, butto
this order.

Frame Width: 100 px
Frame Height: 100

Application Designer allows programmers to create new interactions, game play mechanics and application logics.



```
 * if the pointer                                 e object c
 */
GameControl.protot
    var pointer, c

    thisGroup = th
    pointer = Gene
```

Docentron applications are built on Docentron API. The overall **structure of DC API** is shown below:

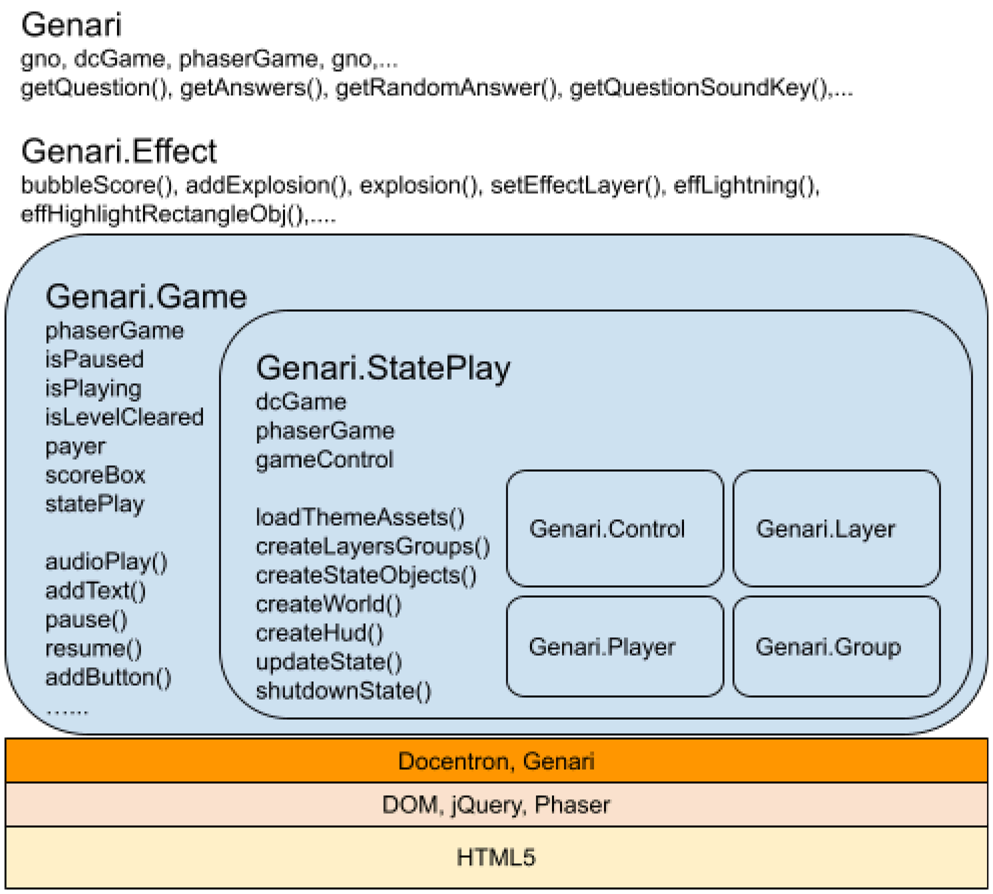


Figure 2. DC Application API structure

Docentron application engine is built on HTML5, which is a cross platform web and mobile application environment. Therefore, DC applications run on PC, MAC, Android and iPhone devices. It utilizes common web application libraries such as jQuery, and Phaser for game animation features. Figure 2 shows DC application API structure. Genari is the package name of DC game application API that contains various supporting functions and classes. Genari.Game represents a DC game application which creates and controls StatePlay, which in turn runs Control, Player, Layer, and Group objects. See DC API code in the game editor to explore functions and classes provided.

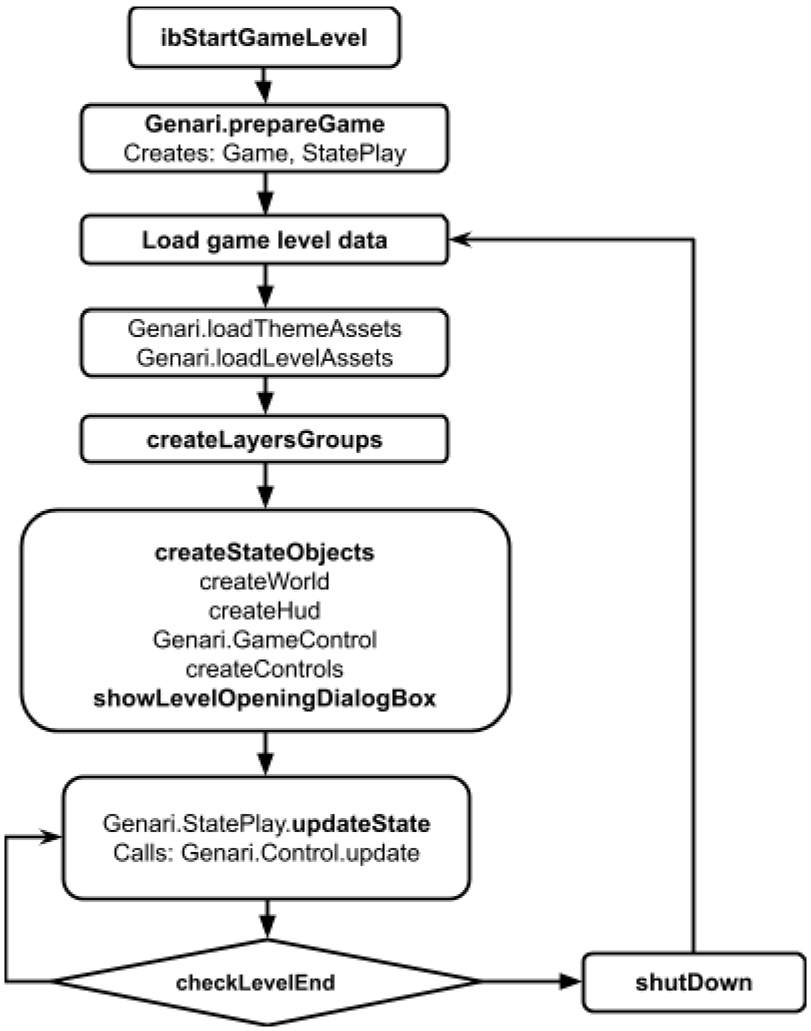


Figure 3. Flow of application states and function calls. The application starts by ibStartGameLevel calls from the application engine.

Figure 3 shows the flow of the API function calls. When the application starts, DC game engine calls ibStartGameLevel() that starts the initialization of the game application: Genari.prepareGame().

Genari.prepareGame() creates a Genari.Game object and shows the application banner. When the user starts the application by clicking on the banner, Generi.StatePlay starts to load a game level data and with theme data for the level. It then preload theme assets and quiz media files (game level assets) from the server. It then initializes Genari.Layer, Genari.Group, Genari.Control, Genari.Player objects for the game level. It then calls serieses of functions for the programmer to populate the game world using createStateObjects. When the initialization is completed, it finally starts to render frames calling updateState() just before each frame for the programmer to update the states of the game world.

1. **Genari** package provides common functions and interfaces to HTML5 and Phaser.

2. **Genari.Effect** class provides common effect animations such as score bubbling and explosion animation.
3. **Genari.Game**: allows the programmer to control the game state, score, game level changes.
4. **Genari.Layer**: each layer represents a rendering layer. The layers manage the rendering orders so some display objects will be rendered on top of another.
5. **Genari.Group**: each group represents a group of display objects. For example, pickup items may share common behaviors, we can manage all pick up items under pickupItemGroup.
6. **Genari.Player** represents a player display object that a player can control
7. **Genari.Control** class manages user input such as keyboard and pointer interactions.
