GCF 3 Beta Release notes

The documentation for GCF 3 is not exhaustive as of now, and it is a work in progress. Stay tuned to the GCF blog for more articles on how to use GCF 3, and for interesting tips & tricks to tweak its functionalities. Also, feel free to write (to gcf-3@vcreatelogic.com) and ask us if you need any assistance in using the library.

The intention of this document is to capture the things to keep in mind while using the GCF 3 Beta. Many of the things listed here comes under the category of “work under progress”, and would most probably change by the final release.

❑ General notes

- ❑ The method that needs to be adopted to create and load Components in GCF 3 is a developer’s choice. GCF 3 doesn’t give an automatic mechanism that parses a list of components and loads them.
- ❑ There are 3 types of configurations possible for a GCF 3 application. They are

        - **Core** - A Qt Core based application configuration, which doesn't involve any GUI elements. Components designed to be loaded in this configuration should be sub-classed from **GCF::Component.** Applications should create an instance of **GCF::Application**.
        - **Gui** - A Qt Widget based application configuration. Components designed to be loaded in this configuration should be sub-classed from **GCF::GuiComponent.** This application configuration can also load core components. Applications should create an instance of **GCF::GuiApplication**.
        - **QML** - A Qt Quick based application configuration. Components designed to be loaded in this configuration should be sub-classed from **GCF::QmlComponent.** This application configuration can also load core components. Applications should create an instance of **GCF::QmlApplication**.

- **Investigator**
    - Investigator is a simple tool designed to make cross platform UI testing of Qt applications a simple task.
    - Investigator is a work under progress.
- **IPC**
    - The IPC module in GCF 3 provides an easy mechanism through which you can enable inter-object communication across Qt applications in a local network.
    - Couple of rules that are imposed are
        - Applications that want to allow incoming IPC communication must create instance of **GCF::IpcServer** and listen on it.
        - The objects who wants to participate in IPC has to be QObjects.
        - The corresponding object should be exposed by the owner component and "allowmetaaccess" should be enabled on the object.
        - All public invokable methods can be given access.
    - Refer to the IPC documentation for a complete reference on how it can be used
- **GDrive**
    - GDrive module in GCF 3 provides a library to deal with the contents from a GDrive account. The library is in its beta state and is a work in progress.
    - The current version supports the following
        - attach and authenticate a GDrive account .
        - querying the contents of an attached GDrive account.
        - offline modification of the contents of an attached GDrive account (create/delete/modify/move/star document, etc).

- **Fiber**
    - **Fiber** module in GCF 3 provides an architecture to offer the functionality in a GCF 3 component as a web-service, accessible using a simple JSON packet format
    - **Fiber** should automatically figure out the correct LocalServer address of the main

Fiber process in most platforms. However in the rare event of this not working for you, please update the ***fiberServerSocket*** variable in the ***postReceiveRequest()*** method in the file **FiberAccessHooks.php** with the address that would be printed by the **Fiber** process at the console.

- There are not much configuration options available for **Fiber** right now. This is intentional. These configuration options would be included based on the feedback from the developers on using **Fiber (**post beta release, and before final release**).**
- **Fiber** is capable of resuming from a crash, automatically. However, right now its not capable of detecting an infinite loop / race condition (In fact, please consider both these features as a work in progress; i.e it would be better if the component doesn't crash after all !).
- **GUI components should not be creating "Widgets" at the server side.**
    - Would result in a blocking debug dialog (on Windows at least) in the event of a crash. And Fiber wouldn't be able to restart the surrogate without manual intervention.
    - There seems to be workarounds for this however (**NOTE:** Following links are untested)
        - http://stackoverflow.com/questions/396369/how-do-i-disable-the-debug-close-application-dialog-on-windows-vista
        - http://pcsupport.about.com/od/windows7/ht/disable-error-reporting-windows-7.htm
- Number of concurrent connections supported is a function of the configurations available at the web server. The most common configuration seems to be max 200. If **Fiber** should handle more concurrent requests this needs to be configured at the web server (Apache + PHP settings).
- Maximum size of **HTTP_POST_REQUEST** is a configuration option of **PHP.** If **Fiber** is expected to transact huge sizes of data, it is a good idea to check this.
- **Fiber** dumps PHP logs at the platform specific **<TEMP>** directory.
    - **PHPFiberErrorLog.txt** - should give all the error messages.
    - **PHPFiberLog.txt** - the verbose log.