Application prestarting for application developers

Alexey Shilov (alexey.shilov@nokia.com), Jussi Lind (jussi.lind@nokia.com)

February 16, 2010

1 Introduction to prestarting

If the start-up time of a DUI application is way too long it can be prestarted at the boot time. This means that it can be started and initialized on the background without showing the UI, so from user point of view it's not running.

Launching of prestarted application is very fast from the user's perspective, it takes less than 1 second, because it's only a matter of showing the UI. Applications in prestarted state consume memory so not all applications can be prestarted, but just selected, time-critical ones like the Camera UI.

An application can be prestarted only if it explicitly supports prestarted mode. Prestarting and normal launching of an application can be distinguished from the command line arguments: `-prestart` argument is provided when only prestarting is wanted. This should start the application without UI being shown.

Technically, an application in prestarted state runs in the Qt main loop, but its window is hidden by DUI framework (`DuiApplicationWindow::show()` does nothing in the prestarted mode). Application should avoid any use of CPU resources when it's in the prestarted state. When user initiates (via D-Bus) launch of the prestated application, the framework notifies the application that it must become visible and by default the application window will be automatically shown.

2 Overview of the prestart API

The prestart API offers two ways to get the notifications:

- Qt signals:
  - `DuiApplication::prestartReleased()`
  - `DuiApplication::prestartRestored()`
- Virtual handlers that user can override in her application class derived from DuiApplication:
  - `void DuiApplication::releasePrestart()`
  - `void DuiApplication::restorePrestart()`

There are two ways how prestarted applications can handle close event which can be caused by user pressing the close (or back) button or someone calling the close method of the D-Bus interface:

- Exit normally
- Return to the prestarted state ("lazy shutdown")

In the "lazy shutdown" mode the application never really exits, it just hides the UI, stops and resets all activites and returns to the prestarted state. In the device, there's a dedicated Application Life-cycle Daemon, applifed, that takes care of prestarting and re-prestarting the applications. So, if a application exits or crashes it will be re-prestarted by applifed. Prestarted applications communicate with applifed via D-Bus service so that daemon knows what applications are currently in prestarted state. This happens behind the scenes.

Note: An application will not be re-prestarted if it wasn't relased at all. This effectively prevents a loop where applifed would re-prestart an application that crashes before it can be even shown.

Currently, only single window applications should be prestarted, since multi-window applications are not officially supported. They can still be dealt with the handler API.

User may mix the signal and handler -based API's freely. The signals will always be emitted. The signal -based API might be easier and faster to use, but handlers give more control and forces the use of an object oriented model.

3 How to enable prestarting

- Call `DuiApplication::setPrestartMode(Dui::PrestartMode mode)` with `Dui::LazyShutdown` or `Dui::TerminateOnClose` parameter in application's `main()` function to notify framework how application is going to handle close event.

  Note: if application is registered to applifed daemon and it doesn't call `setPrestartMode()`, the application will be started at boot time but it will appear on screen immediately. In other words, prestarting will be ignored without the mode being set.

- Application can check if it's in prestarted mode by calling `DuiApplication::isPrestarted()` method.

- Application will receive `DuiApplication::prestartReleased()` signal and `DuiApplication::releasePrestart()` -handler will be called when the application is released from the prestarted state. The application should start its runtime activities only after having received this signal. The application window is shown automatically.

- If application supports LazyShutdown, it will receive `DuiApplication::prestartRestored()` signal and `DuiApplication::releasePrestart()` -handler will be called when the application is returned to the prestarted state. The application should stop all its activity and reset its content so that the user won't notice a difference between a fresh launch and a lazy shutdowned application being released again.

- If you are sub-classing DuiApplicationService, you must call the base class implementations of `ApplicationService::launch()`, `ApplicationService::close()` and `ApplicationService::exit()` methods at the end of the possible re-implementations, otherwise prestarting functionality will be broken.

  Note: prestarted applications are not supposed to have more than one instance.

- In order to get the application prestarted during the boot, there should be a special control file in `/etc/prestart` directory. Applifed then reads these files and prestarts the applications at some point.

  Example of the contents of a .prestart -file:

  ```
  # This defines the prestartable application service
  Service=com.nokia.myapp
  ```

  (There's also an up-to-date example `/etc/prestart/example.prestart.ex`. This file comes with the applifed package.)

  Note: at this point it's not clear who should install and maintain the .prestart files. It might be possible that only applifed could install these files for predefined applications. This is just to prevent random third-party applications from prestarting.

- Developer also needs to add -prestart to the .service file:

  Example from a .service -file:

  ```
  -- clip --

  Exec=/usr/bin/myapp -prestart

  -- clip --
  ```

  This way the "normal" D-Bus launch (DuiApplicationService::launch()) will first prestart the application and immediately release it when it's ready. On the contrary, Applifed won't call DuiApplicationService::launch() so the application will be left in the prestarted state waiting for a release.

4 How to test prestarting

A developer can test the prestart functionality by starting the application with `-prestart` argument from the command line and then normally launch the application from Home Screen's application grid or with some simple helper program (e.g. dbus-send) that straightly calls the DuiApplicationService's launch() method of the desired service.

Note that the prestarted application must be in the same session bus with Home Screen when launching it from the application grid. Otherwise a new copy of the application will be launched. When working in a terminal, this can be ensured by:

```
$ source /tmp/session_bus_address.user
```

5 Code examples

Learning by example is always efficient. Here are a couple of example codes demonstrating how to use the prestart API. These programs are not complete so they cannot be compiled without some extra code.

Example of main() -function of application that supports TerminateOnClose -prestarting using Qt signals:

```
int main(int argc, char ** argv)
{
    DuiApplication app(argc, argv);

    // Use the LazyShutdown mode
    DuiApplication::setPrestartMode(Dui::TerminateOnClose);

    DuiApplicationWindow window;
    window.show();

    MainPage mainPage;
    mainPage.appearNow();

    // Run activateWidgets() here to setup things
    // if app is NOT prestarted, otherwise
    // connect it to prestartReleased() -signal
    // from DuiApplication so that it's run
    // at the time when the window is really being shown to the user.

    if (!app.isPrestarted()) {
        mainPage.activateWidgets();
    }
    else {
        app.connect(&app, SIGNAL(prestartReleased()),
                    &mainPage, SLOT(activateWidgets()));
    }
    return app.exec();
}
```

Example of main() -function of application that supports LazyShutdown -prestarting using Qt signals :

```
int main(int argc, char ** argv)
```

```
{
    DuiApplication app(argc, argv);

    // Use the LazyShutdown mode
    DuiApplication::setPrestartMode(Dui::LazyShutdown);

    DuiApplicationWindow window;
    window.show();

    MainPage mainPage;
    mainPage.appearNow();

    // Run activateWidgets() here to setup things
    // if app is NOT prestarted, otherwise
    // connect it to prestartReleased() -signal
    // from DuiApplication so that it's run
    // at the time when the window is really being shown to the user.

    if (!app.isPrestarted()) {
        mainPage.activateWidgets();
    }
    else {
        app.connect(&app, SIGNAL(prestartReleased()),
                    &mainPage, SLOT(activateWidgets()));

        // Stop and reset widgets when returning to the prestarted state
        app.connect(&app, SIGNAL(prestartRestored()),
                    &mainPage, SLOT(deactivateAndResetWidgets()));
    }
    return app.exec();
}
```

Example of application that supports LazyShutdown -prestarting using virtual handlers :

```
class MyApplication : public DuiApplication
{
    Q_OBJECT

public:
    MyApplication(int argc, char ** argv);
    ~MyApplication();

    //! Re-implementation
    virtual void releasePrestart();

    //! Re-implementation
    virtual void restorePrestart();
```

```
private:

    //! Main window
    DuiApplicationWindow * m_window;

    //! DuiApplicationPage -derived page
    MainPage * m_page;
};

MyApplication::MyApplication(int argc, char ** argv) :
    DuiApplication(argc, argv)
{
    // Use the LazyShutdown mode
    setPrestartMode(Dui::LazyShutdown);

    m_window = new DuiApplicationWindow;
    m_window->show();

    m_page = new DuiApplicationPage;
    m_page->appearNow();

    // Run activateWidgets() here to setup things
    // if app is NOT prestarted, otherwise it will be called
    // from the handler method.

    if (!isPrestarted()) {
        m_page->activateWidgets();
    }
}

void MyApplication::releasePrestart()
{
    // Your stuff here
    m_page->activateWidgets();

    // Call the default implementation to show the window.
    DuiApplication::releasePrestart();
}

void MyApplication::restorePrestart()
{
    // Your stuff here
    m_page->deactivateAndResetWidgets();

    // Call the default implementation to hide the window.
    DuiApplication::restorePrestart();
}

MyApplication::~MyApplication()
{
```

```
    delete m_window;
    delete m_page;
}

int main(int argc, char ** argv)
{
    MyApplication app(argc, argv);
    return app.exec();
}
```

Full version of lifecycle application that supports prestarting via signals can be found in 'examples' directory of libdui package.