Submitting a Bug Report

Overview

Most application developers will file a bug at http://bugs.java.com/. After checking that your bug hasn't already been reported, you can click on the “here” link on that page to go to the bug submission form. Choose “JavaFX” as the Product/Category and then select the appropriate Subcategory, Release, and Operating System.

JavaFX bugs are now part of the JDK Bug System (JBS) hosted at https://bugs.openjdk.java.net/. You can learn more about JBS here. You can browse the bug database without needing an OpenJDK account.

Filing a Bug

Most of this section applies whether you enter a bug via bugs.java.com or directly in JBS.

The most important thing you can do when entering a bug report is to ensure that the person who investigates it has enough information to recreate it. The best way to ensure this is to provide:

- a complete code sample (paste it in the comment section)
- a set of steps (numbered is best)

NOTE: If you are seeing multiple issues, please enter a bug report for each issue. A bug report that contains multiple independent problems can only be marked fixed when all of the problems it covers are fixed. If you have a single bug with what seems like many different manifestations, you can indicate this in the bug report.

Fortunately, JavaFX programs tend to be very short to providing a complete example, including main() is not particularly difficult. Code will not be released without a test case that demonstrates the problem. By numbering the steps in a description and stating what you should see at each step, it helps to focus the reader. Of course, not all problems can be described in this manner but a surprising amount can.

Here are some other things that are helpful in a bug report:

- a screen shot of the problem
- the exception or crash log
- platform specific information (ie. "only happens on Windows")

IMPORTANT: Bugs that are missing the information needed to recreate them will be closed as 'Incomplete'. The message that you should take away from this is "there is more work to do before this bug can be processed" and not "we do not care that there is a problem".

JBS-Specific Information

This section mainly applies to people filing bugs or working with bugs directly on JBS, but may be of interest to application developers browsing JBS.

Project / Component

Use the **JDK** Project in JIRA for FX bugs. This is the default.

Use the **javafx** Component (except for deployment or packager issues, which use the **deploy** Component)

Subcomponent

The javafx component is divided into subcomponents and these are used to assign default ownership of a bug report.

> *UPDATE THIS: Typically, many bugs are in 'controls', 'graphics', or 'base'. If you do not know the component, please make your best effort based on the component in Code Ownership.*

Select the Subcomponent that most closely matches the problem you are seeing, for example **controls** if you have a bug relating to UI controls.

Issue Types

Issues are classified by the following types:

1. **Bug** - A defect.
2. **Enhancement** - A small new feature, minor improvement, optimization, re-factoring, or code cleanup. Enhancements are usually small in scope. As a rule they should be no more than 2 weeks in duration, or else a JEP is needed (LINK TO JEP PAGE).
3. **Task** - A work item that does not change the repository. Examples: baselining the PRD, open source approvals or architecture reviews. Do not use this for a bug or other enhancement. If it touches any file in the repo, it isn't a Task!
4. **Sub-task** - A sub item of any of the above.

5. **Backport** - A port of a bug or enhancement to another release. For example, if you push a bug fix to 9 and want to backport it to 8u60 then you create a backport for the fix to 8u60.

IMPORTANT: Many processes are different for different issue types. Unless you are a committer, submit only **Bugs** or **Enhancements**.

More Detail About Issue Types

Sub-tasks are often used by developers to break a complex issue into more manageable parts. As such they are a convenience for the development teams. The release team does not track at the level of sub-tasks. To make this work, however, an issue should not have sub-tasks with different Fix For versions. If an issue is so complex that it spans multiple releases, it should be broken into several separate issues.

Summary

The summary field should contain a very short description of the problem such that it is possible to read it and understand the area that is affected and what the symptoms are. Phrases such as "buttons do not work" are not very helpful. Instead, look for better descriptions like "Push button draws garbage when pressed" or "Push button fails to send callback when pressed".

Sometimes, component owners and committers will add ad hock categorization in the summary field. For example, "[ListView] List does not deselect when Ctrl+Click pressed". Another common ad hock categorization is "[Windows]" or "[Linux]" to indicate quickly that a problem only happens on a certain platform.

If you add an ad hock categorization, it might be deleted or changed later by the component owner or another committer.

Affects Version

This should be the version that you are running / testing in which you found the Bug.

Fix Version

If this field is present, do not set it or alter it. This field is for use by committers only.

Description

This is the most important part of the bug. A bug that cannot be reproduced cannot be fixed. In fact, committers won't make code changes without a test case that shows the problem and then shows the that problem is gone after the fix is applied.

The best way to ensure that a bug can be recreated is to provide:

- a complete code sample
- a set of steps (numbered is best)

Here is an example:

//TODO - find good example in JIRA and point to it

Environment

This field is used to capture the platform where you are running.

//TODO - list environments

THE FOLLOWING INFORMATION IS OUT OF DATE...

Priorities

Priorities indicate the order in which issues should be resolved *relative to a release*. Priority encompasses the concepts of importance and urgency. The two concepts are highly correlated in that most of the time the worst bugs should be fixed first. There are cases, however, when an issue might be urgent but not important or vice versa.

The following priorities are available:

Priority	Urgency	Importance
P1 - Blocker	Emergency. Do it now.	Build or SWAT failures. Issues that stop 1 or more teams from all work.
P2 - Critical	Expected for next milestone.	Basic functionality unusable. Data loss, crashes, security problems or severe memory leaks.
P3 - Major	Expected for this release.	Major functionality unusable. No reasonable workarounds.
P4 - Medium	Optional for this release.	Problem limited in scope or with a reasonable workaround.
P5 - Minor	Optional for this release.	Very limited scope or with trivial workaround.

The purpose of priorities is to make it easier to find the issues to focus on. This principle has several consequences:

1. If the system is functioning properly, there should be more issues at a given priority level than at the next higher level. Think of it this way. The purpose of priority level *n* is to find the important stuff out of the pile of issues at priority level *n+1*. A priority level can't do that if there is little discrimination against neighboring levels. Or put another way, if everybody is somebody then nobody's anybody.
2. The distribution of priorities should apply to future releases as well as the current release. It is not helpful to insist that everything that is not done right away is therefore a low priority. That just makes it hard to find the important stuff in the future. This is why priorities are relative to a release. (This has some limits.)
3. Priorities may change over time. Neither importance nor urgency are immutable.
4. Priority values are under control of the Assignee.

Labels (a.k.a. keywords, tags)

In general, teams or individuals may label their issues as they see fit. There are, however, a set of common labels that are defined project wide. These labels should be used as defined and should not be used for other purposes.

//TODO - list of common labels