JBS Overview

JDK Bug System (JBS) Overview

Preface

The JDK bug system (JBS) hosted at http://bugs.openjdk.java.net/ is a JIRA instance which provides bug tracking for Projects in the OpenJDK Community. A large fraction of the bugs are associated with a JDK Release Project, such as JDK 8, and are resolved by changes to the source code for such a Project.

In November 2006, the source code for HotSpot and javac was relicensed and published as open source; this was followed by the relicensing as open source and publishing of the majority of the remaining source code for JDK 7 in May 2007. Since then, the OpenJDK Community has continued to work on various releases, including JDK 6, JDK 7 and its updates, and JDK 8.

While the history of the JDK source code under OpenJDK goes back to 2006 or so, the history of the JDK in its historical bug database stretches back to 1994. Tens of thousands of JDK-related bugs from Sun's legacy bug tracking system were migrated into JBS before JBS went live. Many of these bugs predate OpenJDK activities. By importing this collection of JDK bugs into JBS, the JBS bug database provides continuity of history, reaching back far beyond the start of OpenJDK.

Bugs for some closed source technologies included in Oracle's JDK, but not included in OpenJDK, are also tracked in JBS. This continues and expands a long-standing practice of publishing bug information about closed source code on the historical `bugs.sun.com`, a practice that is as old as Java itself.

This guide describes norms for using JBS; those with sufficient cause and authority can deviate from these norms. This guide concerns general OpenJDK development and is applicable to multiple releases. The detailed polices related to bug management have evolved over time and vary somewhat by domain. Documentation for a particular release or technology area should be consulted for logistics and details of working in that release or area.

Account Eligibility

OpenJDK Roles, Groups, and Projects are explained in the OpenJDK Bylaws. This JBS guide will use terms defined in the Bylaws; the Bylaws should be consulted for details.

An individual with at least one OpenJDK Project role of Author or higher has sufficient cause to get a JBS account. A JBS account grants an individual general read and write access to issues, including the ability to file new issues, transitioning issues among the states of the workflow, adding comments, changing field values (including adding and removing labels). The holder of a JBS account can also be the assignee of an issue.

A user's JBS username is his or her OpenJDK name. The password reset page can be used both to reset a lost password and to establish a an initial password.

At the time of launch, self-service account creation is not supported. Users without an account can browse JBS anonymously or use bugreport.java.com to view a time-delayed and simplified snapshot of bug state. Users without an account can also use bugreport.java.com to submit an issue. When such an issue is submitted, a record is created in the Java Incidents (JI) project in JBS; at the time of launch, the JI project is not publicly visible. Issues in the JI project have an identifier like JI-9XXXXXX, where the numeric portion corresponds to the bug identifier sent back to the submitter. After an initial triage process, if the incidents needs further review, it can be transferred to be an issue in the JDK project. When such a transfer occurs, the issue gets a new identifier in the JDK project (JDK-8YYYYYY) but references to the original JI-9XXXXXX number will be redirected.

Design considerations

The design of JBS was informed by Sun's legacy bug system used for JDK bugs, the use of JIRA by the JavaFX team, the OpenJDK bugzilla instance, and various other bug tracking systems. For JBS, the natural JIRA way for modeling a condition was chosen by default. Configuration changes were favored over customizations. Customizations were considered only when configuration changes could not achieve functionality deemed necessary for JBS.

Evolution and time horizon

JBS holds issues dating back nearly 20 years in the past and the information in JBS today is expected to be of interest for more than 20 years into the future. Therefore, a long-term and disciplined perspective will be used when evolving the system. For example, adding new fields will be approached with caution and the information stored in any new fields will need to be expected to be of interest in five to ten years time.

JIRA Version

At the time of external launch, JBS is running JIRA 5.2. For general questions about using JIRA, refer to the extensive documentation available for that JIRA release. Over time, JBS is expected to upgrade to newer versions of JIRA.

Projects

At the highest level, bugs in JBS are grouped into top-level *projects*. At external launch, there are two public projects in JBS:

- CODETOOLS: hosts bugs for jcheck, jtreg, and miscellaneous other utilities.
- JDK: host bugs for past, present, and future JDK release projects.

An OpenJDK Project may request that a JBS project be created to host its bugs by sending an email to ops@openjdk.java.net. In many cases, an existing JBS project should be used rather than creating a new JBS project. For example, a new JDK release would use the existing "JDK" project in JBS rather than creating a new project.

A new JBS project can request a component / subcomponent configuration suitable for that effort. A particular project should not expect to be granted further customizations, such as a novel set of status values.

Common Features across Projects

Issue Types

When filing a new bug, there are five feature types to choose from:

- Bug: a defect or flaw
- Enhancement: an improvement to an existing feature
- Task: An item whose resolution does not involve changing the source code repositories.
- Backport: used in JBS as part of multi-release support.
- New feature: Reserved for future use

The above issue types can by further refined into *Subtasks*. A Subtask is a issue type used to track breaking up an issue (not just a task!) into smaller pieces. The Subtasks can be resolved individually and then the parent issue can be resolved after all its Subtasks are resolved.

State model

Two fields represent the primary state of an issue, Status and Resolution. In JBS, a number of additional fields are used to hold substatus information: *Und erstanding* for substatus values of the *In Progress* Status and *Verification* for substatus information about how a bug resolved with a fix was transitioned to a *Closed* Status.

Status

The basic state model used in JBS includes five values for the Status field:

- After creation, issues start out in the *New* state.
- After triage, a New issue transitions to being *Open*
- Once work starts on an Open issue, it moves to *In Progress*. An In Progress issue can hold further substate information in the *Understanding* field to indicate *Cause Known* or *Fix Understood*.
- After an In Progress issue is resolved, it moves to the *Resolved* state. For JDK release Projects, typically the transition to Resolved is performed automatically as a result of pushing a fix to an Hg repository for a particular release.
- After being verified, Resolved bug moves to the *Closed* state. The Closed state is a terminal state and no further bug transitions are expected once a bug is closed.

Several additional fields record more specialized status information.

Understanding

When an issue has a status of In Progress, the Understanding field can be optionally used to further refine what is known about the bug:

- Cause Known: the cause of the issue has been determined.
- Fix Understood: in addition to the cause being known, the way to fix the problem is understood.

Resolution

Once an issue is Resolved or Closed, the Resolution field is also set. If an issue is resolved by changing a source code repository, the Resolution field is set to *Fixed*. Other commonly used Resolution values are *Won't Fix*, *Duplicate*, *Incomplete*, *Cannot Reproduce*, and *Not an Issue*.

Incomplete Issues

An incomplete issue is one where there is insufficient initial information for the assignee to make further progress. JBS distinguishes between two classes of incomplete issues: ones where more information is expected, and ones where more information is *not* expected.

An incomplete bug where more information is expected is modeled as:
Resolution = Incomplete, Status = Resolved
In this state, the submitter should provide additional information.

If no further information is provided in a reasonable period of time, an incomplete issue can transition to a closed state. An incomplete bug where more information is expected is *not* modeled as:
Resolution = Incomplete, Status = *Closed*

Closing as a Duplicate

Many issues are filed in JBS more than once. When the same issue is filed repeatedly, one issue should be closed as a duplicate of another. To close a bug as a duplicate:

- Select the Close workflow action.
- Set Resolution to Duplicate.
- Set Linked Issues to "duplicates"
- Add ID of reference bug (which might remain open)

Normally the issue with the most information should be kept open; this is often the oldest issue.

Verification

A bug that is resolved by a fix typically goes through a process to have the fix verified for correctness and completeness. If set, the Verification field has three possible values:

- Verified: the fix has been successfully verified. Verification might include checking that the regression test for a fix passes on all platforms of interest.
- Not verified: the verification process was skipped for the fix.
- Fix failed: the fix is faulty. (For JDK Release Projects, a failed fix is left in a state of (Status = Closed, Resolution = Fixed, Verification = Fix Failed) and a new bug is opened to cover the aspects of the fix that are failed, partial, or incorrect. In JDK Release Projects, an issue number can only be used for at most one push to a given repository. Therefore, if the original bug were reopened, it could not be used to push an updated version of the fix.)

Field Summary

Priority

The highest priority bugs are P1. The lowest priority bugs are P5. Specific criteria for determining priority may vary by project.

Textual fields

There are several fields in jbs which store textual information about an issue.

The *summary* is a one line title for an issue.

The *description* describes the circumstances behind an issue.

The *comment* field contains a set of comments, potentially from many parties, relating to the issue.

Who does what

The *reporter* is the party who filed an issue.

The *assignee* is the party who is responsible for making progress on developing a fix.

Version Management and Tracking

Each project defines a set of version values. These versions are then used in several fields to record information about the issue:

- fixVersion: what release is an issue intended to be fixed in/what release is an issue actually fixed in? (Although in JIRA, the fixVersion is multi-valued, in JBS fixVersion can semantically hold at most one value.)
- affectedVersion: what version(s) of a product are affected by an issue? This multi-valued field can hold multiple versions that are impacted by an issue. This field is informative and the set of releases listed is not intended to be exhaustive.
- Introduced in Version: what specific version was an issue introduced in? Setting this field may require non-trivial investigative work; however, it can provide crucial information for analyzing bug escapes.

Labels

Users can associate one or more labels with an issue. Such labels are often used to manage informal processes and record ad hoc information. In particular, for JDK release projects, there is a set of labels defined to explain why a bug fix omits a regression test.

Environmental fields

The OS (operating system), CPU (central processing unit), and Environment fields store environmental information about the system affected by an issue.

- OS: there are several dozen operating system values to choose from:
    - generic: believed to impact all/most operating systems
    - linux*: a Linux version or variant
    - iOS, OSX: Apple operating systems
    - solaris*: a Solaris version or variant
    - windows*: a MS Windows version or variant
    - other: believed to impact a particular OS not available as an option
- CPU: there are a handful of particular processor values to choose from:
    - generic: believed to affect all CPU values
    - x86, sparc, arm, ppc, itanium, etc.: distinct processor families
    - other: believed to affect some particular specific CPU value not available as an option
    - unknown: no information on CPU type

For both the OS and CPU fields, when more detailed information is available and relevant, such as particular OS patch levels or CPU steppings/versions, the free-form Environment field should be used to store the additional information.

JQL Customizations

JIRA supports both simple and advanced queries. Simple queries are composed through menu selection; advanced queries are written in the JQL a query language used within JIRA. JBS includes several extensions to default JQL:

- regexLabel: Matches labels with a regular expression.
  Example: labels stating with "8":

  ```
  labels in regexLabel ("8.*")
  ```

- regexOption: Matches option with a regular expression.
  Example: Finds all issues where the os field starts with "solaris":

  ```
  os in regexOption ("solaris.*")
  ```

- regexVersion: Matches version field types with a regular expression.
  Example: Finds all issues with a fixed version starting with "7u"

  ```
  fixVersion in regexVersion ("7u.*")
  ```

Multi-release tracking

In stock JIRA, the fixVersion/s field is used for tracking whether or not a fix has been applied to multiple releases. In other words, the status of a fix across all releases is stored in a single issue. For the purposes of the JDK, this was judged as not providing an adequate level of detail. Information of interest for JDK bug tracking across releases includes

- the release
- assignee for the release
- status and resolution for the release
- release-specific comments

Storing this information allows the bug database to easily answer questions such as:

- Does an issue impact the release I'm managing? / What releases does the issue need to be addressed in?
- Who needs to fix this issue? / Am I on the hook for fixing this issue?

The solution above is implemented in JBS using a custom *Backport* issue type along with a custom Backport link type. Backports are full issues, but only a few fields are expected to be set:

- fixVersion: indicates which specific release a backport is targeting or fixed in
- assignee: holds the engineer responsible for the backport
- Status and Resolution: hold the status and resolution for the release
- Release specific comments: such as ones added by Hg Updater

When viewing the master bug, its backports are grouped together in a tabular format.

As used with Backports, semantically the Fix Version/s field has a single value. This constraint is not currently enforced in the database used by JBS, but may be enforced in the future.

The bug numbers of backports should *not* be used in changeset comments when a fix is pushed. The bug number of the master issue should be used in changeset comments for all releases into which a fix is pushed.

(Note that a backport from an earlier release to later one, such as JDK 7u6 to JDK 8, is just a backport going in a negative direction.)

Further Note: The 'hgupdate-sync' label is used to denote bug records which are already fixed in a previous release. When code lines are synced a new backport record will be created with the hgupdate-sync label to capture the sync activity. For the most part, such records can be ignored since they indicate that the issue was resolved in an earlier update release.

JDK Project

Much of the work in OpenJDK occurs in JDK release Projects which use the "JDK" project in JBS. Therefore, This section will discuss some of the particulars of the JDK project.

JDK Development Practices

Typically, a JDK release Project will have a series of source code repositories:

- Developer repositories: managed by developers
- Integration repositories: where developers push their fixes, managed by integrators
- Master repository: after additional vetting by an integrator, fixes from the integration repositories are pushed into master, managed by release engineering

At a regular cadence, say weekly, builds of the master repository are created, validated, and then distributed as promoted builds.

JBS Bug Ranges

Bugs in the JDK project with a numeric value of 8000000 or higher were created after the transition to JIRA. Bugs with numeric values less than 8000000 were imported from the Sun legacy bug tracking system, preserving the numeric value on import. As the legacy system had a somewhat different design, there may be artifacts from the import. Bugs in the 2000000-2999999 range are for *subCRs*, which were analogous to backports in the Sun legacy system.

Security Vulnerabilities

To prevent exploits in the wild while a fix is being developed, security vulnerabilities use a different bug handling process than other bugs. In particular, security vulnerabilities do not have publicly visible issues in JBS. A suspected security vulnerability in the JDK should be reported using the Oracle Security procedures rather than creating a bug in JBS.

Classification Scheme

Within a project, JBS use a two-level component / subcomponent classification scheme. (Subcomponents are a JBS customization not found in standard JIRA.) New components are expected to be added only very infrequently. To ease long-term use of JBS, new subcomponents will only be added judiciously. By default, an existing subcomponent should be used for new work if possible. To be considered for a subcomponent, an area should be expected to have at least fifty bugs and anticipated to be of interest for several years.

The main top-level components of the JDK product (with main corresponding OpenJDK lists, if any) include:

- core-libs (core-libs-dev, nio-dev)
- client-libs (2d-dev, awt-dev, swing-dev, beans-dev, sound-dev)
- security-libs (security-dev)
- other-libs
- deploy
- install
- tools (language-related tools discussed on compiler-dev)
- hotspot (hotspot-dev, hotspot-compiler-dev, hotspot-gc-dev, hotspot-runtime-dev)
- core-svc (serviceability-dev)

For the libs areas, the primary name of the subcomponent will be the package of the API in question. When a more detailed classification within the technology area of a package is needed, the package name will be followed by a colon (":") and a name for the more detailed area. For example, in the core-libs component, there will be subcomponents like:

- java.lang
- java.lang:class_loading
- java.math
- java.util
- java.util:i18n

In the tools component, subcomponents will primarily correspond to command names in $JDK/bin like, jar, javac, and javap.

Other material particular to the JDK project

For JDK release Projects, an Hg Updater process on the Mercurial servers largely automates the setting of several fields. When a fix is pushed to a team integration repository (like TL), Hg Updater sets:

- Status = Resolved
- Resolution = Fixed
- Resolved in Build = *team*
- Adds a comment including the changeset URL

When an integrator pushes a fix to master, Hg Updater sets:

- Status = Resolved
- Resolution = Fixed
- Resolved in Build = *master*
- Adds a comment including the changeset URL

When release engineering promotes a build, Hg Updater sets Resolved in Build = $BUILD_NUMBER for the bugs fixed in that build.

A result of the Hg updater automation is that once a development engineer has pushed a fix to an integration repository, that engineer is not expected to update the Status value further for the lifetime of the bug.

If a bug tracked in the JDK project is addressed by making a change in a non-JDK repository (such as a doc repository or web staging SCM/CMS), the "Inapplicable" Resolved-in-build value is used.

There are a number of special version values:

- tbd_major and tbd_minor: used to target a fix to some unspecified major or minor release, respectively.
- 7-pool, 8-pool, etc.: some unspecified future update release in a particular update train.

When a bug is actually fixed, the fixVersion field must be changed to a specific release value rather than some general value. The Hg updater process will perform this task.

The special version values like tbd_minor should *not* be used in the affectedVersion field; only version values corresponding to actual releases should be used.

When a JBS issue is discussed on an OpenJDK mailing list, such as a review request, it is recommended to include a link to the OpenJDK thread in a comment in the JBS issue.

Questions and Comments

General questions about JBS can be sent to discuss@openjdk.java.net.

Feature requests for JBS itself can be sent to ops@openjdk.java.net.

Conclusion

> Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.
> — Douglas Hofstadter, Gödel, Escher, Bach: An Eternal Golden Braid

http://mail.openjdk.java.net/mailman/listinfo/core-libs-dev