Open CASCADE Technology

Contribution Workflow

CONTENTS

1. INTRODUCTION

The purpose of this document is to describe standard workflow for processing contributions to certified version of OCCT.

1.1. Use of issue tracker system

Each contribution should have corresponding issue (bug, or feature, or integration request) registered in the MantisBT issue tracker system accessible by URL http://tracker.dev.opencascade.org. The issue is processed further according to the described workflow.

1.2. Access Levels

Access level defines the permissions of the **user** to view, register and modify issues in a Mantis bugtracker. The correspondence of **access level** and user permissions is defined in accordance with the table below.

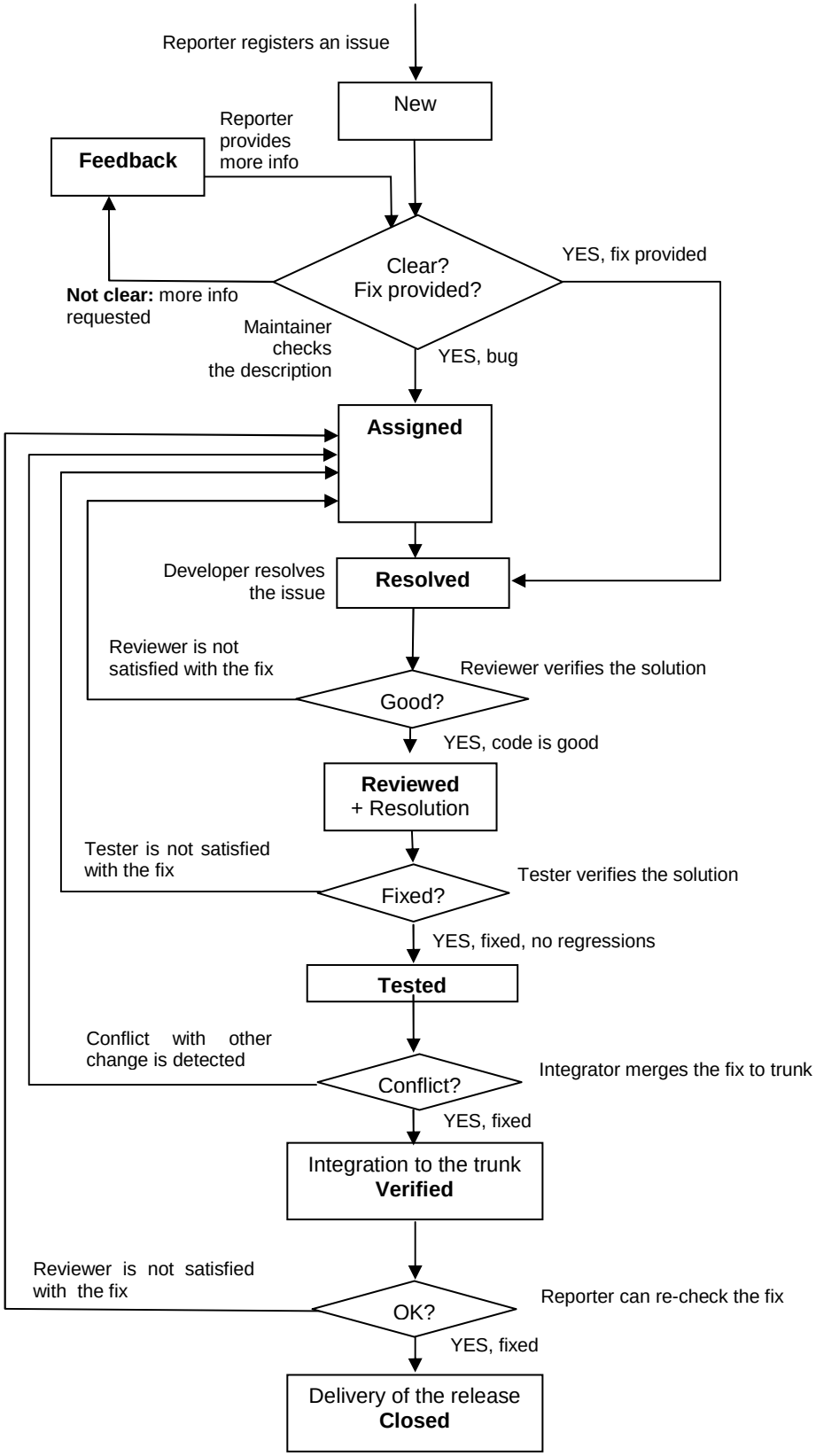
Access level	Granted to	Permissions	Can set statuses
Viewer	Everyone (anonymous access)	View public issues only	-
Reporter	Users registered on dev.opencascade.com	View, report, and comment issues	New, Resolved
Updater	Users of dev.opencascade.com in publicly visible projects	View and comment issues	New, Resolved
Developer	OCC developers and external contributors who signed CLA	View, report, modify, and handle issues	New, Assigned, Resolved, Reviewed
Tester	OCC engineer devoted to certification testing	View, report, modify, and handle issues	Assigned, Tested
Manager	Person responsible for a project or OCCT component	View, report, modify, and handle issues	New, Resolved, Reviewed, Tested, Closed

According to his access level, the user can participate in the issue handling process under different roles, as described below.

2. TYPICAL WORKFLOW FOR AN ISSUE

2.1. General scheme

A typical life cycle of an issue is represented in the scheme below. See also the description after the scheme.



2.2. Issue registration

An issue is registered in **Mantis** bugtracker by **Reporter** with definition of the necessary attributes.

The definition of the following attributes is obligatory:

- **Category**

  Indicates component of OCCT to which the issue relates. If in doubt, assign OCCT:Foundation Classes.

- **Reproducibility**
- **Severity**
- **Priority**
- **Profile**

  Profile option allows defining the platform on which the problem was detected from the list of predefined platforms. If a platform is absent in the list of predefined platforms it is possible to use **Or Fill In** option to define the platform manually.

  - **Platform**
  - **OS**
  - **OS Version**

- **Products Version**

  This attribute defines the version of Open CASCADE on which the problem has been detected.

- **Summary**

  Summary should be a short, one sentence description of the issue. It has a limit of 128 characters. It should be informative and useful for the developers. It is advisable to avoid vague or misleading phrases, such as "it doesn't work" or "it crashed". It is not allowed to mention the issue originator, and in particular the customer, in the name of the registered issue.

- **Description**

  The Description should contain a detailed definition of the nature of the registered issue depending on its type. For a bug it is required to submit a detailed description of the incorrect behavior, including the indication of the cause of the problem (if possible at this stage) or any inputs from the originator. For a feature or integration request it is recommended to describe the proposed feature in details (as possible at that stage), including the changes required for its implementation and the main features of the new functionality. Filling the bug description is obligatory.

- **Steps To Reproduce**

  In this field it is possible to describe in detail how to reproduce the issue. This field considerably helps to find the cause of the problem, to eliminate it and to create the test case.

  It is recommended to attach a prototype test case in form of a Tcl script for DRAW, using either existing DRAW commands, or a C++ code which can be organized in DRAW commands, as well as sample shapes or other input data (if applicable), immediately after the issue registration.

The additional field **Upload File** allows attaching the shapes, scripts or modified source files of OCCT.

The newly registered issue gets status **NEW** and is assigned to the developer responsible for the OCCT component indicated in the Category field (**Maintainer**).

2.3. Assigning the issue

The description of the new issue is checked by **Maintainer** and if it is feasible, he may assign the issue to some **Developer**. Alternatively, any user with access level developer or higher can assign the issue to himself if he want to provide a solution. The recommended way to handle contributions is that **Reporter** assigns the issue to himself and provides solution.

The **Maintainer**, **Technical Project Manager**, or **Bugmaster** can close or reassign the issue (in **FEEDBACK** state) to the **Reporter** after it has been registered, if its description does not contain sufficient details to reproduce the bug or explain the purpose of the new feature**.** That decision shall be documented in the comments to the issue in the Bugtracker.

The assigned issue should have state **ASSIGNED**.

2.4. Resolving the issue

The **Developer** responsible for the issue assigned to him provides solution as a change on the version of OCCT indicated in the issue attributes, or last development version.

The modified sources should be submitted for review and testing to the dedicated branch of the official OCCT Git repository:

- Branch should be created for the issue with name composed of letters 'CR' followed by issue ID number (without leading zeroes). Optional suffix can be added to the branch name after issue ID, e.g. to distinguish between several version of the fix.

- The branch should be based on recent version of the master branch (not later than commit tagged as last OCCT release).

- The first line of the first commit message should contain the Summary of the issue (starting with its ID followed by colon, e.g. "0022943: Bug TDataXtd_PatternStd"). The consequent lines should contain a description of the changes made. If more than one commit has been made, the commit messages should contain description of the changes made.

- The amount of the code affected by the change should be limited to only the changes required for the bug fix or improvement. Change of layout or re-formatting of the existing code is allowed only in the parts where meaningful changes related to the issue have been made.

- The name of the branch where the fix is submitted should be given in the note to the Mantis issue (providing the direct link to relevant branch view in GitWeb is encouraged).

- The description of the changes made should be put to the field "Additional information and documentation updates" of the Mantis issue.

In some cases (if Git is not accessible for the contributor), external contributions can be submitted as patch (diff) files or sources attached to the Mantis issue, with indication of OCCT version on which the fix is made. Such contributions should be put to Git for processing by someone else, and hence they have less priority in processing than the ones submitted directly through Git.

The issue for which solution is provided should be switched to **RESOLVED** state and assigned to the developer who is expected to make a code review (**Reviewer**; by default, can be set to **Maintainer** of the component).

2.5. Code review

The **Reviewer** analyzes the proposed solution for applicability in accordance with OCCT Code reviewing rules and examines all changes in the sources to detect obvious and possible errors, misprints, conformity to coding style.

- If **Reviewer** detects some problems, he can either:
    - Fix these issues and provide new solution, reassigning the issue (in **RESOLVED** state) to the **Developer**, who then becomes a **Reviewer**. Possible disagreements should be resolved through discussion, which is done normally within issue notes (or on the OCCT developer's forum if necessary).
    - Reassign the issue back to the **Developer**, providing detailed list of remarks. The issue then gets status **ASSIGNED** and new solution should be provided.

- If **Reviewer** does not detect any problems, he changes status to **REVIEWED**.

2.6. Testing

The issues that are in **REVIEWED** state are subject of certification (non-regression) testing.

The issue is assigned to OCC **Tester** when he starts processing it.

The results of tests are checked by the **Tester**:

- If the **Tester** detects build problems or regressions, he changes the status to **ASSIGNED** and reassigns the issue to the **Developer** with a detailed description of the problem. The **Developer** should produce a new solution.
- If the **Tester** does not detect build problems or regressions, he changes the status to **TESTED** for further integration.

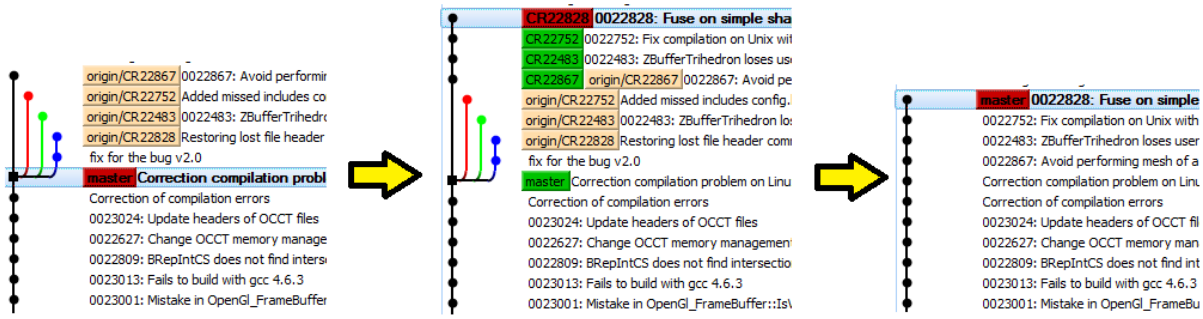
2.7. Integration of a solution

Before integration into the master branch of the repository the **Integrator** checks the following conditions:

- the change has been reviewed;
- the change has been tested without regressions (or with regressions treated properly);
- the test case has been created for this issue (when applicable), and the change has been rechecked on this test case;
- "**Additional information and documentation updates**" field is filled by the developer;
- the change does not conflict with other changes integrated previously.

If the result of check is successful the Integrator integrates solution into the master branch of the repository.

Each change is integrated into the master branch as a single commit without preserving the history of changes made in the branch (by rebase, squashing all intermediate commits), however, preserving the author when possible. This is done to have the master branch history plain and clean. The following picture illustrates the process:



Integration of several branches

The new master branch is tested against possible regressions that might appear due to interference between separate changes. When the tests are Ok, the new master is pushed to the official repository and the original branches are removed from it.

The issue status is set then to **VERIFIED** and is assigned to **Reporter** so that he could check the fix as-integrated.

2.8. Closing a bug

The **Bugmaster** closes the issue after regular **OCCT Release** provided that the issue status is **VERIFIED** and that issue was really solved in that release, by rechecking the corresponding test case.

The final issue state is **CLOSED**.

2.9. Reopening a bug

If a regression is detected, the **Bugmaster** may reopen and reassign the **CLOSED** issue to the appropriate developer with comprehensive comments about the reason of reopening. The issue then becomes **ASSIGNED** again.

3. APPENDIX

3.1. Issue attributes

3.1.1. Severity

Severity shows at which extent the issue affects the product.
The list of used severities is given in the table below in the descending order.

Severity	Description	Weight for Bug Score
crash	Crash of the application or OS, loss of data	5
block	Regression corresponding to the previously delivered official version. Impossible operation of a function on any data with no work-around Missing function previously requested in software requirements specification. Destroyed data.	4
major	Impossible operation of a function with existing work-around. Incorrect operation of a function on a particular dataset. Impossible operation of a function after intentional input of incorrect data. Incorrect behavior of a function after intentional input of incorrect data.	3
minor	Incorrect behavior of a function corresponding to the description in software requirements specification. Insufficient performance of a function.	2
tweak	Ergonomic inconvenience, need of light updates.	1
text	Inconsistence of program code to the Coding Standard. Errors in source text (e.g. unnecessary variable declarations, missing comments, grammatical errors in user manuals).	1
trivial	Cosmetic bugs.	1
feature	Bug fix, new feature, improvement that requires workload estimation and validation.	0
integration request	Requested integration of an existing feature into the product.	0
Just a question	A question to be processed, without need of any changes in the product.	0

3.1.2. Statuses of issues

The bug statuses that can be applied to the issues are listed in the table below.

Status	Description
New	New just registered issue. Testing case should be created by **Reporter**.
Feedback	The issue requires more information; the original posters should pay attention.
Assigned	Assigned to a developer.
Resolved + one of resolutions	The issue has been fixed, and now is waiting for revision.
Revised	The issue has been revised, and now is waiting for testing.

+ one of resolutions	
Tested	The fix has been internally tested by the tester with success on the full non-regression database or its part and a test case has been created for this issue.
Verified	The fix has been integrated into the trunk of corresponding repository
Closed	The fix has been integrated to a trunk. The corresponding testing case has been executed successfully. The issue is no longer reproduced.

3.1.3. Resolutions

Resolution is set when the bug is resolved. “Reopen” resolution is added automatically when the bug is reopened.

Resolution	Description
Open	The issue is being processed.
Fixed	The issue has been successfully fixed.
Reopened	The bug has been reopened because of insufficient fix or regression.
Unable to reproduce	The bug is not reproduced.
Not fixable	The bug cannot be fixed because it is a bug of third party software, or because it requires more workload than it can be allowed.
Duplicate	The bug for the same issue already exists in the tracker.
Not a bug	It is a normal behavior in accordance with the specification of the product
No change required	The issue didn’t require any change of the product, such as a question issue
Suspended	This resolution is set for Acknowledged status only. It means that the issue is waiting for fix until a special administrative decision is taken (e.g. a budget is not yet set in accordance with the contract)
Documentation updated	The issue was a normal behavior of the product, but the actions of the user were wrong. The specification and the user manual have been updated to reflect this issue.
Won’t fix	An administrative/contractual decision has been taken to not fix the bug

3.2. Update and evolution of documentation

The documentation on Open CASCADE Technology currently exists in three forms:

- OCCT Technical Documentation generated automatically with Doxygen tool on the basis of comments in CDL or HXX files.
- User’s Reference Documentation on OCCT packages and Products supplied in the form of PDF User’s guides
- OCCT Release Documentation supplied in the form of Release Notes with each release.

It is strictly required to properly report the improvements and changes introduced in OCCT in all three forms of Documentation.

3.2.1. Maintenance of CDL files

Every developer providing a contribution to the source code of OCC should make a relevant change in the corresponding header file, including CDL.

Making the appropriate comments is mandatory in the following cases:

- Development of a new package/class/method/enumeration;
- Modification of an existing package/class/method/enumeration that changes its behavior;
- Modification / new development impacts at other packages/classes/methods/enumerations, the documentation of which should be modified correspondingly.

The only case when the comments may be not required is introducing a modification that does not change the existing behavior in any noticeable way or brings the behavior in accordance with the existing description.

CDL description must be in good English, containing as much relevant information and as **clear** as possible. If the developer is unable to properly formulate his ideas in English or suspects that his

description can be misunderstood, he should address to the Documentation Engineer for language assistance. Such action is completely subject to the discretion of the developer; however, the Documentation Engineer can require that the developer should provide a relevant technical documentation and reopen a bug until all documentation satisfies the requirements above.

3.2.2. Maintenance of the User's Reference Documentation

The User's Reference Documentation is distributed among a number of User's Guides, each describing a certain module of OCCT. The User's Guides do not cover the entire functionality of OCCT; however, they describe most widely used and important packages.

In most aspects the User's Guides present the information that is already contained in CDL descriptions for methods, classes, etc., only with different formatting,

Thus, it is required that any developer who implements a new or modifies an existing package/class/method/enumeration and adds a description of new development or changes in the corresponding CDL file should also check if this class package/class/method/enumeration or the package/class to which the added class/method belongs is already described in the documentation and update the User's Reference Documentation correspondingly.

3.2.3. Preparation of the Release Documentation

Before changing the bug Status to RESOLVED, the developer should provide a description of the implemented work using the "**Additional information and documentation updates**" field of Mantis bugtracker.

This description is used for the Release Documentation and has the following purposes:

- to inform the OCCT users about the main features and improvements implemented in the platform in the release;
- to give a complete and useable list of changes introduced into the OCCT since the latest version.

The changes should be described from the user's viewpoint so that the text could be comprehensible even for beginners having a very vague idea about OCCT. If the developer is unable to properly formulate his ideas in English or suspects that his description can be misunderstood, he should address to the Documentation Engineer for language assistance. Such action is completely subject to the discretion of the developer; however, the Documentation Engineer can require that the developer should provide a relevant technical documentation and reopen a bug until all documentation satisfies the requirements.

Note: It is required to single out the changes in the OCCT behavior as compared to the previous versions and especially the changes to be considered when porting from the previous version of OCCT.

For example:

If global macros XXX() was used in the code of your application, revise it for direct use of the argument stream object.

or:

You might need to revise the code related to text display in 3d viewer to take into account new approach of using system fonts via XXX library.

The Documentation Engineer is responsible for preparation of the version Release Notes and update of the User's Guides. If the Documentation Engineer considers that the description currently provided by the Developer is somehow inadequate or unsatisfactory he can demand the Developer to rewrite the documentation with the Documentation Engineer's assistance.