Open CASCADE Technology 7.0.0.dev

Contribution Workflow

December 18, 2015

Contents

1 Introduction 1
1.1 Use of issue tracker system 1
1.2 Access levels 1

2 Standard workflow for an issue 2
2.1 General scheme 2
2.2 Issue registration 2
2.3 Assigning the issue 4
2.4 Resolving the issue 4
2.4.1 Requirements to the code modification 4
2.4.2 Providing a test case 5
2.4.3 Updating user and developer guides 5
2.4.4 Submission of change as a Git branch 5
2.4.5 Requirements to the commit message 6
2.4.6 Marking issue as resolved 9
2.5 Code review 9
2.6 Testing 10
2.7 Integration of a solution 10
2.8 Closing a bug 11

3 Additional workflow elements 12
3.1 Requesting more information or specific action 12
3.2 Defining relationships between issues 12
3.3 Submission of a change as a patch 12
3.4 Updating branches in Git 12
3.5 Minor corrections 13

4 Appendix: Issue attributes 14
4.1 Category 14
4.2 Severity 14
4.3 Status 15
4.4 Resolution 15

(c) Open CASCADE 2015

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 according to the described workflow.

1.2 Access levels

Access level defines the permissions of the user to view, register and modify issues in the issue tracker. The correspondence of access level and user permissions is defined in the table below.

Access level	Granted to	Permissions	Can set statuses
Viewer	Everyone (anonymous access)	View public issues only	None
Updater	Users registered on dev.opencascade.com, in Open CASCADE project	View and comment issues	None
Reporter	Users registered on dev.opencascade.com, in Community project	View, report, and comment issues	New, Resolved, Feedback
Developer	OCC developers and (in Community project) external contributors who signed the CLA	View, report, modify, and handle issues	New, Assigned, Resolved, Reviewed, Feedback
Tester	OCC engineer devoted to certification testing	View, report, modify, and handle issues	Assigned, Tested, Feedback
Maintainer	Person responsible for a project or OCCT component	View, report, modify, and handle issues	New, Resolved, Reviewed, Tested, Closed, Feedback
Bugmaster	Person responsible for Mantis issue tracker, integrations, certification, and releases	Full access	All statuses

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

(c) Open CASCADE 2015

2 Standard workflow for an issue

2.1 General scheme

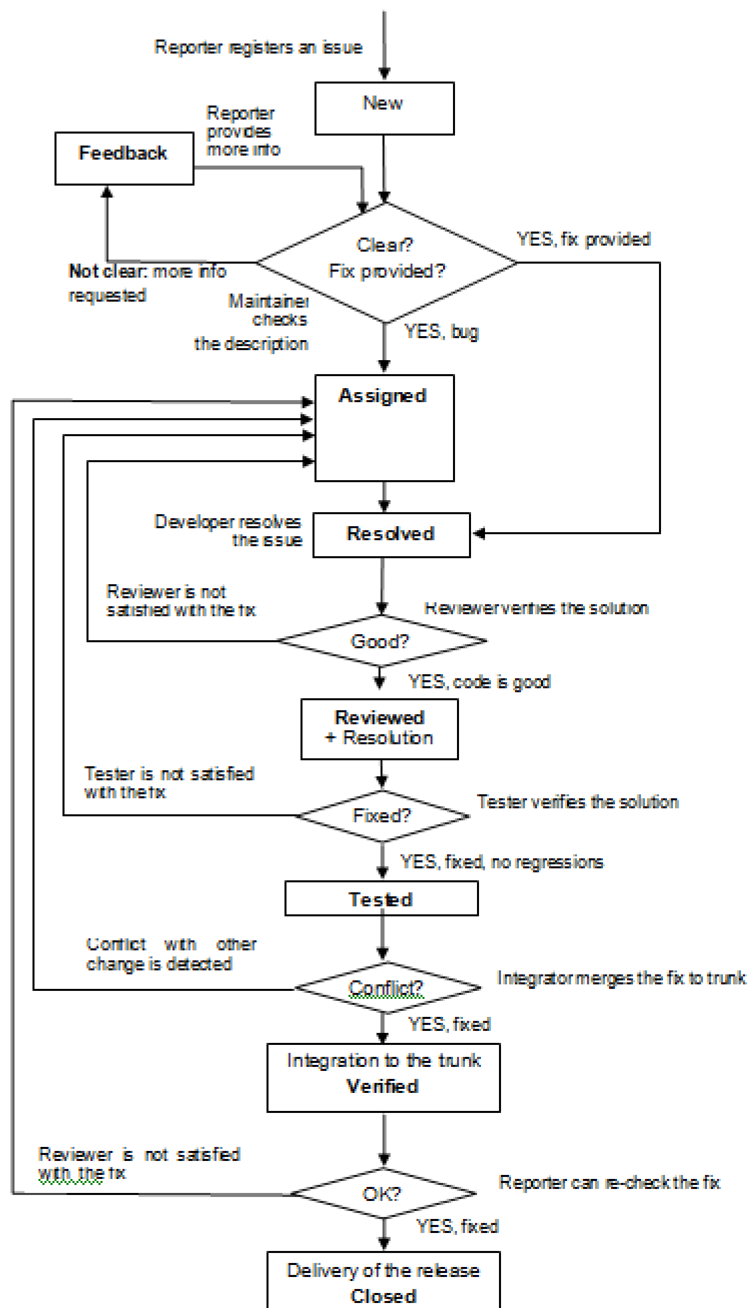

Figure 1: Standard life cycle of an issue

2.2 Issue registration

An issue is registered in Mantis bugtracker by the **Reporter** with definition of the necessary attributes (see also Appendix):

Category - indicates component of OCCT to which the issue relates. (If in doubt, assign OCCT:Foundation Classes.)

Severity - indicates the impact of the issue in the context where it was discovered.

(c) Open CASCADE 2015

Profile - specifies the configuration on which the problem was detected. For specific configurations, it is possible to specify separately platform, OS, and version.

These fields can be left empty if the issue is not configuration-specific; otherwise additional details relevant for the environment where the issue is reproduced (such as compiler version, bitness etc.) can be provided in **Description**.

Products Version - defines version of Open CASCADE on which the problem has been detected.

It is preferable to indicate the version of the earliest official release where the problem can be reproduced (as far as it is known). If the issue is reported on current development version of OCCT, number of version currently developed should be used (for convenience, this version is marked by asterisk in the list).

Note

> OCCT version number can be consulted in the file Standard_Version.hxx (value of OCC_VERSION_COMPLETE macro).

Assign to - developer to whom the issue will be assigned. By default, set to **Maintainer** of the OCCT component selected in **Category** field.

Target Version - defines target version for the fix to be provided. By default, set to the current version under development.

Summary - a short, one sentence description of the issue.

Summary has a limit of 128 characters. It should be informative and useful for the developers. It is not allowed to mention the issue originator, and in particular the customer, in the name of the registered issue.

A good practice is to start the issue with indication of the relevant component (OCCT module, package, class etc.) to better represent its context.

The summary should be given in imperative mood when it can be formulated as goal to be achieved or action to be done. In particular, this applies to feature requests and improvements, for instance:

```
Visualization - Implement level of detail API in AIS
```

If the issue reports a problem, summary should be given in Present Simple. If reported problem is believed to be a regression, it is recommended to indicate this in the summary, like this:

```
[Regression in 6.9.0] IGES - Export of a reversed face leads to wrong data
```

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 much as possible at that stage), including the changes required for its implementation and the main features of the new functionality.

Example:

```
3rd-party library New_Image has been implemented in OCCT.
This library provides build-in support of popular image processing formats, such as JPEG, PNG, GIF or BMP.
It should replace all obsolete code for image load/dump in the future.
Currently it is used to save images from Window / OpenGL context.
```

Note

> In the description and notes to the issues, you can refer to another issue by its ID prefixed by number sign (e.g.: #12345), and refer to a note by its ID prefixed by tilde (e.g.: ∼20123). These references will be expanded by Mantis to links to corresponding issue or note. When number sign or tilde followed by digits are part of a normal text, add a space before digits (e.g.: "face # 12345 contains ∼ 1000 edges") to avoid this conversion.

Steps To Reproduce - in this field it is possible to describe in detail how to reproduce the issue.

(c) Open CASCADE 2015

Filling this field considerably helps to find the cause of the problem and to create the test case. The optimal approach is to give a sequence of DRAW Test Harness commands to reproduce the problem in DRAW. Alternatively, Tcl script can be attached to the issue.

Additional information and documentation updates - any additional information, remarks to be taken into account in Release Notes, etc..

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

This field can be used to attach a prototype test case in form of a Tcl script for DRAW, a C++ code which can be organized in DRAW commands, sample shapes, or other data required for reproduction of the issue. Where applicable, pictures demonstrating a problem and/or desired result can be attached.

The newly registered issue gets status **NEW** and is assigned to the person indicated in the **Assign to** field.

2.3 Assigning the issue

The description of the new issue is checked by the **Maintainer** and if it is feasible, he may assign the issue to a **Developer**. Alternatively, any user with **Developer** access level or higher can assign the issue to himself if he wants to provide a solution.

The recommended way to handle contributions is that the **Reporter** assigns the issue to himself and provides a solution.

The **Maintainer** 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 has status **ASSIGNED**.

2.4 Resolving the issue

The **Developer** responsible for the issue assigned to him provides a solution including:

- Changes in the code, with appropriate comments
- Test case (when applicable) and data necessary for its execution
- Changes in the user and developer guides (when necessary)

The change is integrated to branch named CRxxxxx (where **xxxxx** is issue number) in the OCCT Git repository, based on current master, and containing single commit with appropriate description. Then the issue is switched to **RESOLVED** for further review and testing.

The following sub-sections describe this process, relevant requirements and options, in more details.

2.4.1 Requirements to the code modification

The amount of code affected by the change should be limited to 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.

Note

> If deemed useful, re-formatting or cosmetic changes affecting considerable parts of the code can be made within a dedicated issue.

Every developer providing a contribution to the source code of OCC should make the appropriate comments in the code to ensure its clarity and maintainability. 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 or new development impacting other packages / classes / methods / enumerations, the documentation which of should be modified correspondingly.

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

The comments must be in plain English (or Simplified Technical English), containing as much relevant information and as clear as possible.

The modification should be tested by running OCCT tests (on the platform and scope available to **Developer**) and ensuring absence of regressions. In case if modification affects results of some existing test case and the new result is correct, such test case should be updated to report OK (or BAD), as descibed in Automated Test System / Interpretation of Test Results.

2.4.2 Providing a test case

For modifications affecting OCCT functionality, a test case should be created (unless already exists) and included in the commit or patch. See Automated Test System / Creating a New Test for relevant instructions.

The data files required for a test case should be attached to corresponding issue in Mantis (i.e. not included in commit).

When test case cannot be provided for any reason, maximum possible information on how the problem can be reproduced and how to check the fix should be provided in the **Steps to Reproduce** field of an issue.

2.4.3 Updating user and developer guides

If change affects functionality described in User Guides, corresponding user guide should be updated to reflect the change.

If change affects OCCT test system, build environment, or development tools described in Developer Guides, corresponding guide should be updated.

Changes that break compatibility with previous versions of OCCT (i.e. affecting API or behavior of existing functionality in the way that may require update of existing applications based on previous official release of OCCT to work correctly) should be described in the document Upgrade from previous OCCT versions. It is recommended to add a sub-section for each change described. The description should provide explanation of the incompatibility introduced by the change, and describe a way how it can be resolved (at least, in known situations).

When feasible, automatic upgrade procedure (adm/upgrade.tcl) can be extended by a new option to perform required upgrade of the dependent code automatically.

2.4.4 Submission of change as a Git branch

The modification of sources should be provided in the dedicated branch of the official OCCT Git repository.

The branch should contain single commit, with appropriate commit message (see Requirements to the commit message below).

In general, this branch should be based on the recent version of the master branch. It is highly preferable to submit changes basing on the current master. In case if the fix is implemented on previous release of OCCT, the branch can be based on the corresponding tag in Git, instead of master.

The branch name should be composed of letters **CR** (abreviation of "Change Request") followed by the issue ID number (without leading zeros). It is possible to add an optional suffix to the branch name after the issue ID, e.g. to distinguish between several versions of the fix (see Updating branches in Git).

See Guide to using GIT for help on using Git.

(c) Open CASCADE 2015

Note

When branch with name given according to the above rule is pushed to Git, a note is automatically added for the corresponding issue in Mantis, indicating person who made push, commit hash, and (for new commits) description.

2.4.5 Requirements to the commit message

The commit message posted in Git constitutes an integral part of both the fix and the release documentation.

The first line of the commit message should contain the Summary of the issue (starting with its ID followed by colon, e.g. "0022943: Bug in TDataXtd_PatternStd"), followed by an empty line.

The following lines should provide a description of the context and details on the changes made.

The contents and the recommended structure of the description depend on the nature of the bug.

In a general case, the following elements should be present:

- **Problem** – a description of the unwanted behavior;
- **Change** – a description of the implemented changes, including the names of involved classes / methods / enumerations etc.;
- **Result** – a description of the current behavior (after the implementation).

It is not advisable to go too deep into implementation details, however, a reader should be able to understand what has been fixed and where.

For example:

```
The exception in method BRepFill_Cone::Truncate caused by hard-coded number of symbols in the parameter for
       truncation operation has been fixed.

Now the parameter provided by method BRepFill_Cone::Parameter is stored in a variable, which is used for
      truncation.
```

Here, in this example:

- the **Problem** is "hard-coded number of symbols in the parameter for truncation operation"
- the **Change** is "The exception in method BRepFill_Cone::Truncate has been fixed"
- the **Result** is "Now the parameter provided by method BRepFill_Cone::Parameter is stored in a variable, which is used for truncation".

Other cases may require an even more detailed description, or vice versa, some element can be not worth indicating.

Exposing the Problem

The **Problem** can be omitted if it is stated correctly and in full in the **Summary** of the issue.

For example:

```
Summary: Error in IntPatch_PrmPrmIntersection: change of local resolution leads to break of walking line.
```

completely exposes the **Problem** and the description

```
The method IntWalk_Pwalking::TestDeflection has been modified to take into account the local resolution of
      the chosen surface during the computation of the local step and the maximum step.
```

consists only of:

- the **Change**, which is "The method IntWalk_Pwalking::TestDeflection has been modified", and

(c) Open CASCADE 2015

- the **Result** "to take into account the local resolution of the chosen surface", etc.

However, the Summary may need additional clarification, for example:

```
Summary:  Offset on faces with opposite orientation.


The offset algorithm has been improved in method BRepOffset_Offset::Init to properly calculate normals if
      mirror transformations are associated with input objects.
```

Here, "if mirror transformations are associated with input objects" describes the **Problem**.

Structurally, the **Problem** can be introduced, for example:

- By a separate sentence, probably starting with "previously", "initially", "in earlier OCCT versions", etc. For example:

```
Previously the method XCAFDoc_ShapeTool::UpdateAssembly() rebuilt the shape of assembly, however, it did
      not follow the back-references, i.e. the users of the assembly. Now this method checks back-references in the
      bottom-up direction to ensure the shape data consistency in an XCAF document
```

- By an "if" or "when" clause defining the prerequisites of the erroneous behavior:

```
The function ComputePurgedWLine() now can delete excess points in the walking line if the distance between
      points is too small or they lie in one pipe without big jump on chord length.
```

- By a complement to such verbs as "avoid" or "eliminate" :

```
Equality checks have been added to camera modification methods in Graphic3d_Camera class to avoid camera
      updates when performing identity operations.
```

- By a phrase "the problem/issue with"

```
The problem with a failure to resize objects when the view area is maximized has been fixed in Qt samples.
```

It can be noted that stating only the **Problem** and the **Change** can be sufficient in case of regressions, when the behavior of algorithms is restored rather than modified.

Exposing the Change

The **Change** is basically a mention of the introduced or amended function/method/class.

The **Change** can be omitted in the message if only one method has been amended, and that can be seen in GIT.

- In complex fixes or improvements when there are several methods/classes concerned by the fix, and correspondingly several different statements, it is necessary to indicate which corresponds to which:

```
Protection against 0 dimensions has been added in OpenGl_FrameBuffer::Init().

New method V3d_View::IsInvalidated() checks view cache validation state.

ViewerTest::ViewerInit() now creates a virtual window without decorations on Windows.
```

- If numerous similar fixes are applied in one (or several) classes, packages or toolkits, it is possible to generalize:

```
BRepBuilderAPI package has been revised to avoid declaring methods operator TopoDS_Shape and Shape as const
      , which can cause a compilation error.
```

- Same for more global revisions:

(c) Open CASCADE 2015

```
OCCT code has been refactored to remove redundant const qualifiers of return types of functions returning
      values.
```

Presenting only the **Change** is enough when we speak about removal of obsolete entities, however it is advisable to mention which entities can be used instead.

```
Package SortTools and its derived classes have been removed and replaced throughout OCCT by STL sorting
      algorithms (i.e. std::sort).
```

Or:

```
Class BOPCol_Array1 has been removed.

Instead, new method Ncollection_BaseVector::SetIncrement allows setting the size of increment dynamically.

Classes from package BOPDS have been modified accordingly.
```

Exposing the Result

The result usually is the most interesting information for the end-user, so this is really preferable to concentrate on this part of the message.

I.e. instead of focusing on the **Problem**

```
The use of reference to a destroyed temporary object has been fixed in method
      Adaptor3d_SurfaceOfRevolution::GetType().
```

it is better to focus on the result

```
The method Adaptor3d_SurfaceOfRevolution::GetType() now makes a copy of temporary object for further use.
```

Here are some *typical* examples of results and appropriate commit messages:

- Implementation of a new method/class:

```
New method CheckFace::Point allows checking if the point is IN the face.
```

- Improvement of an existing algorithm:

```
The concatenation algorithm in class BSpline_Surface now works with periodic BSpline surfaces.
```

- Introduction of rules and limitations:

```
Comparison of handle to NULL has become impossible. Method IsNull() should be used instead.
```

Description of a large-scale development

The implementation of new functionalities or functional overhauls can contain numerous modifications concerning numerous packages or classes in one package.

The description of such implementation should include two elements:

- General description or Summary, which usually defines the **Problem**;
- List of modifications with necessary comments, which include the **Changes** and the **Results**.

According to the requirements of section Issue registration, the purpose of an improvement should be already given in the Issue Description.

However, if the Issue Description is missing or if it describes the problems rather then their solution or if some other important information is missing, it should be given in the commit message.

Here is the example of information about numerous changes in connection with the issue that should be provided:

(c) Open CASCADE 2015

```
B-spline cache has been separated into classes BSplCLib_Cache for 2D and 3D curves and BSplSLib_Cache for
      surfaces. They are now used in the corresponding  adaptor classes Geom2dAdaptor_Curve, GeomAdaptor_Curve and
      GeomAdaptor_Surface when the curve or surface is a B-spline. Consequently:


- Classes BOPAlgo_WireSplitter, BOPTools_AlgoTools, BRepLib_CheckCurveOnSurface and ShapeAnalysis_Wire now

use adaptors for B-spline calculations instead of geometric entities (curves or surfaces);

- Classes from Geom2dAdaptor and GeomAdaptor packages now use the adaptor for the base curve for evaluation
       of offset curves;

- The derivatives of a surface of revolution now can be precisely calculated in class
      Geom_SurfaceOfRevolution for the points of surface placed on the axis of revolution.
```

Some recommendations:

- It is unnecessary to mention modification test-cases. It goes without saying that you do it.
- Try to generalize similar modifications in one entry
- If the modification is meaningless or too small, don't mention it.
- Sometimes more modifications are added while the fix is reviewed via several iterations. Sometimes early modifications are overridden or dropped or on the contrary later modifications do not add anything important. Please, review the list of modifications when you finalize the bug.

2.4.6 Marking issue as resolved

To mark the change as ready for review and testing, the corresponding issue should be switched to **RESOLVED** state. By default, the issue gets assigned to **Maintainer** of the component, who is thus responsible for its review. Alternatively, another person can be selected as reviewer at this step.

When switching issue to **RESOLVED**, it is required to fill the field **Steps to reproduce**. The possible variants are:

- Name of existing or new test case (preferred variant)
- Sequence of DRAW commands
- N/A (Not required / Not possible / Not applicable)
- Reference to issue in bug tracker of another project

2.5 Code review

The **Reviewer** analyzes the proposed solution for applicability in accordance with OCCT Coding Rules and examines all changes in the sources, test case(s), and documentation 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. The issue can then be switched to **REVIEWED**.

  In case of doubt or possible disagreement **Reviewer** can reassign 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 a new solution should be provided.

If Reviewer does not detect any problems, or provides corrected version, he changes status to **REVIEWED**. The issue gets assigned to **Bugmaster**.

(c) Open CASCADE 2015

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.

If branch submitted for testing is based on obsolete status of the master branch, **Tester** rebases it on master HEAD. In case of conflicts, the issue is assigned back to **Developer** in **FEEDBACK** status, requesting for rebase.

Certification testing includes:

- Building OCCT on sub-set of supported configurations (OS and compiler), controlling errors and warnings
- Execution of tests on sub-set of supported platforms (at least, one Windows and one Linux configuration), controlling regressions
- Building OCCT samples, controlling errors
- Building and testing of OCC products based on OCCT

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

- If the **Tester** does not detect build problems or regressions, he changes the status to **TESTED** for further integration.
- 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 problems. The **Developer** should abnalyze the reported problems and, depending on results of this analysis, either:
  - Confirm that the detected problems are expected changes and they should be accepted as new status of the code. The issue should be switched to **FEEDBACK** and assigned to **Bugmaster**.
  - Produce a new solution (see Resolving the issue, and also Minor corrections).

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;
- the change does not conflict with other changes integrated previously.

If the result of check is successful the Integrator integrates solution into the integration branch. Integrations are performed weekly; integration branches are named following pattern IR-YYYY-MM-DD.

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

(c) Open CASCADE 2015

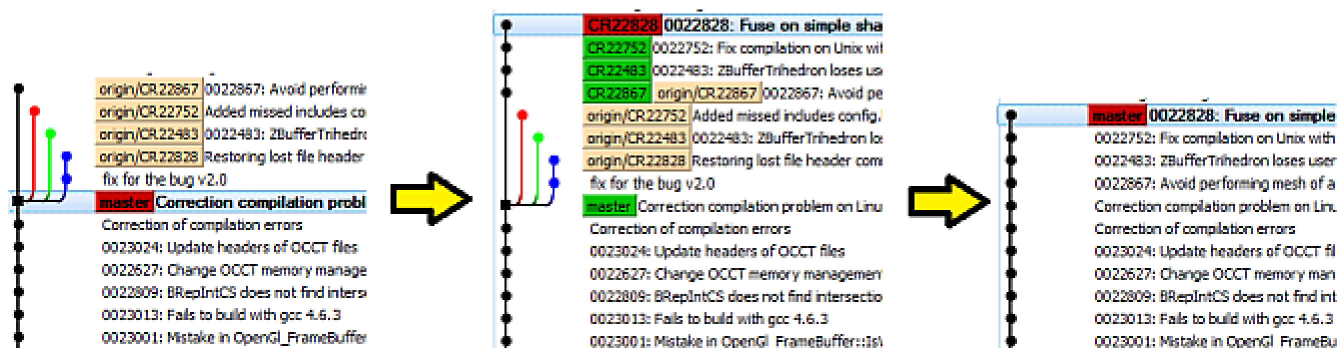

Figure 2: Integration of several branches

The new integration branch is tested against possible regressions that might appear due to interference between separate changes. When the tests are Ok, the integration branch is pushed as new master to the official repository. The issue status is set then to **VERIFIED** and is assigned to the **Reporter** so that he could check the fix as integrated.

The branches corresponding to the integrated fixes are removed from the repository by **Bugmaster**.

2.8 Closing a bug

The **Bugmaster** closes the issue after regular OCCT Release, provided that the issue status is **VERIFIED** and the change was included in the release. The final issue state is **CLOSED**.

The field **Fixed in Version** of the issue is set to the version of OCCT where it is fixed.

(c) Open CASCADE 2015

3 Additional workflow elements

3.1 Requesting more information or specific action

If, at any step of the issue lifetime, the person responsible for it can not process it due to absence of required information, expertise, or rights, he can switch it to status **FEEDBACK** and assign to the person who is (presumably) able to resolve the block. Some examples of typical situations where **FEEDBACK** is used are:

- **Maintainer** or **Developer** requests for more information from **Reporter** to reproduce the issue
- **Tester** requests **Developer** or **Maintainer** for help in interpretation of results of testing
- **Developer** or **Maintainer** asks **Bugmaster** to close the issue that is found irrelevant or already fixed

In general, issues having status **FEEDBACK** should be processed as fast as possible, to avoid unjustified blocking of the issue processing.

3.2 Defining relationships between issues

When two or more issues are related to each other, this relationship should be reflected in the issue tracker. It is also highly recommended to add a note to explain the relationship. Typical cases of relationships are:

- Issue A is caused by previous fix made for issue B (A is a child of B)
- Issue A describes the same problem as issue B (A is a duplicate of B)
- Issues A and B relate to the same piece of code, functionality etc., in the sense that fix for one of these issues will affect the other (A is related to B)

When fix made for one issue resolves also another one, these issues should be marked as related or duplicate. In general, the duplicate issue should have the same status, and, when closed, be marked as fixed in the same OCCT version, as the main one.

3.3 Submission of a change as a patch

In some cases (if Git is not accessible for the contributor), external contributions can be submitted as patch file (generated by diff command) or modified sources attached to the Mantis issue. OCCT version for which the patch is generated should be clearly specified (e.g. as hash code of Git commit if patch is based on intermediate state of the master).

Note

> Such contributions should be put to Git by someone else (e.g. **Reviewer**), and hence they have less priority in processing than the ones submitted directly through Git.

3.4 Updating branches in Git

Updates of existing branch (e.g. to take into account remarks of the **Reviewer**, or fixing regressions) should be provided as new commits on top of previous state of the branch.

It is allowed to rebase the branch on the new state of the master and push it to the repository under the same name (with –force option) provided that original sequence of commits is preserved.

When a change is squashed into single commit (e.g. for submission for review), it should be pushed into branch with different name. The recommended approach is to add numeric suffix (index) indicating version of the change, e.g. "CR12345_5". Usually it is worth keeping non-squashed branch in Git for reference.

To avoid confusions, branch corresponding to the latest version of the change should have greater index.

(c) Open CASCADE 2015

Note

Make sure to revise commit message after squashing a branch, to keep it meaningful and comprehensive.

3.5 Minor corrections

In some cases review remarks or results of testing require only minor corrections to be done in the branch containing a change. "Minor" implies that correction does not impact functionality and does not affect description of the previously committed change.

As an exception to general single-commit rule, it is allowed to put such minor corrections on top of existing branch as separate commit, and re-submit it for further processing in the same branch, without squashing.

Minor commits should have single-line message starting with //. These messages will be ignored when the branch is squashed at integration.

Typical cases of minor corrections are:

- Amendments of test cases (including those made by **Tester** to adjust test script to specific platform)
- Trivial corrections of compilation warnings (such as removal of unused variable)
- Addition or correction of comments or documentation
- Corrections of code formatting (e.g. revertion of irrelevant formatting changes made in main commit)

(c) Open CASCADE 2015

4 Appendix: Issue attributes

4.1 Category

The category corresponds to the component of OCCT where the issue is found:

Category	Component
OCCT:Foundation Classes	Foundation Classes module
OCCT:Modeling Data	Modeling Data classes
OCCT:Modeling Algorithms	Modeling Algorithms, except shape healing and meshing
OCCT:Shape Healing	Shape Healing component (TKShapeHealing)
OCCT:Mesh	BRepMesh algorithm
OCCT:Data Exchange	Data Exchange module
OCCT:Visualization	Visualization module
OCCT:Application Framework	OCAF
OCCT:DRAW	DRAW Test Harness
OCCT:Tests	Automatic Test System
OCCT:Documentstion	Documentation
OCCT:Coding	General code quality
OCCT:Configuration	Configuration, build system, etc.
OCCT:Releases	Official OCCT releases
Website:Tracker	OCCT Mantis issue tracker, tracker.dev.opencascade.org
Website:Portal	OCCT development portal, dev.opencascade.org
Website:Git	OCCT Git repository, git.dev.opencascade.org

4.2 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
crash	Crash of the application or OS, loss of data
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.
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.
minor	Incorrect behavior of a function corresponding to the description in software requirements specification. Insufficient performance of a function.
tweak	Ergonomic inconvenience, need of light updates.
text	Non-conformance of program code to the Coding Rules, mistakes and non-functional errors in source text (e.g. unnecessary variable declarations, missing comments, grammatical errors in user manuals).

(c) Open CASCADE 2015

trivial	Cosmetic issues.
feature	Request for a new feature or improvement.
integration request	Requested integration of an existing feature into the product.
Just a question	A question to be processed, without need of any changes in the product.

4.3 Status

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

Status	Description
New	New just registered issue.
Acknowledged	Can be used to mark issue as postponed.
Confirmed	Can be used to mark issue as postponed.
Feedback	The issue requires more information or specific action.
Assigned	Assigned to a developer.
Resolved	The issue has been fixed, and now is waiting for review.
Reviewed	The issue has been reviewed, and now is waiting for testing (or being tested).
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 master of the corresponding repository
Closed + resolution	The fix has been integrated to the master. The corresponding test case has been executed successfully. The issue is no longer reproduced.

4.4 Resolution

Resolution is set when the bug is closed. "Reopen" resolution is added automatically when the bug is reopened.

Resolution	Description
Open	The issue is pending.
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 e.g. it is a bug of third party software, OS or hardware limitation, etc.
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	The issue is postponed (for Acknowledged status).
Documentation updated	The documentation has been updated to resolve misunderstanding causing the issue.
Won't fix	It is decided to keep existing behavior.

(c) Open CASCADE 2015