Getting Started with Open CASCADE Technology 7.0

with some notes on porting to the new release

Generated by Doxygen 1.8.9.1

Contents

Chapter 1

Preface

This document describes how to get started with OCCT 7.0. It is intended for developers who are already familiar with Open CASCADE Technology and want to port their projects from older versions of the library. After reading this document you should get a clear vision on the amount of efforts required for such porting. The document begins with an overview of CMake-based build process which is now suggested as a standard way to produce the binaries of Open CASCADE Technology from sources. After that we describe the principal changes in the foundation classes of OCCT which require particular attention as these changes affect the entire ecosystem of the library. The structure of the document is given below:

- Building Open CASCADE Technology 7.0 with CMake. From this chapter you will learn how to build OCCT 7.0 from bare sources in a modern and platform-independent way without WOK;
- OCCT smart pointers and RTTI. This chapter describes the new mechanism of smart pointers and RTTI;
- Porting notes. This final chapter gives you concrete recipes on porting your OCCT-based software from versions 6.x to the recent version 7.0.

Chapter 2

Building with CMake

CMake build is a novel platform-independent way to build Open CASCADE Technology from sources. OCC↩ T requires CMake version 3.0 or later.

Note

We use CMake version 3.1.0 in this documentation.

Comparing to the previous (6.x) releases of Open CASCADE Technology, the recent OCCT 7.0 comes with a complete set of CMake scripts and projects, so that there is no need to use WOK anymore. Moreover, CMake gives you a powerful configuration tool which allows to control many aspects of OCCT deployment. At the same time this tool is quite intuitive which is a significant advantage over the legacy WOK utilities.

2.1 Building from scratch

Here we describe the steps you have to take in order to buid OCCT library from sources.

Note

We discuss the build procedure on example of Windows platform. However, the workflow is almost the same for ∗nix and Mac operating systems.

2.1.1 Get sources

You have to get sources of OCCT from the official development web-site [1] by either downloading the universal source package (available at official download page [2]) or by cloning the Git repository:

```
git clone ssh://gitolite@git.dev.opencascade.org/occt occt
```

As a result, you obtain the following directory structure in your filesystem (`workbench` name is used for example):

```
workbench\occt\adm
              \data
              \dox
              \samples
              \src
              \tests
              ...
```

The bare sources distribution contains not only the sources of Open CASCADE Technology, but also documentation, samples and non-regression test scripts. We refer you to the official manual [3] for details on the distribution contents.

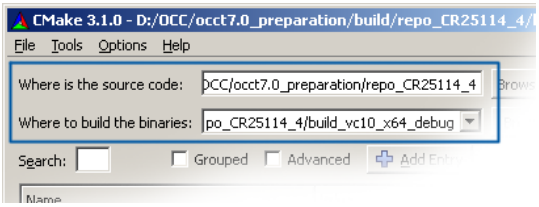
2.1.2 Configuring process

Now it is time to run CMake GUI which will generate the actual project files for the target IDE. The first step is to specify where the sources of OCCT are located in your filesystem and what is the working directory for build. The good practice is not to mix up different build configurations in a single directory and not to use the source directory as a build one. Let us suppose that the target platform is Windows x64 and the target IDE is Visual Studio 2010. Normally we want to have two builds of OCCT: one is for optimized mode (release) and another for maintenance (debug). Therefore, we prepare two empty directories:

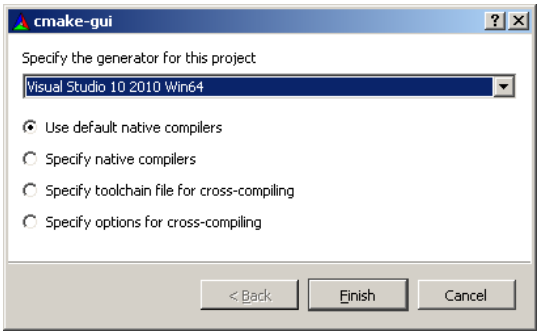
```
workbench\build\build_vc10_x64_debug
              \build_vc10_x64_release
```

Note

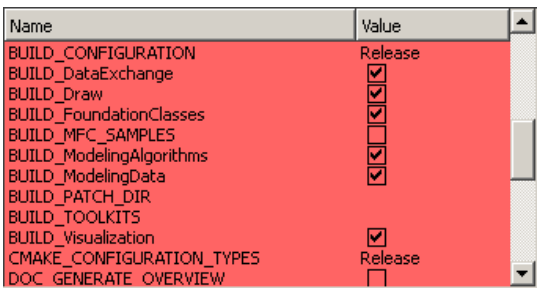
We continue description for debug configuration only. The process is nearly the same for the release mode (just different build directory `build_vc10_x64_release` will be used with slightly different CMake options: the actual differences should become clear after reading this chapter).



Once the source and build directories are selected, "Configure" button should be pressed in order to start manual configuration process. It begins with selection of a target configurator. It is "Visual Studio 10 2010 Win64" in our exercise.



Once "Finish" button is pressed, CMake outputs the list of environment variables which have to be properly initialized for successful compilation. These variables define the paths to the 3-rd party products (most of them are optional) and, generally, the contents of the resulting build (e.g. whether to skip or not some particular module of OCCT). After the first configuration attempt these variables need to be reviewed, which is clearly identified by CMake GUI with red notification color.



Note

You can get all the required 3-rd party products visiting the official download page [4]. Basically, you should choose the newest version of every product unless you have a good reason to use some older binaries.

The following table enumerates the full list of environment variables used at configuration stage:

Variable	Type	Purpose
`USE_FREEIMAGE`	Boolean flag	Indicates whether Freeimage product should be used in OCCT visualization module for support of popular graphics image formats (PNG, BMP etc)
`USE_GL2PS`	Boolean flag	Indicates whether GL2PS product should be used in OCCT visualization module for support of vector image formats (PS, EPS etc)
`USE_TBB`	Boolean flag	Indicates whether TBB 3-rd party is used or not. TBB stands for Threading Building Blocks, the technology of Intel Corp, which comes with different mechanisms and patterns for injecting parallelism into your application. It is worth saying that OCCT remains parallel even without TBB product
`USE_VTK`	Boolean flag	Indicates whether VTK 3-rd party is used or not. VTK stands for Visualization ToolKit, the technology of Kitware Inc intended for general-purpose scientific visualization. OCCT comes with a bridge between CAD data representation and VTK by means of its dedicated VIS component (VTK Integration Services). You may easily skip this 3-rd party unless you are planning to use VTK visualization for OCCT geometry. We refer you to the official documentation [3] ("VTK Integration Services" user's guide) for the details on VIS

3RDPARTY_DIR	Path	Defines the root directory where all required 3-rd party products will be searched. Once you define this path it is very convenient to click "Configure" button in order to let CMake automatically detect all necessary products. This "intelligence" significantly reduces the amount of time you spend on the configuration process
3RDPARTY_FREETYPE_*	Path	Path to Freetype binaries
3RDPARTY_TCL_* 3RDPARTY_TK_*	Path	Path to Tcl/Tk binaries
3RDPARTY_FREEIMAGE*	Path	Path to Freeimage binaries
3RDPARTY_GL2PS_*	Path	Path to GL2PS binaries
3RDPARTY_TBB*	Path	Path to TBB binaries
3RDPARTY_VTK_*	Path	Path to VTK binaries
BUILD_<MODULE>	Boolean flag	Indicates whether the corresponding OCCT module should be built or not. It should be noted that some toolkits of a module can be built even if this module is not checked (this happens if some other modules depend on these toolkits). The main modules and their descriptions can be found in [3] ("User Guides" chapter) and partially in [5]
BUILD_TOOLKITS	String	Enumerates individual toolkits to include into build process. If you want to build some particular libraries (toolkits) only, then you may uncheck all modules in the corresponding BUILD_<MODULE> options and provide the list of necessary libraries here. Of course, all dependencies will be resolved automatically

BUILD_CONFIGURATION	Release/Debug	Release is default
BUILD_BISON_FLEX_FILES	Boolean flag	Enables Flex/Bison lexical analyzers. A couple of OCCT source files are generated automatically with Flex/Bison (STEP reader and ExprIntrp functionality). Checking this options leads to automatic search of Flex/Bison binaries and regeneration of the mentioned files
BUILD_MFC_SAMPLES	Boolean flag	Indicates whether MFC samples should be built together with OCCT. Obviously, this option is only relevant to Windows platforms
BUILD_PATCH_DIR	Path	Points to the directory recognized as a "patch" for OCCT. If specified, the files from this directory take precedence over the corresponding native OCCT sources. This way you are able to introduce patches to Open CASCADE Technology not affecting the original source distribution
CMAKE_CONFIGURATION_T↩ YPES	String	Semicolon-separated CMake configurations
INSTALL_DIR	Path	Points to the installation directory where all OCCT binaries and include files will be deployed
INSTALL_FREETYPE	Boolean flag	Indicates whether Freetype binaries should be installed into the same directory as OCCT
INSTALL_FREEIMAGE*	Boolean flag	Indicates whether Freeimage binaries should be installed into the same directory as OCCT
INSTALL_GL2PS	Boolean flag	Indicates whether GL2PS binaries should be installed into the same directory as OCCT
INSTALL_TBB	Boolean flag	Indicates whether TBB binaries should be installed into the same directory as OCCT

INSTALL_VTK	Boolean flag	Indicates whether VTK binaries should be installed into the same directory as OCCT
INSTALL_TCL	Boolean flag	Indicates whether Tcl/Tk binaries should be installed into the same directory as OCCT
INSTALL_GROUP_INC	Boolean flag	Indicates whether the OCCT header files should be distributed by subdirectories corresponding to their native packages
TESTS_INSTALL	Boolean flag	Indicates whether to copy non-regression test scripts to the installation directory
TESTS_RUN	Boolean flag	Indicates whether to prepare a specific project for running non-regression tests from IDE
TESTS_SHAPES_DIR	Path	Points to the directory with reference test files

Note

Do not ever use back slashes ("\") in those CMake options defining paths. Only the forward slashes ("/") are acceptable.

One mandatory option to specify is the installation directory. This option stores a path where OCCT will be deployed after build. We will create a single destination directory `install` as both the target platform and configuration (debug/release) will be taken into account automatically in form of subdirectories (explained later).

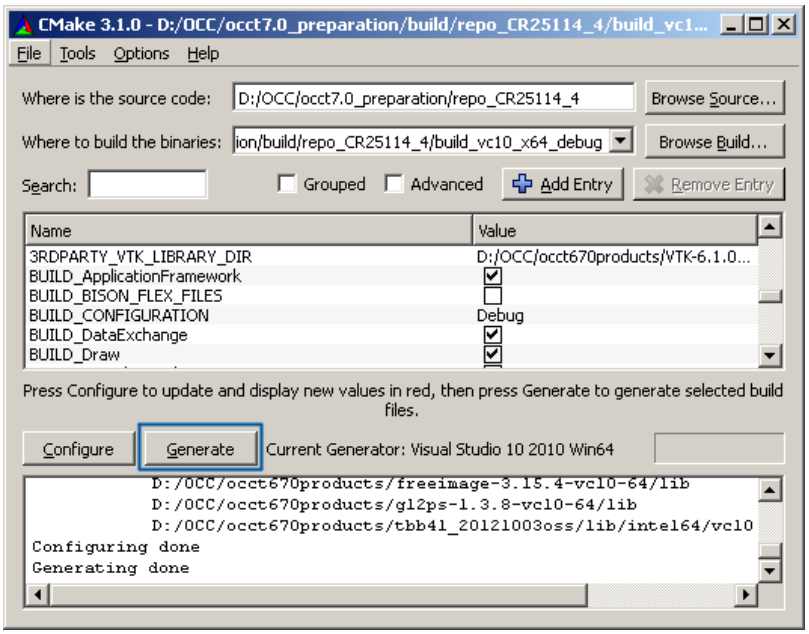
```
workbench\install
```

If all options are defined consistently, then pressing "Configure" button will lead to creation of CMake configuration files in the target build directory.

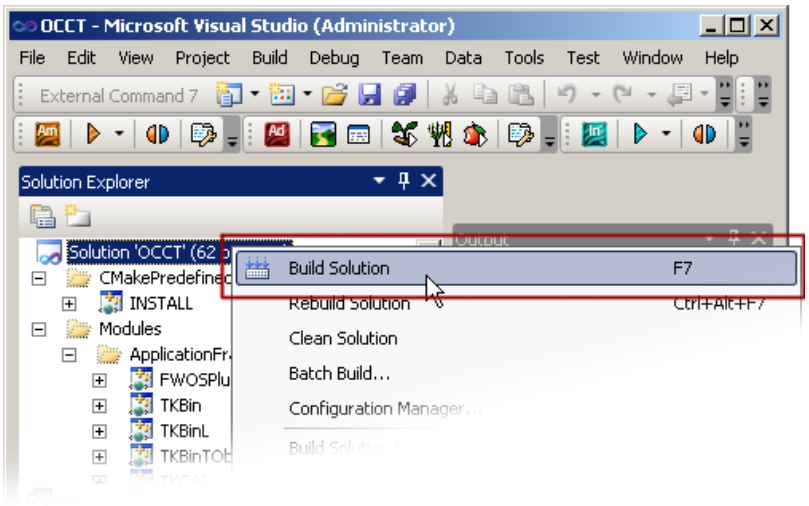
Todo It is not possible to run Draw from build directory: draw.bat is not there

2.1.3 Actual build

Once the configuration process is done, "Generate" button is used to prepare project files for the target IDE. In our exercise the Visual Studio solution will be automatically created.



Then it is a simple question of running your target IDE and building everything as usual.



2.1.4 Installation process

Installation is a process of extracting redistributable resources (binaries, include files etc) from the working build directory into a target place. The target directory will be free of project files, intermediate object files and any other information related to the build routines. Normally you use the installation directory of OCCT to link against your specific application. In order to perform installation, you are supposed to build the dedicated `INSTALL` project which simply copies all necessary resources to the desired destination directory (which you have pre-configured with `INSTALL_DIR` CMake variable). The installation directory will be automatically expanded as follows:

```
workbench\install\data
                 \inc
                 \samples
                 \src
                 \win64\vc10\bind
                           \libd
```

Note

The actual contents of the `install` directory depend on the pre-configured CMake options. E.g. the subdirectory for non-regression tests (with name `tests`) will appear or not depending on the value of `TESTS_INSTALL` option.

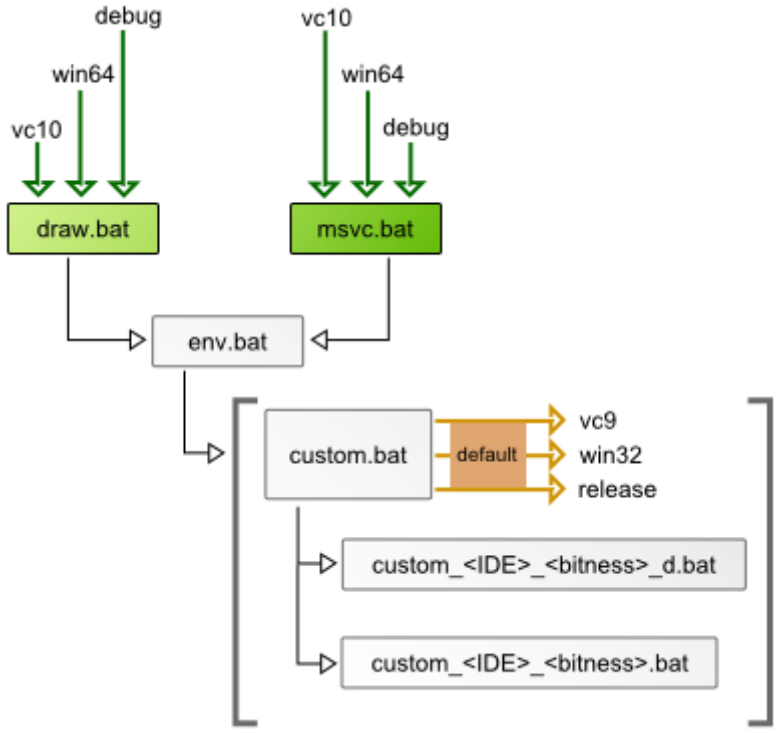
The above example is given for debug configuration. However, it is generally safe to use the same `install` directory for the release build. In the latter case the contents of `install` directory will be enriched with subdirectories and files related to the release configuration. In particular, the binaries directory `win64` will be expanded as follows:

```
\win64\vc10\bind
          \libd
          \bin
          \lib
```

Here the suffix *d* denotes *debug* configuration, so that it is easy to guess where to find the debug version of OCCT binaries. Per contra, the release binaries are available in `bin` and `lib` directories. If CMake installation flags are enabled for the 3-rd party products (e.g. `INSTALL_FREETYPE`), then the corresponding binaries will be copied to the same `bin(d)` and `lib(d)` directories together with the native binaries of OCCT. Such organization of libraries can be especially helpful if your OCCT-based software does not use itself the 3-rd parties of Open CASCADE Technology (thus, there is no sense to pack them into dedicated directories).

2.1.4.1 Notes on environment

OCCT is installed together with several scripts setting up its environment. The picture below illustrates the principal interrellation between these scripts.



- Two entry points `draw.bat` and `msvc.bat` pass the desired configuration as command line arguments:
  - IDE version: `${VCVER}`;
  - Platform (bitness): `${ARCH}`;
  - Debug/release mode: `${CASDEB}`;

- The core script setting up all OCCT environment variables is `env.bat` which in turn calls the optional `custom.bat` script;
- If present, `custom.bat` script calls two specialized utilities `custom_<IDE>_<bitness>.bat` and `custom_<IDE>_<bitness>_d.bat` setting up the paths to the 3-rd party libraries;
- `custom.bat` is a machine-dependent file which you may alter in order to specify the default build configuration: variables `${VCVER_DEFAULT}`, `${ARCH_DEFAULT}` and `${CASDEB_DEFAULT}` have to be modified.

Todo it is necessary to improve environment mechanism so that to allow setting up default arguments in custom.bat rather than in env.bat. Moreover, environment should be non-conflicting. Request for improvement in form of two versions of environment set: old (CMake-generated) and new (manually improved)

Todo Release-mode installation generates file with name `custom_vc10_64_.bat`. What about cutting off the trailing "_" character?

2.2 More features of CMake

The procedure described above gives you a general direction on how to build Open CASCADE Technology 7.0 from scratch. Playing with CMake configuration variables you may affect the contents of the distribution very intuitively (each CMake variable is supplied with a small hint arising in CMake GUI on mouse hover). However, some specific options of CMake still worth additional explanation.

Todo Some hints start from upper case, while some of them start from the lower case. It is better to unify the formatting here. Moreover, some tooltips are just non-informative and look weird. E.g. what does "Toolkits are also included in OCCT" message actually mean (the one for `BUILD_TOOLKITS` option)? It seems like the tooltips should be carefully reviewed. It is necessary to reuse hints from the table given above. Doing so, one can also check that all the options are really described in the table. If not, this document should be extended.

2.2.1 Non-regression tests

Note

Consider skipping this paragraph if you are not planning to modify OCCT sources.

Open CASCADE Technology comes with a non-regression test suite organized as a bunch of Tcl scripts designed for automatic launch in Draw interpretor (Draw Test Harness plays a role of a test engine for OCCT). Whenever you introduce some change to OCCT sources (bug fix, improvement or new feature), the non-regression tests have to be executed in order to check whether this change has negative (or "false negative") impact on the library.

There are three CMake options configuring the non-regression test suite:

- `TESTS_INSTALL`
- `TESTS_RUN`
- `TESTS_SHAPES_DIR`

The first `TESTS_INSTALL` option is used to copy the actual grid of test cases to the installation directory. As a result, additional `tests` directory will appear in your distribution of OCCT. The directory containing the reference test files should be specified in `TESTS_SHAPES_DIR` variable.

There are two ways to run the non-regression tests. The standard way is to use the `testgrid` command in the Draw interpretor:

```
Draw> testgrid
```

However, this approach requires some manual work within the Draw interpretor. Another possibility consists in enabling TESTS_RUN CMake option which results in creation of a specific START_TESTS project. Building this project launches the non-regression tests automatically, so there is no need to cooperate with a console anymore.

Todo The TESTS_RUN name is really weird. One would never correctly guess what does this variable mean because of its name and the associated tooltip (it looks like enabling this option should run the test suite automatically after build, while this is not true). Can it be renamed to something more meaningful, e.g. TESTS_CREATE_PROJECT?

Todo Once the START_TESTS project is built (to run non-regression tests right from the IDE), it uses its own output directory TestResults which is different from the directory results utilized by Draw (when "testgrid" command is executed manually). Why not to simply reuse results directory with the same approach of generating temporary subdirectories inside? Moreover, the CamelCase format of the directory name (TestResults) ruins the naming feng-shui accepted in the installation dir (lowercase style).

2.2.2 Preparing patch for OCCT

Note

Consider skipping this paragraph if you are not planning to modify OCCT sources.

Now let us suppose that we have a patch for OCCT sources which is necessary to apply in the following conditions:

- The patch should be completely separated from the original sources;
- We do not want to override includes and binaries by agreement on their order in the list of compilation parameters (i.e. patch includes and binaries preceed the native includes and binaries). It is less error-prone to simply inject the patched sources into the project files right at the build stage.

Fortunately, OCCT facilitates this way of patching by means of a dedicated BUILD_PATCH_DIR CMake option.

Todo BUILD_PATCH_DIR has no effect. Patching functionality does not work

2.2.3 Building individual toolkits

Todo What kind of format is expected here? I was not able to build individual toolkits at all. Just reset all modules and type "TKFillet" to reproduce

2.2.4 Organization of headers

You can choose between two styles of organization of include files in the installation directory. If the INSTALL_↩ GROUP_INC option is enabled (default behavior), then all header files are grouped into internal subdirectories with names of the corresponding packages. If not, all include files will be simply copied to inc directory without any hierarchy.

2.2.5 Samples (MFC)

Building of OCCT sample applications can be included into the building process of OCCT itself by means of a dedicated BUILD_MFC_SAMPLES option.

Todo build with samples does not work: compilation errors

Todo remove all CMake DOC* options as we do not want to manage documentation at configuration stage

Todo It is necessary to check that all modules can be built individually. E.g. I have got linkage errors trying to build Modeling Algorithms only.

Chapter 3

Handles and RTTI

Todo description of new Handle mechanism with RTTI is coming

Todo Official training materials have to be updated: get rid of WOK, new Handles etc

Chapter 4

Porting notes

4.1 Overview

In many aspects upgrade to OCCT 7.0 is a trivial process as most of interfaces are kept unchanged or modified only slightly. Even though the internal implementation of some basic mechanisms is completely reworked, the general compatibility of OCCT 7.0 with the previous versions is maintained on a rather high level. Those changes requiring your close attention are enumerated in the release notes document [9] and are not discussed here. In this chapter we only focus on those porting issues which can (in fact, require to) be automated. OCCT 7.0 comes with auxiliary `adm/upgrade.tcl` tool facilitating such upgrading process for your projects. You can find the technical description of this tool and its capabilities in [8].

As an ordinary Tcl script, this tool can be executed from any Tcl shell. The most natural way for us is using Draw Test Harness shipped with OCCT 7.0:

```
Draw> source %CASROOT%/adm/upgrade.tcl
```

Having this script loaded, you may execute different upgrade routines using the following syntax:

```
Draw> occt_upgrade ${YOUR_PROJECT_DIR} [upgrade_options] [other_options]
```

Note

The root directory of your project `${YOUR_PROJECT_DIR}` must contain `src` subdirectory with all sources.

We logically separate the available list of options on two groups: `upgrade_options` and `other_options`. The first `upgrade_options` group is a set of keys which specify which conversions exactly you want to perform: normally you do not have to apply all the available upgrade routines. The group of parameters denoted as `other_options` is a set of additional keys which allow to configure different aspects of conversion. We discuss both groups in detail below.

4.1.1 Upgrade options

The following upgrade modes are available:

Option	Purpose

`-rtti`	Applies the upgrade routines related to RTTI mechanism
`-handle`	Replaces all explicit names of smart pointer classes (e.g. `Handle_Geom_Curve`) with macro `Handle()` (e.g. `Handle(Geom_Curve)`) and removes all forward declarations like `"class Handle(...)"`
`-inc`	Allows to easily switch the include style from `"occtClass.hxx"` to `"occtPackage/occtClass.hxx"` in your project. This option is useful if you took advantage of `INSTALL_GROUP_INC` flag during the build process of OCCT
`-tcollection`	Replaces CDL instantiations of `TCollection` generic classes by instantiation of equivalent `NCollection` template (useful for WOK-based projects)
`-mutable`	Allows to get rid of `mutable` keyword in CDL files (useful for WOK-based projects)
`-all`	Performs all available upgrade routines except `-inc` as the latter one should not be applied in case if `INSTALL_GROUP_INC` flag was turned off

4.1.2 Other options

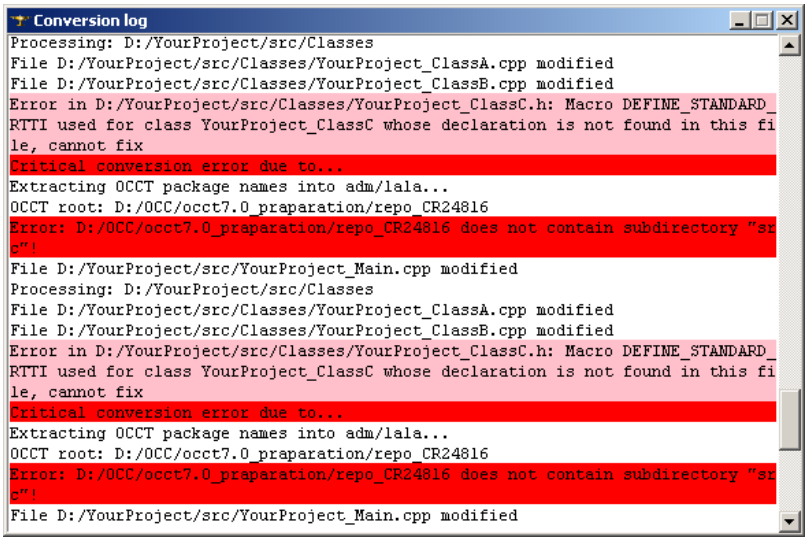
Other available options are enumerated in the following table:

Option	Purpose
`-compat`	Allows to reduce incompatibilities with older versions of OCCT when upgrading to OCCT 7.0. E.g. some obsolete macro will be kept with empty substitution
`-check`	Runs upgrade routine in a read-only mode. Thus you can check the status of your project before and after the real conversion is applied
`-rtti_nocdl`	Option which allows to get rid of CDL files during RTTI upgrade (useful for WOK-based projects)
`-ui`	Creates simple Tk-based logger which allows to track the conversion process conveniently
`-nlf_dos`	Allows to switch to DOS line endings preferred style (CRLF). The default is the unix-like style (LF)

`-info=<filename>`	Specifies a path to additional settings file. In particular, this file contains the list of OCCT packages used to resolve includes in `-inc` mode

4.1.3 Example

We recommend to use `-ui` option in order to have a convenient log window receiving all conversion messages. Otherwise the standard output is used for logging.



The following scenario runs RTTI conversion in compatibility mode, so that the changes in your sources are minimal:

```
Draw> source adm/upgrade.tcl
Draw> occt_upgrade D:/MyProject -rtti -compat -ui
```

In order to convert includes in your application, you can use the following scenario:

```
Draw> source adm/upgrade.tcl
Draw> occt_upgrade D:/MyProject -inc -ui -info=adm/upgrade_info.txt
```

It should be noted that `upgrade_info.txt` can be generated by the same `upgrade.tcl` script in case of need:

```
Draw> source adm/upgrade.tcl
Draw> extract_package_names ${CASROOT} ${CASROOT}/adm/upgrade_info.txt
```

Generated in such a way, this `upgrade_info.txt` file will contain all package names extracted for the given OCCT installation.

Note

Normally you do not have to regenerate this `upgrade_info.txt` file as it comes with OCCT distribution.

4.2 Upgrade issues

4.2.1 Includes style

As it was mentioned in building with CMake chapter, using `INSTALL_GROUP_INC` variable you may organize OCCT header files into subdirectories corresponding to package names. If you do so, you will obviously need to upgrade the include directives of your project. For example,

```
#include <Standard_Type.hxx>
```

has to be changed with

```
#include <Standard/Standard_Type.hxx>
```

Use `-inc` option to run this type of conversion.

4.2.2 RTTI macro

Note

> Consider using the `-handle` option together with `-rtti` (you can find the brief overview of `-handle` option in the summary table above).

To remind, OCCT comes with its own RTTI mechanism which allows to maintain the complete type hierarchy of those classes inheriting `Standard_Transient`. In previous versions of Open CASCADE Technology two macro were used in order to inject RTTI functionality into class definition:

- `DEFINE_STANDARD_RTTI(...)` placed in the header file (class declaration);
- `IMPLEMENT_STANDARD_RTTIEXT(...)` placed in the source file.

Starting from OCCT 7.0, the second `IMPLEMENT_STANDARD_RTTIEXT` macro becomes obsolete and can be kept only for compatibility with older versions of the library. The `DEFINE_STANDARD_RTTI` macro has changed to accept the base class name as a second argument.

In order to automatically upgrade your project for using the new RTTI macro, you may take advantage of `-rtti` option:

```
Draw> occt_upgrade ${YOUR_PROJECT_DIR} -rtti [-compat]
```

As a result, the following changes are introduced:

- The second argument is added to `DEFINE_STANDARD_RTTI` macro;
- Includes of `Standard_DefineHandle.hxx` are changed by `Standard_Type.hxx`;
- For all classes used as arguments in `STANDARD_TYPE()` macro the corresponding includes are added;
- If the compatibility mode (`-compat` option) is not enabled, all `IMPLEMENT_DOWNCAST` and `IMPLEMENT_STANDARD_*` macro are removed. Use `-compat` option if you want to minimize the changes of your sources yet keeping an easy way to downgrade to the older versions of OCCT.

4.2.2.1 Note on multiple inheritance

It should be noted that in case of multiple inheritance the upgrade routine will take the last enumerated parent as a second argument for the `DEFINE_STANDARD_RTTI` macro. Consider the following sample:

```
class MyClass : public Standard_Transient,
                       MyInterface
{
  ...
};
```

Of course, in this sample we presume that `MyInterface` class does not extend `Standard_Transient` as we need reference counter to be defined once. Therefore, such multiple inheritance is legal. However, the result of `upgrade.tcl` with `-rtti` flag will give the following macro which is obviuosly incorrect:

```
/* Standard_Transient should be passed as a second argument */
DEFINE_STANDARD_RTTI(MyClass, MyInterface)
```

You have to handle this kind of issues manually (i.e. exchange `MyInterface` argument with `Standard_Transient` or its particular descendant used in your inheritance list).

4.2.2.2 Better control over RTTI in your code

As long as the upgrade routine runs, some information messages are sent to the standard output. In some cases the errors like the following may appear:

```
Error in {HEADER_FILE}: Macro DEFINE_STANDARD_RTTI used for class
{CLASS_NAME} whose declaration is not found in this file, cannot fix
```

These errors help you to diagnose cases where `DEFINE_STANDARD_RTTI` macro is used improperly: either not found or passes invalid class name as an argument. You have to check all such violations manually as these errors indicate hidden problems with RTTI in your original code.

Todo describe improper inheritance: improved because of static_cast

4.2.2.3 One peculiarity of new RTTI

Todo the issue described in this temporary section is to be fixed rather than documented

In order to get access to type information with OCCT RTTI mechanism, you normally use `STANDARD_TYPE(ClassName)` macro which returns `Handle(Standard_Type)` instance playing a role of a type descriptor. OCCT 7.0 comes with a new global registry of type descriptors which ensures single instance of each type descriptor shared by all dynamic libraries. This registry is stored in the static memory, so its populating is done together with library loading. Now consider the following code:

```
class MyRegistrator
{
public:
  MyRegistrator()
  {
    MyFactory::RegisterFunctionForType(STANDARD_TYPE(SomeClass)->Name(),
                                       FunctionPtr);
  }
};
static Registrator TypeRegistrator;
```

The idea of the given code is quite simple: we want to associate some function `FunctionPtr` with the class `SomeClass` using its name as a key. The important point here is that such binding is implemented in a static way by means of the artificial `TypeRegistrator` object constructed in the static memory. This code demonstrates something which works well on the older versions of OCCT, but not on OCCT 7.0. Indeed, the internal `STANDARD_TYPE` invocation does not create type descriptor anymore: it rather attempts to access the descriptor from the static registry. However, there is no guarantee for the type registy to be initialized *before* `TypeRegistrator` object. Therefore, access violation is likely to occur once `Name()` method is invoked.

The purpose of the mentioned sample code was just to bind some information to the type name. Thus the following fix will do the trick:

```
class MyRegistrator
{
public:
```

```
    MyRegistrator()
    {
      MyFactory::RegisterFunctionForType(C::get_type_name(),
                                         FunctionPtr);
    }
  };
  static Registrator TypeRegistrator;
```

Todo test the new version of upgrade.tcl on OCCT sources

Todo `upgrade7.md` document seems to be incomplete at CR24816

Todo Try out `-check` option

Chapter 5

References

1. http://dev.opencascade.org, official development portal, OPENCASCADE SAS
2. http://www.opencascade.org/getocc/download/loadocc, official download page, OPE↩ NCASCADE SAS
3. http://dev.opencascade.org/doc/overview/html/index.html, official documentation, OPENCASCADE SAS
4. http://www.opencascade.org/getocc/download/3rdparty, download page for the 3-rd party products, OPENCASCADE SAS
5. http://isicad.net/articles.php?article_num=17368, "Open CASCADE Technology Overview", OPENCASCADE SAS
6. http://www.stack.nl/~dimitri/doxygen/download.html, Doxygen download page
7. http://www.graphviz.org/Download.php, Graphviz official download page
8. , "Upgrade to OCCT 7.0", official manual for `upgrade.tcl`, OPENCASCADE SAS
9. , "OCCT 7.0 Release Notes", OPENCASCADE SAS

Chapter 6

Todo List

Page Building with CMake

It is not possible to run Draw from build directory: draw.bat is not there

it is necessary to improve environment mechanism so that to allow setting up default arguments in custom.bat rather than in env.bat. Moreover, environment should be non-conflicting. Request for improvement in form of two versions of environment set: old (CMake-generated) and new (manually improved)

Release-mode installation generates file with name `custom_vc10_64_.bat`. What about cutting off the trailing "_" character?

Some hints start from upper case, while some of them start from the lower case. It is better to unify the formatting here. Moreover, some tooltips are just non-informative and look weird. E.g. what does "Toolkits are also included in OCCT" message actually mean (the one for `BUILD_TOOLKITS` option)? It seems like the tooltips should be carefully reviewed. It is necessary to reuse hints from the table given above. Doing so, one can also check that all the options are really described in the table. If not, this document should be extended.

The `TESTS_RUN` name is really weird. One would never correctly guess what does this variable mean because of its name and the associated tooltip (it looks like enabling this option should run the test suite automatically after build, while this is not true). Can it be renamed to something more meaningful, e.g. `TESTS_CREATE_↩PROJECT`?

Once the `START_TESTS` project is built (to run non-regression tests right from the IDE), it uses its own output directory `TestResults` which is different from the directory `results` utilized by Draw (when "testgrid" command is executed manually). Why not to simply reuse `results` directory with the same approach of generating temporary subdirectories inside? Moreover, the CamelCase format of the directory name (`TestResults`) ruins the naming feng-shui accepted in the installation dir (lowercase style).

`BUILD_PATCH_DIR` has no effect. Patching functionality does not work

What kind of format is expected here? I was not able to build individual toolkits at all. Just reset all modules and type "TKFillet" to reproduce

build with samples does not work: compilation errors

remove all CMake `DOC*` options as we do not want to manage documentation at configuration stage

It is necessary to check that all modules can be built individually. E.g. I have got linkage errors trying to build Modeling Algorithms only.

Page Handles and RTTI

description of new Handle mechanism with RTTI is coming

Official training materials have to be updated: get rid of WOK, new Handles etc

Page Porting notes

describe improper inheritance: improved because of static_cast

the issue described in this temporary section is to be fixed rather than documented

test the new version of upgrade.tcl on OCCT sources

`upgrade7.md` document seems to be incomplete at CR24816

Try out `-check` option