Open CASCADE TECHNOLOGY

Training: Preliminaries

Preliminaries: overview

What is Open CASCADE Technology?

Open CASCADE Technology (OCCT) is a powerful open-source C++ library, consisting of thousands of classes and providing solutions in the area of:

- Surface and solid modeling: to model any object.
- 3D and 2D visualization: to display and animate objects.
- Data exchange: to import and export standard CAD formats.
- Rapid application development: to manipulate custom application data.

OCCT is also applicable in many other areas of CAD/CAM/CAE, including AEC, GIS, and PDM. OCCT is designed for the industrial development of 3D modeling and visualization applications that require good quality, reliability, and a robust set of tools.

OCCT libraries are distributed in open source for multiple platforms under GNU Lesser General Public License (LGPL) version 2.1 with an additional exception.

How to get OCCT?

You can obtain Open CASCADE Technology by:

- Downloading a release from the official website:
  - Create an account on the OPEN CASCADE company official website.
  - Once registered, you will get access to the download page.
- Building OCCT from sources (Git repository).

To obtain OCCT sources from the official Git repository, you should:

- Create an account on the collaborative development portal.
- Clone OCCT Git repository (using your ssh key).
- Download necessary 3rd-party libraries.

Once OCCT sources are downloaded, you can generate project files for your IDE using CMake.

Building from sources

OCCT building from sources involves several steps:

1. Download the OCCT source code.
2. Download necessary third party products (TCL and FreeType are minimal requirements for the standard build).
3. Configure CMake:
    1. Push the “configure” button.
    2. Set `3RDPARTY_DIR` variable pointing to unpacked third parties folder.
    3. Change `INSTALL_DIR` to any desirable directory.
    4. Enable necessary products.
    5. Enable `BUILD_USE_PCH` to speed up calculations using precompiled headers if no OCCT development is expected.
4. Push the “configure” button again.
5. Push the “generate” button.

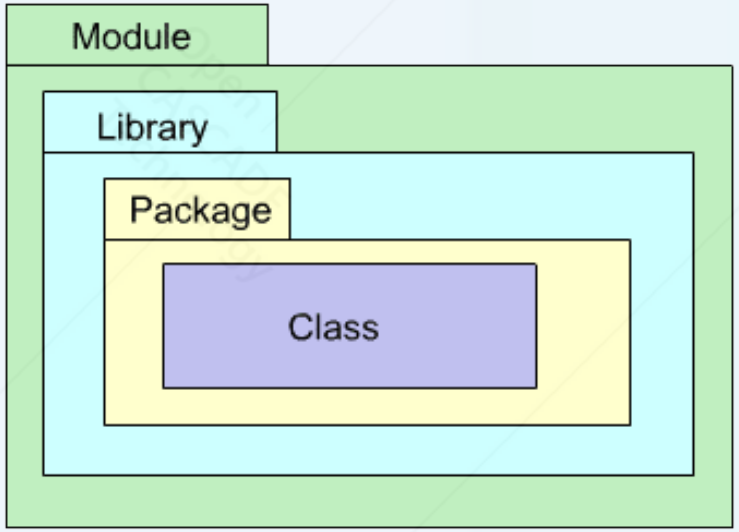
Video instruction: 01_preliminaries_occt_build_from_sources.mp4

What is in distribution?

Object libraries

Object libraries are grouped into modules:

- Foundation classes.
- Modeling data.
- Modeling algorithms.
- Visualization.
- Data exchange.
- Shape healing.
- Application framework (OCAF).



Each module contains several libraries, each library contains classes grouped into packages:

What is in distribution?

Documentation

- Automatically generated from Doxygen comments.
- Manually written user and developer guides.

Programming samples

Programming samples using different GUI:

- MFC.
- C#.
- Qt.
- Java.

Test harness application

- TCL-based command interpreter.
- A set of predefined commands.
- Most of OCCT API is available in Test Harness.
- Test Harness is a prototyping framework of OCCT.
- This application is used to test OCCT itself.

What you should know

Mandatory knowledge:

- C++ object-oriented language.
- Programming technologies.
- Common mathematics, algebra, geometry.

Optional knowledge :

- Basics of computer-aided design.

Literature

“Introduction to solid modeling” by Martti Mantyla

- Introduction-level book for a newcomer.
- Requirements to geometric modeling: geometric question concept.
- Various modeling techniques: wireframe, boundary representation, voxels. Their advantages and drawbacks.
- Difference between geometric modeling and other kinds of engineering software.

“Solid modelling and CAD systems” by Ian Stroud and Hildegarde Nagy

- Assembly structure.
- General pieces of advice about data model organization.

“Parametric and Feature Based CAD/Cam: Concepts, Techniques, and Applications” by Jami J. Shah and Martti Mantyla

- Feature concept.
- Parametric modeling concept.

Literature

“A Practical Guide to Splines” by Carl de Boor

- OCCT follows this book for underlying spline mathematics.
- Code samples are based on Fortran.
- Indexes start from 1.

“The NURBS Book” by Les Piegl and Wayne Tiller

- SMLib kernel is based on this book.
- The best book about b-splines.
- A lot of code samples.
- Code samples are based on C/C++.
- Indexes start from 0.

“Geometric Modeling” by Nikolay Golovanov

- C3D Toolkit is built on this book.
- Covers a wide set of problems.
- Utilizes the recipe-based approach.

Performance tips

Parallel mode

Several algorithms have parallel mode:

- BRepAlgoAPI
- BRepMesh_IncrementalMesh
- BVH Tree construction

Part-related

- Use simple objects to get better performance:
  - C2 continuity;
  - Low degree splines;
  - Small knot vector
- Use canonical geometry where possible:
  - Effective implementations for many algorithms
- Use topological information instead of geometric data:
  - Data sharing, connectivity, etc.

Data model

Utilize data sharing. This technique is called instancing:

- Use the same part several times, placing them with different transformations
- Also, this concept can be reused on the visualization level (AIS Connected Interactives)

Performance tips

Visualization

Try to keep a number of primitive groups as low as possible:

- Each group involves corresponding set of OpenGL operations (glDrawArrays / glDrawElements). These calls are computationally costly.
- Big amount of Primitive groups affects CPU performance. Each group should be processed individually on the CPU side. In the case of OpenGL, this process works in a single thread mode.

Coding

- For big order matrices do not use OCCT linear algebra since it is designed for low-order inputs.
- Use OCCT allocators to avoid dynamic memory allocation.
- Use accelerating data structures like BVH.
- Use local operations where possible.
- Use face maximization after the Boolean algorithm.
- Avoid usage of NCollection_Sequence. Use NCollection_Vector or NCollection_List instead.

TRAINING

Open CASCADE TECHNOLOGY

Preliminaries: Draw

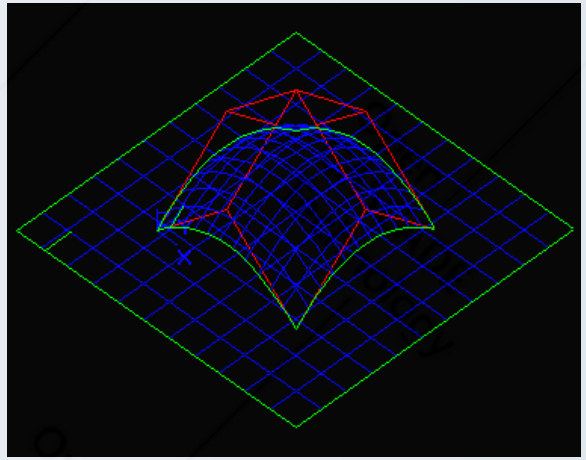
What is Draw?

Draw is a basic testing tool for Open CASCADE Technology libraries. It provides:

- A command interpreter based on TCL language.
- A set of predefined commands.
- A possibility to add custom commands.

In Draw you may create and manipulate, save, and restore entities, display them in the views, etc. Draw may be used for prototyping of custom scenarios for automated testing, etc. Open CASCADE Technology distribution provides a ready-to-use implementation of the Draw executable called DRAWEXE. Start it as follows:

```
call %CASROOT%\env.bat
DRAWEXE.exe
```



What is Draw?

DRAWEXE provides an interface for loading new commands from plugin libraries. Draw commands defined in Open CASCADE Technology modules are organized into such plugins and can be loaded to DRAWEXE by running the `pload` command.

```
pload [-PluginResourceFile] [[Key1] [Key2] ...
```

The default resource file for Draw plugins is %CASROOT%/src/DrawResources/DrawPlugin, it collects definitions for all standard plugins. In order to load all standard Draw commands, run:

```
Draw[1]> pload ALL
```


Draw command language

A command consists of one or more words: the first word is the name of the command, and additional words are arguments. Words are separated by spaces or tabs. The following command constructs a box:

```
Draw[1]> box b 10 10 10
```

Commands are separated by newlines or semicolons:

```
Draw[1]> box b 10 10 10; fit
```

Commands can be read from a file using the source command:

```
Draw[1]> source myscript.txt
```


Basic commands

`help` command provides help on commands:

```
Draw[1]> help [command]
```

`exit` command terminates program execution:

```
Draw[1]> exit
```

`source` command reads commands from a file:

```
Draw[1]> source box
```

`directory` command is used to return a list of all draw global variables matching a pattern:

```
Draw[1]> directory[pattern]
```

`whatis` command gets information on a draw variable:

```
Draw[1]> whatis <varname>
```

`dump` command gets long information about a draw variable:

```
Draw[1]> dump <varname>
```

`save` puts the contents of a draw variable in a file:

```
Draw[1]> save <varname> <filename>
```

`restore` puts the contents of a file in a draw variable:

```
Draw[1]> restore <filename> <varname>
```


View commands

Draw provides a set of standard screen layout commands:

```
Draw[1]> axo; pers; top; bottom; left;
right; front; back; mu4; v2d; av2d
```

`wzoom` command allows to select with the mouse the area to zoom:

```
Draw[1]> wzoom
```

Commands to pan a view:

```
Draw[1]> pu; pd; pr; pl; 2pdu; 2dpd; 2dpr;
2dpl
```

Commands to rotate the view up, down, right, left:

```
Draw[1]> u; d; l; r;
```

`display` command makes objects visible:

```
Draw[1]> display <varname> [...]
```

`donly` command makes objects visible and erases all other objects:

```
Draw[1]> donly <varname> [...]
```

`erase` command erases objects from all the views:

```
Draw[1]> erase <varname> [...]
```

Commands to clear objects from views:

```
Draw[1]> clear; 2dclear
```


3D view commands

`vinit` command initializes 3D view:

```
Draw[1]> vinit
```

`vdisplay` command makes objects visible:

```
Draw[1]> vdisplay <varname> [...]
```

Clear objects from views:

```
Draw[1]> vlcear [varname] [...]
```

Set background color:

```
Draw[1]> vsetcolorbg <r> <g> <b>
```

Set camera options:

```
Draw[1]> vcamera [options]
```

Set view parameters:

```
Draw[1]> vviewparams [options]
```

Set light options:

```
Draw[1]> vlight [options]
```


DrawTrSurf package

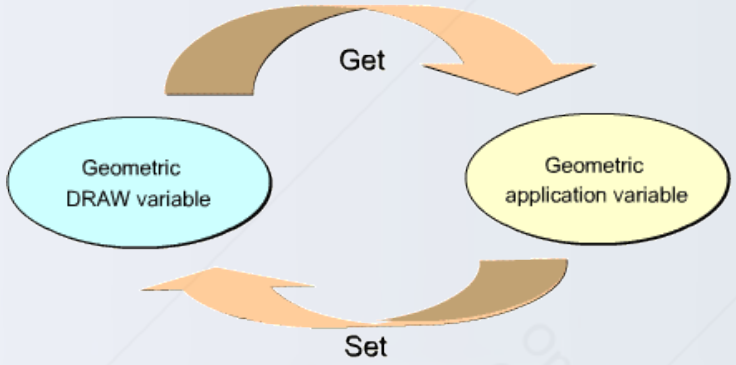
`DrawTrSurf` package is used to connect C++ geometric entities with Draw instance. The methods of this package allow to:

- Set an object to a Draw geometric variable with a given name.

```
DrawTrSurf::Set(Standard_CString, Handle(Geom_Geometry))
```

- Get an object from a Draw geometric variable with a given name.

```
Handle(Geom_Geometry)DrawTrSurf::Get(Standard_CString)
```



DBRep package

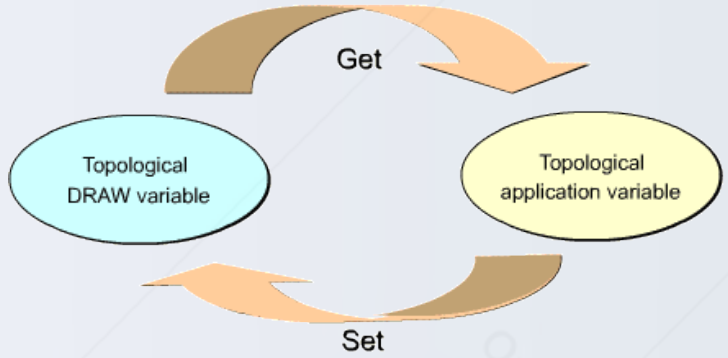
DBRep package is used to connect C++ topological entities with Draw instance. The methods of this package allow to:

- Set TopoDS_Shape to a Draw topological variable with a given name.

```
DBRep::Set(Standard_CString, TopoDS_Shape)
```

- Get TopoDS_Shape from a Draw topological variable with a given name.

```
TopoDS_Shape DBRep::Get(Standard_CString)
```



Draw-based application

To write a Draw program it is necessary to respect the following structure:

- Start and initialize Draw from the `main` function:

```
int main(int argc, char* argv[])
{
  Draw_Appli(argc, argv, Draw_InitAppli);
  return 1;
}
```

- Alternatively, the launch can be made with the use of macro:

```
#include <Draw_Main.hxx>
DRAW_MAIN
```


Definition of new commands

When you run Draw, the system calls the function `Draw_InitAppli`. This function must be defined in the executable, and the commands must be added (registered) in the Draw command interpreter within this function.

```
void Draw_InitAppli(Draw_Interpretor& theCommands)
{
  // Add standard Draw commands such as:
  Draw::Commands(theCommands);
  GeometryTest::AllCommands(theCommands);
  BRepTest::AllCommands(theCommands);

  // Add user commands here :
  theCommands.Add("mycommand",       // Command name.
                  "mycommand help", // Help string.
                  __FILE__,         // Macro giving the name of the current file.
                  MyCommand,        // C++ function name.
                  "Group name"      // Group the command belongs to.
  );
}
```


Draw plugin mechanism

Alternatively, new Draw commands can be combined into some dynamic library and loaded into the DRAWEXE executable with a plugin interface. The entry point to the plugin should be implemented as the static method Factory() of some class which should be declared with the macro DPLUGIN, for instance:

```
void ViewerTest::Factory(Draw_Interpretor& theDI)
{
  // Definition of viewer commands.
  ViewerTest::Commands(theDI);
}

// Declare entry point PLUGINFACTORY
DPLUGIN(ViewerTest)
```

OCCT is delivered with the DrawPlugin resource file located in the $CASROOT/src/DrawResources directory. The format of the file is compliant with standard OCCT resource files (see the Resource_Manager.hxx file for details). Each key defines either a sequence of further (nested) keys or the name of the dynamic library. Keys can be nested down to an arbitrary level. However, cyclic dependencies between the keys are not being checked. Example:

```
ALL                 : MODELING, OCAFKERNEL, DATAEXCHANGE
MODELING            : TOPTEST
TOPTEST             : TKTopTest
```


Command definition

An example of the definition of a new Draw command is the following:

```
#include <tcl.h>

// Command entry point.
static Standard_Integer TestBox(Draw_Interpretor&           di,
                                Standard_Integer            argc,
                                const Standard_Character** argv)
{
  // Check command arguments.
  if (argc < 5)
  {
    cout << "Invalid number of arguments\n" << endl;
    return TCL_ERROR;
  }

  // Create a solid box, compute its volume.
  const Standard_Real dx = atof(argv[2]);
  const Standard_Real dy = atof(argv[3]);
  const Standard_Real dz = atof(argv[4]);
  TopoDS_Solid S = ...;
  Standard_Real volume = ...;

  // Command output.
  DBRep::Set(argv[1], S);
  cout << argv[1] << "Volume is " << volume << endl;
  return TCL_OK;
}
```


TRAINING

Open CASCADE TECHNOLOGY

Preliminaries: OCCT basics

Standard types

OCCT defines a set of aliases to the elementary data types which are used throughout the library instead of pure C++ data type names:

OCCT type	C++ type
Standard_Integer	int
Standard_Real	double
Standard_ShortReal	float
Standard_Boolean	bool
Standard_CString	const char*
Standard_ExtString	const short*
Standard_Address	void*

Standard_Boolean can take one of the two values:

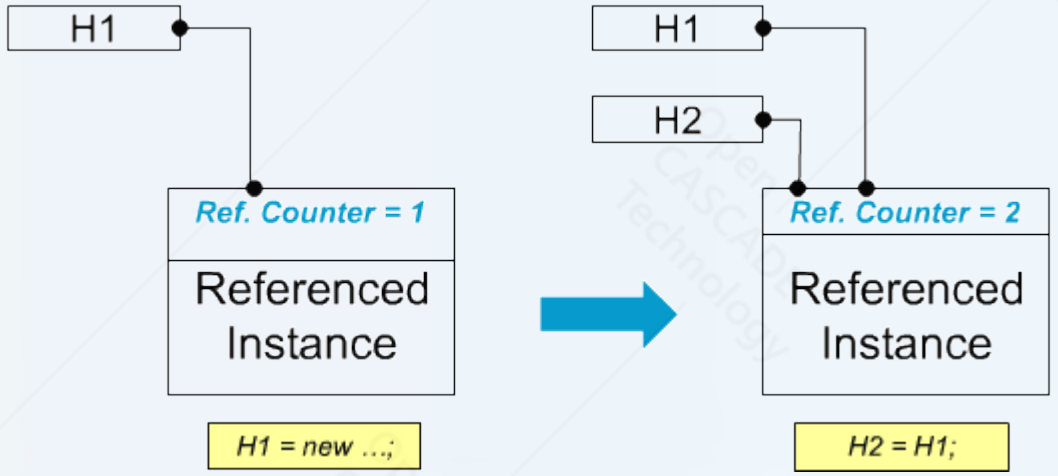
Standard_Boolean value	C++ value
Standard_False	false
Standard_True	true

The handle mechanism

Handles are smart points used throughout the OCCT. In general, the handle mechanism is based on two elements:

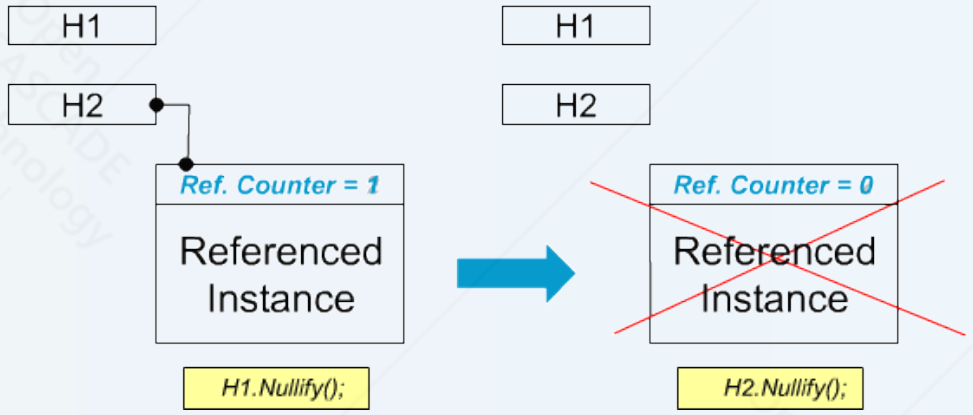
- A counter that stores the number of references to an object in memory.
- A pointer to an object in memory.

Every time a new handle to an object is created, the object's reference counter is incremented.



The handle mechanism

Every time a handle to an object is removed, the object's reference counter is decremented. As soon as the reference counter reaches 0, the object is automatically deleted.



This way, the reference counter secures the delete function. Handles can refer to nothing like usual pointers.

Handles and smart pointers

There are built-in smart pointers starting from the C++11 standard. What is the difference between OCCT handles and smart pointers?

Ownership type	Smart pointer type	Usage
shared	shared_ptr	Multiple ownership to underlying pointer
temporary	weak_ptr	Cyclic dependencies avoidance
unique	unique_ptr	Single ownership to underlying pointer

It is easy to see that OCCT's handles are the full equivalent of the shared pointer from C++11. OCCT does not contain equivalents of other smart pointer types.

Handles implementation

This section presents how to implement a new class wrapped by OCCT's handles mechanism. The new class should be inherited from the `Standard_Transient` class or its descendant:

```
class BRepAdaptor_HCurve : public Adaptor3d_HCurve
```

The special macro should be declared in the class definition for the possibility of creation of a handle to class:

```
DEFINE_STANDARD_RTTI_INLINE(BRepAdaptor_HCurve,Adaptor3d_HCurve)
```

The complete sample can be found in OCCT source code:
`\src\BRepAdaptor\BRepAdaptor_HCurve.hxx`

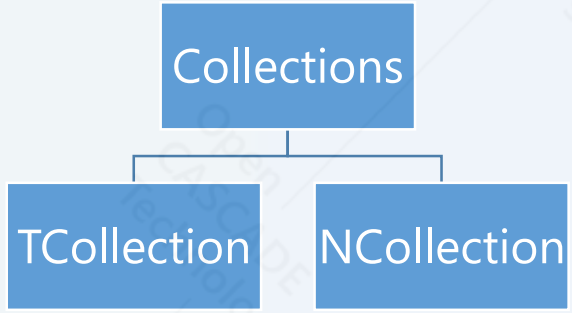
OS abstraction layer

OCCT provides platform-independent interfaces to low-level operating system facilities in the `OSD` package. This package is similar to the "os" module from the "python" programming language. The following functionality is presented:

- Method `OSD::SetSignal()` sets up a C signal handler that converts software signals (such as floating-point errors, access violation, etc.) to usual C++ exceptions.
- Classes `OSD_File`, `OSD_Path`, `OSD_Directory` give a uniform interface to file system.
- Classes `OSD_Mutex` and `OSD_Thread` provide a uniform interface to system-dependent implementation of mutexes and threads.
- Class `OSD_Timer` implements a convenient timer object.
- Classes `OSD_Host`, `OSD_Process` provide access to the information on the current computer and process.

Collections

Historically, there are two collection sets within OCCT.



`TCollection` package was used to emulate templates before its standardization in 1998. Starting from OCCT 7.0, it is replaced by the corresponding `NCollection` instances.

`NCollection` package is a newer template-based package providing modern collections. It should be used instead of `TCollection` classes where possible.

Indexation starts from 1 in OCCT except from `NCollection_Vector`, where indexation starts from 0. It is done to be compatible with STL-based algorithms.

Collection classes are manipulated by value; the versions manipulated by Handle are also provided for most of them. For instance, see `TCollection_HArray1`.

Collections & STL

There are a lot of collections in OCCT; it is fruitful to compare them with STL collections since they are a universal reference point for a C++ language. As usual, there are several advantages and disadvantages in comparison to STL.

Advantages:

- Interface. OCCT collections are designed and implemented to be human-readable. The following piece of code demonstrates STL equivalent of the OCCT's `Seek()` method for maps:

```
//! Inserts value to the map if it does not have and
//! returns reference to stored value.
//! \param [in] m Map to perform search in.
//! \param [in] k Key.
//! \param [in] v New value to insert.
template <class M,class Key>
typename M::mapped_type&
GetElseUpdate(M&                                m,
              const Key&                        k,
              const typename M::mapped_type& v)
{
  return m.insert(typename M::value_type(k, v)).first->second;
}
```

- Memory efficiency. OCCT collections are designed to be memory efficient. For instance, when a new element is added to `NCollection_Vector` and a repack is expected; a new chunk of fixed size will be allocated instead of complete memory reallocation. A new element will be added to a new chunk instead of memcpy and element assignment.

Collections & STL

Disadvantages:

- Random index access is slower than in STL collections. The measurable differences can be observed in some extreme cases. It is possible to face that problem during low-level memory optimizations when indirect addressing makes sense.

Collections comparison:

OCCT class	STL equivalent	C++11 STL equivalent	comment
NCollection_Array	-	std::array	Lower and upper indexes are defined at the construction time
NCollection_Vector	std::vector	-	Indexation starts from 0
NCollection_List	std::list	std::forward_list	OCCT list is one-directional list
NCollection_IndexedMap	std::set	std::unordered_set	OCCT uses hash map instead of tree
NCollection_IndexedDataMap	std::map	std::unordered_map	OCCT uses hash map instead of tree

Exceptions

By convention, all exception classes used in Open CASCADE libraries are inherited from the class `Standard_Failure`. A hierarchy of exception classes derived from it is defined in the package `Standard`.

For robust work of OCCT algorithms in case of numerical instabilities, it is necessary to call the method `OSD::SetSignal()` somewhere in the application (once; on Windows - once in each execution thread).

After that, both standard C++ exceptions and software signals can be treated in a uniform way in a usual C++ manner:

```
// Some user function.
void SomeFunction(arguments)
{
  ...

  try
  {
    // Set up a special signal handler code (necessary for Linux/UNIX only).
    OCC_CATCH_SIGNALS

      // Do the actions.
      ...
  }
  catch (Standard_Failure& anErr)
  {
    // Treat error here.
    ...
  }
}
```


Messages

Package `Message` provides facilities for OCCT and application-level algorithms to communicate with the user in the abstract and customizable way:

- Class `Message_Messenger` is an interface for the output of string messages. It can be customized by directing the output to one or more streams, GUI window, or whatever else, in accordance with the application needs.
- Class `Message_Msg` provides means for messages localization. It is initialized by message key and parameters and constructs a corresponding message as a text string. The definitions of the message strings associated with keys are loaded from the resource file.
- Class `Message_Algorithm` provides a convenient way for classes to implement complex algorithms to return an extended status of execution, including user-oriented messages on encountered problems and situations.

Progress indicator

The progress indicator is an entity that allows tracking progress of downstream algorithms and reports it. Class `Message_ProgressIndicator` provides an interface for complex algorithms to inform about their current status of execution during runtime, with a possibility to break upon the user command. Currently (OCCT 7.4.0), there is no support for concurrent progress indicator that could be useful in multi-threaded applications. Typical wrapper for progress indicator is presented below:

```
class core_ProgressIndicator : public Message_ProgressIndicator
{
public:

  //! Constructor.
  core_EXPORT core_ProgressIndicator();

  //! Update progress state
  //! \param[in] theForce the flag for forcing update
  //! \return false if update interval has not elapsed and true otherwise.
  virtual Standard_Boolean Show( const Standard_Boolean theForce =
Standard_True );

  //! Check for cancelled state.
  //! \return True if the user has send to a break signal
  virtual Standard_Boolean UserBreak();

  ...

};
```


About Open Cascade

It is a software development company which is laser-focused on digital transformation of industries through the use of 3D technologies.

Open Cascade offers a wide range of high-performance proprietary 3D software tools both open-source and commercial. The first ones have been developed, maintained and continuously improved since 2000. Whereas the second ones have been progressively aggregated in the Commercial Platform based on which the company offers creating modern tailor-made industrial solutions that meet even the most sophisticated client's requirements.

Moreover, Open Cascade expands its portfolio by offering end-user industrial software products and delivering software customization and integration services. Open Cascade provides its solutions and services worldwide. The company is a part of the Capgemini's Digital Engineering and Manufacturing Services global business line.

Learn more about Open Cascade at www.opencascade.com

OPEN CASCADE

Backing your path to digital future