Chapter

20 *Using OM Access*

This chapter describes the OM Access feature.

OM Access is a module that provides the end user with access to the information in a diagram created by the OM Editor through a C++-interface, based on the UML Meta-model.

OM Access

OM Access is a C++ application programmer's interface to the OM Editor. It can be used in applications that uses the information contained in OM diagrams, such as:

- Code generators
- Documentation generators
- Report generators
- Statistics.

Note:

This chapter is an OM Access primer and is not intended to provide knowledge about C++ programming.

OM Access Files

These files can be found under the `<installation dir>/orca/omaccess/>` directory of your installation.

Filename	Description
`stlmini.h`	Minimal implementation of the standard C++ library, used by `omaccess.cc`
`uml.h`	Declarations of UML Meta-structure
`omaccess.h`	Header-file for `omaccess.cc`
`omaccess.cc`	Functions for accessing the data in an OM diagram
`pmtool.h`	Declarations for low-level communications with the OM Editor
`om2cpp.cc`	A small demo program that takes a diagram and makes a C++ include file
`trace.cc`	A small demo program that prints almost everything it gets from OM Access

General Concepts

The OM Access Application

An OM Access application is an application that uses the data in an OM Diagram. This OM Diagram is represented in the program by a data structure similar to UML's metamodel.

The first section introduces the basic methods.

Basic Methods

First of all there are some definitions needed; these can be found in the include file `omaccess.h`:

```
#include "omaccess.h"
```

This file also includes `uml.h` and `stlmini.h`. The file `uml.h` contains declarations for the UML metamodel, and `stlmini.h`[1] contains a minimal implementation of `string` and `list` classes in the standard C++ library used for handling the data.

An instance of the `OMModule` is needed to hold the information:

```
OMModule module;
```

To load the contents of a diagram into the module either `GetFile` or `GetBufID` can be used:

```
bool result = GetFile(module, filename);
bool result = GetBufID(module, bufid);
```

Returns `true` on success, false if failed to load the module.

1. `stlmini.h` can be replaced by the appropriate standard C++ headers `<list.h>` and `<string.h>` if you have access to an implementation of the standard C++ library.

Accessing the Information

The `OMModule` class keeps the information fetched from the OM Editor in lists. The lists are:

```
list<Class> classList
list<Generalization> generalizationList
list<AssociationClass> associationlist
```

Aggregations and associations are stored into the `associationList`, to simplify handling. To pick out the associations and aggregations separately, use the functions:

```
GetAggregations(OMModule&, list<AssociationClass>*)
GetAssociations(OMModule&, list<AssociationClass>*)
```

These functions fill the list in the second parameter with the associations and aggregations in the module.

The lists can be traversed using iterators. The following example shows how to print all the names of the classes in a module:

Example 155 ----------

```
for (list<Class>::iterator ci=omModule.classList.begin();
     ci != omModule.classList.end();
     ++ci) {
        const Class &omClass = *ci;
        cout << omClass.name << endl;
}
```

Relations

The links between associations/aggregations/generalizations and classes are represented as the names of classes, stored as `strings`.

Generalizations

The `Generalization` class contains the data members `subtype` and `supertype`. The names of the super- and subclass are stored as `string`.

The data member `discriminator` contains the discriminator, stored as `string`.

To list the superclasses and subclasses of a class, the following functions can be used:

```
GetSuperClassList(OMModule&, string&, list<string>*)
GetSubClassList(OMModule&, string&, list<string>*)
```

They put the names of the superclasses or subclasses belonging to the name of the class in `string`, into the `list`.

Aggregations/Associations

The `AssociationClass` class can contain both aggregations and associations as well as associations with association classes connected to them. The `boolean` member `isAggregation` can be used to determine if it is an aggregation or association.

To get a list of the names of the classes connected to the association/aggregation, the function

```
GetEndPoints(AssociationClass&, list<string>*)
```

can be used.

Example

Here follows a small example that shows how to access the class information. For each class in the diagram it prints the name of the class and the name of its subclasses.

Example 156: printclasses.cc

```
#include <iostream.h>
#include "omaccess.h"

#include "stlmini.h"

int main(int argc, char *argv[]) {
        if (argc !=2) {
                cout << "Usage : printclasses <filename>" << endl;
                return EXIT_FAILURE;
        }

        OMModule omModule;

        // Retrieve the file from the OM Editor
        if(! GetFile(argv[1], &omModule) )
                return EXIT_FAILURE;

        // Iterate over the classes
        for (list<Class>::iterator ci=omModule.classList.begin();
             ci != omModule.classList.end();
             ++ci) {
                const Class &omClass = *ci;
                cout << omClass.name << endl;

                // Extract the subclassess
                list<string> classlist;
                GetSubClassList(omModule,omClass.name,&classlist);
                // And iterate through them
                for (list<string>::iterator ni=classlist.begin();
                     ni != classlist.end();
                     ++ni) {
                        const string &name = *ni;
                        cout << " Subclass: " << name << endl;
                        }
        }
        return EXIT_SUCCESS;
}
```

Files and Compiling

General

To use OM Access a C++ compiler that can handle templates is needed, such as g++ 2.7.2, Borland C++ 5.02, or MSVC 5.

There are some Makefiles together with the examples (`$telelogic/orca/omaccess/examples`) that might be usable as templates.

OM Access is based on the public interface to Telelogic Tau; see "Post-Master Reference" on page 490 in chapter 11, *The PostMaster*.

UNIX

The following files should be included into the compilation/linking phase for UNIX is:

```
$telelogic/orca/omaccess/src/omaccess.cc
$telelogic/orca/omaccess/src/pmtool.cc
$telelogic/lib/<platform>lib/post.o
```

where `$telelogic` refers to the installation path, and `<platform>` can be one of `hppa` or `sunos5`.

Two paths to the include files is also necessary:

```
-I$telelogic/orca/omaccess/include
-I$telelogic/include/post
```

Example 157: Compile `printclasses.cc` with g++ on HP

```
g++ -o printclasses -I$telelogic/orca/omaccess/include \
        -I$telelogic/include/post \
        printclasses.cc \
        $telelogic/orca/omaccess/src/omaccess.cc \
        $telelogic/orca/omaccess/src/pmtool.cc \
        $telelogic/lib/hppalib/post.o
```

If Solaris is used, the switches `-lgen -lsocket -lnsl` have to be added.

Example 158: Compile `printclasses.cc` with g++ on Solaris

```
g++ -o printclasses -I$telelogic/orca/omaccess/includes \
        -I$telelogic/include/post \
        printclasses.cc \
        $telelogic/orca/omaccess/src/omaccess.cc \
        $telelogic/orca/omaccess/src/pmtool.cc \
        $telelogic/lib/sunos5lib/post.o -lgen -lsocket -lnsl
```

The OM Access files could also be copied to a local directory to ease up compilation.

Windows

> **Note:**
>
> During runtime the DLL `post.dll` (located in a subdirectory to `<installation directory>\sdt\sdtdir\wini386\include`) must either be in the path or the directory the application is started from.

Borland C++

Add the files `omaccess.cc, pmtool.cc` (can be found in `<installation directory>\orca\src`) and `post.lib` (can be found in `<installation directory>\sdt\sdtdir\wini386\include\borland502`) to the project.

Add `<installation directory>\orca\includes` and `<installation directory>\include\post` to the include path.

Microsoft Visual C++

Add the files `omaccess.cc, pmtool.cc` (can be found in `<installation directory>\orca\src`) and `post.lib` (can be found in `<installation directory>\sdt\sdtdir\wini386\include\msvc50)` to the project.

Add `<installation directory>\orca\includes` and `<installation directory>\include\post` to the include path.

Using OM Access Together with the SDL Suite

Applications written with OM Access can easily be called from within the OM Editor, for example from a menu (for more info about defining your own menus see *"Defining Menus in the SDL Suite" on page 18 in chapter 1, User Interface and Basic Operations*.

The following example adds a choice to generate C++ code skeleton of an OM Diagram from the OM Editor and pops up a text editor with the generated file in it.

Example 159: `ome-menus.ini` ----------

```
SDT-DYNAMICMENUS-3.6
[MENU]
Name=&OM Access
[MENUITEM]
ItemName=Generate C++
Separator=off
StatusbarText=Generate C++ code skeleton
ProprietaryKey=1
AttributeKey=0
Scope=Always
ConfirmText=You might want to change the name of the output file
ActionInterpretation=OS_COMMAND
BlockCommand=off
FormattedCommand=om2cpp %b `basename %b .som`.h
[MENUEND]
```

Reference

The reference contains two parts:

- Data Model: information about the data model used. For each class, a description is provided and then the members of the class are presented in a table.
- Functions: information about the functions in OM Access. For each function, the definition is shown followed by a description and a table of the parameters.

Data Model

The data model in OM Access is based on the UML Metamodel. For more information about the UML Metamodel, see the "UML Semantics" chapter of the "Unified Modeling Language, version 1.1" documentation. This document can be found at `http://www.rational.com/uml/documentation.html`.

If not mentioned otherwise, the members are of type `string`.

OMModule

An instance of the `OMModule` class contains information about an OM Diagram.

Member	Description
`classList`	A list of `Class`, containing all of the classes.
`generalizationList`	A list of `Generalization`, containing all of the generalizations.
`associationList`	A list of `AssocationClass`, containing all of the associations/aggregations.

Class

An instance of the `Class` class contains information about a class.

Member	Description
`name`	The class name as a string.
`attributeList`	A list of `Attribute`, each entry representing one attribute in the class.
`operationList`	A list of `Operation`, each entry representing one attribute in the class.

Operation

An instance of the `Operation` class contains information about an operation.

Member	Description
`visibility`	Visibility of operation.
`name`	Name of operation.
`returnType`	Return type of operation.
`parameterList`	List of `Parameter` containing information about the operations parameters.

Parameter

An instance of the `Parameter` class contains information about a parameter.

Member	Description
`name`	Name of the parameter.
`type`	Type of the parameter.

Attribute

An instance of the `Attribute` class contains information about an attribute.

Member	Description
`visibility`	Visibility of attribute.
`name`	Name of attribute.
`type`	Type of the attribute.
`value`	Default value.

Generalization

An instance of the `Generalization` class contains information about a generalization.

Member	Description
`subtype`	The name of the subtype, ‘subclass’ of the generalization.
`supertype`	The name of the supertype, ‘superclass’ of the generalization.
`discriminator`	The discriminator of the generalization.

AssociationClass

An instance of the `AssociationClass` class contains information about an association/aggregation. The connection between the different classes is maintained by `fromEnd` and `toEnd`.

Member	Description
`fromEnd`	An `AssociationEnd` representing one of the ends of the association.
`toEnd`	An `AssociationEnd` representing the other end of the association.

Member	Description
isAggregation	A boolean that is true if the association is an aggregation.
hasAssociationClass	A boolean that is true if the association also have an AssocationClass.

If the association also has an associationclass, the class-information is contained in the instance, inherited from Class.

AssociationEnd

An instance of the AssociationEnd class contains information about a connection to a class, for example in an association.

Member	Description
name	Name of this end.
rolename	Role name of association.
type	Name of the class connected to.
multiplicity	Multiplicity of association.
qualifier	Qualifier of association.
constraint	Constraint of association.
aggregation	Represents the type of the association.
isSorted	True if the association is sorted.
isOrdered	True if the association is ordered.

Functions

GetFile

```
bool GetFile(filename, omModule, status)
```

Description

Loads the contents of `filename` into `omModule`. Returns `true` on success, `false` if failed to load the module.

Parameters

Parameter	Type	Description
`filename`	`string`	The (absolute) name of the file containing the diagram.
`omModule`	`OMModule *`	A pointer to the OMModule where the diagrams data will be put.
`status`	`string *`	(optional) If loading failed a diagnostic message is inserted into this string.

GetBufID

```
bool GetBufID(bufID,omModule,status)
```

Description

Loads the contents of the buffer `bufID` into `omModule`. Returns `true` on success, `false` if failed to load the module.

Parameters

Parameter	Type	Description
`bufID`	`string`	The bufferID of the diagram.
`omModule`	`OMModule *`	A pointer to the OMModule where the diagrams data will be put.
`status`	`string *`	(optional) If loading failed a diagnostic message is inserted into this string.

GetSuperClassList

```
GetSuperClassList(omModule,className,superClasses)
```

Description

Fills the list `superClasses` with the names of the superclasses to `className` in `omModule`.

Parameters

Parameter	Type	Description
`omModule`	`OMModule &`	The OMModule containing the diagram information.
`className`	`string`	The name of the class whose superclasses are searched for.
`superClasses`	`list<string> *`	The list to put the name of the superclasses in.

GetSubClassList

```
GetSubClassList(omModule,className,superClasses)
```

Description

Fills the list `subClasses` with the names of the subclasses to `class-Name` in `omModule`.

Parameters

Parameter	Type	Description
`omModule`	`OMModule &`	The OMModule containing the diagram information.
`className`	`string`	The name of the class whose subclasses are searched for.
`subClasses`	`list<string> *`	The list to put the name of the subclasses in.

SourceAggregationEnd

```
string SourceAggregationEnd(association)
```

Description

Returns the name of the class which starts the aggregation (the end with the square in it), the owner-part. (The `source` contains the `end`.)

Parameters

Parameter	Type	Description
`association`	`AssociationClass`	The association to check.

TargetAggregationEnd

```
string TargetAggregationEnd(association)
```

Description

Returns the name of the class that ends the aggregation (the end without the square in it), the owned-part. (The `source` contains the `end`.)

Parameters

Parameter	Type	Description
`association`	`AssociationClass`	The association to check.

GetEndPoints

```
void GetEndPoints(association, classes)
```

Description

Fills the classes list with the names of the classes that this association/aggregation is connected to.

Parameters

Parameter	Type	Description
`association`	`AssociationClass`	The association to check.
`classes`	`list<string> *`	The list to insert the names into.

HasEndpoint

```
bool hasEndPoint(association, name)
```

Description

Returns true if `association` has an endpoint at `name`.

Parameters

Parameter	Type	Description
`association`	`AssociationClass`	The association to check.
`name`	`string`	The class to check.

GetAggregations

```
void GetAggregations(omModule,aggregations)
```

Description

Fills `aggregations` with all the aggregations in `omModule`.

Parameters

Parameter	Type	Description
`omModule`	`OMModule`	The OMModule containing the diagram information.
`aggregations`	`list<AssociationClass>`	The list to fill with aggregations.

GetAggregations

```
void GetAggregations(comModule,aggregations,owner)
```

Description

Fills `aggregations` with all the aggregations in `omModule`, that has `owner` as source.

Parameters

Parameter	Type	Description
`omModule`	`OMModule`	The OMModule containing the diagram information.
`aggregations`	`list<AssociationClass>`	The list to fill with aggregations.
`owner`	`string`	the name of the class that has to be the owner.

GetAssociations

```
void GetAssociations(omModule, associations)
```

Description

Fills `associations` with all the associations in `omModule`.

Parameters

Parameter	Type	Description
`omModule`	`OMModule`	The OMModule containing the diagram information.
`associations`	`list<AssociationClass>`	The list to fill with associations.

GetAssociations

```
void GetAssociations(omModule, associations,
                     firstEnd)
```

Description

Fills `associations` with all the associations in `omModule`, that is connected to `firstEnd`, in some end.

Parameters

Parameter	Type	Description
`omModule`	`OMModule`	The OMModule containing the diagram information.
`associations`	`list<AssociationClass>`	The list to fill with associations.
`firstEnd`	`string`	The name of one of the classes that the association has to be connected to.

GetAssociations

```
void GetAssociations(omModule,associations,
                     firstEnd,secondEnd)
```

Description

Fills `associations` with all the associations in `omModule`, that is connected to `firstEnd`, and `secondEnd`.

Parameters

Parameter	Type	Description
`omModule`	`OMModule`	The OMModule containing the diagram information.
`associations`	`list<AssociationClass>`	The list to fill with aggregations.

Parameter	Type	Description
firstEnd	string	The name of one of the classes that the association has to be connected to.
secondEnd	string	The name of the second class that the association has to be connected to.

TraceModule

```
void TraceModule(omModule,os)
```

Description

Outputs a textual dump of `omModule` to `os`, with as much information as possible, useful for debugging purposes.

Parameters

Parameter	Type	Description
omModule	OMModule	The OMModule containing the diagram information.
os	ostream	The stream to output the dump to.