LibOV API Reference

Dirk Meyer <dirk@plt.rwth-aachen.de>
Christian Poensgen <christian@plt.rwth-aachen.de>

LibOV API Reference (Version 1.2.1)

1 Legal terms and conditions

2 Preprocessor symbols used in the library

When compiling the object management library the following preprocessor symbols are used:

2.1 Definition of the compiler used

The preprocessor symbol `OV_COMPILER_`*COMPILER* defines which compiler is used (symbol is defined as `1`) and which ones not (symbol is defined as `0`).

For instance, if you are compiling using the Borland compiler, the following symbols are defined:

```
#define OV_COMPILER_BORLAND 1
#define OV_COMPILER_CYGWIN  0
#define OV_COMPILER_GCC     0
```

These symbols are automatically defined in the configuration header file `libov/ov_config.h`.

Available Symbols:

```
OV_COMPILER_CYGWIN
OV_COMPILER_GCC
OV_COMPILER_MSVC
OV_COMPILER_BORLAND
OV_COMPILER_KEIL
OV_COMPILER_DECCXX
```

2.2 Definition of the target operating system

The preprocessor symbol `OV_SYSTEM_`*SYSTEM* defines for which target operating system the library is compiled (symbol is defined as `1`) and for which ones not (symbol is defined as `0`).

If the target operating system is any Unix system (e.g. Linux, HP-UX, Solaris, ...), the more generic symbol `OV_SYSTEM_UNIX` is defined as `1` and defined as `0` otherwise.

For instance, if you are compiling on a Linux system, the following symbols are defined:

```
#define OV_SYSTEM_UNIX  1
#define OV_SYSTEM_LINUX 1
#define OV_SYSTEM_NT    0
```

Usually, you should define the appropriate target operating system when calling the compiler, e.g.

```
gcc -Wall -O2 -DOV_SYSTEM_LINUX=1 -c foo.c -o foo.o
```

All `OV_SYSTEM_`*SYSTEM* symbols not defined are automatically defined as `0` in the configuration header file `libov/ov_config.h`.

Available Symbols:

```
OV_SYSTEM_NT
OV_SYSTEM_HPUX
OV_SYSTEM_LINUX
OV_SYSTEM_OPENVMS
OV_SYSTEM_SOLARIS
OV_SYSTEM_MC164
OV_SYSTEM_RMOS
```

2.3 Definition whether debugging features are used

The preprocessor sysmbol `OV_DEBUG` defines if you want to use the ACPLT/OV debugging features defined in the header file `libov/ov_debug.h`. If you define this symbol, the debugging macros log debugging information to the logfile. If you do not define this symbol, the debugging macros are ignored during compilation.

For example, the compiler call

```
gcc -Wall -O2 -DOV_SYSTEM_LINUX=1 -DOV_DEBUG -c foo.c -o foo.o
```

compiles using the debugging features.

You may also define `OV_DEBUG` in your source file:

```
/*
*   foo.c
*/

#define OV_DEBUG

#include "libov/ov_ov.h"
#include <stdio.h>

void foo(
    OV_STRING   text
) {
    Ov_WarnIfNot(text); /* warn if text is NULL */
    printf(text);
}
```

Make sure, that you define the symbol **before** including any ACPLT/OV header files!

2.4 Definition whether static libraries are used

The preprocessor symbol `OV_STATIC_LIBRARIES` defines whether there are ACPLT/OV libraries statically linked to the application (symbol is defined as `1`) or not (symbol is defined as `0`).

Usually if the target operating system supports dynamic linking (DLLs on Windows NT, shared libraries on Unix systems), this symbol is defined as `0`), because you get the advantage, that you do not have to determine the libraries available for the application at link time but can load them at run time.

This symbol is automatically defined in the configuration header file `libov/ov_config.h`.

2.5 Definition whether dynamic libraries are used

The preprocessor symbol `OV_DYNAMIC_LIBRARIES` defines whether ACPLT/OV libraries can by dynamically linked to the application (symbol is defined as `1`) or not (symbol is defined as `0`).

This symbol can only be defined as `1` if the target operating supports dynamic linking or you are unable to compile the file `libov/ov_library.c`.

Usually if the target operating system supports dynamic linking (DLLs on Windows NT, shared libraries on Unix systems), this symbol is defined as `1`), because you get the advantage, that you do not have to determine the libraries available for the application at link time but can load them at run time.

This symbol is automatically defined in the configuration header file `libov/ov_config.h`.

2.6 Definition whether dynamically growing databases are used

The preprocessor symbol `OV_DYNAMIC_DATABASE` defines whether ACPLT/OV databases may dynamically grow when memory is allocated (symbol is defined as `1`) or not (symbol is defined as `0`).

3 Import/Export of functions and global variables

In order to generate storage class modifiers for importing and exporting functions and global variables from a dynamically linked library (DLL), the following preprocessor symbols are used:

3.1 Modifier for functions that are exported from a DLL

The preprocessor symbol `OV_DLLFNCEXPORT` is used to tell the compiler, that a function needs to be exported from a DLL. Using this symbol you do not need a module definition file (`.def`-file) listing all functions to be exported from a DLL.

This symbol is automatically defined in the configuration header file `libov/ov_config.h`.

Example

The following piece of code defines a DLL function which may be used by other modules linking to the import library of the DLL:

```
/*
*   hello.c
*/

#include "libov/ov_ov.h"
#include <stdio.h>

OV_DLLFNCEXPORT void hello(
    OV_STRING   name
) {
    printf("hello %s\n", name);
}
```

Remarks

Note, that you need to use this modifier in the declaration of function prototypes as well (compare file `ov.ovf`)!

3.2 Modifier for global variables that are exported from a DLL

The preprocessor symbol `OV_DLLVAREXPORT` is used to tell the compiler, that a global variable is exported from a DLL.

This symbol is automatically defined in the configuration header file `libov/ov_config.h`.

Example

The following piece of code defines a global variable which is exported from a DLL:

```
/*
*   foo.c
*/

#include "libov/ov_ov.h"
#include <stdio.h>

/*
*   the exported global variable
*/
OV_DLLVAREXPORT OV_STRING pstring;
```

```
/*
*   a function setting and printing the global variable
*/
void foo(void) {
    pstring = "hello world";
    printf("pstring = %s\n", pstring);
}
```

A module using the DLL can access this variable using the preprocessor symbol `OV_DLLVARIMPORT`.

3.3 Modifier for global variables that are exported from a DLL

The preprocessor symbol `OV_DLLVARIMPORT` is to tell the compiler, that a global variable is imported from a DLL.

This symbol is automatically defined in the configuration header file `libov/ov_config.h`.

Example

The following piece of code declares a global variable which is imported from a DLL:

```
/*
*   bar.c
*/

#include "libov/ov_ov.h"
#include <stdio.h>

/*
*   the imported global variable
*/
OV_DLLVARIMPORT OV_STRING pstring;

/*
*   a function setting and printing the global variable
*/
void bar(void) {
    pstring = "hello world";
    printf("pstring = %s\n", pstring);
}
```

A module using the DLL can access this variable using the preprocessor symbol `OV_DLLVARIMPORT`.

3.4 Definition whether a LibOV source file is compiled

The preprocessor symbol `OV_COMPILE_LIBOV` defines whether a source file (module) of the ACPLT/OV Object Management Library LibOV itself is compiled (symbol is defined as `1`) or not (symbol is defined as `0`).

This symbol is necessary for Windows NT in order to determine whether global variables need to be exported (symbol is defined as `1`) or imported (symbol is defined as `0`) from a DLL.

Always define this symbol if you add source files to the ACPLT/OV Object Management Library LibOV and **never** define it in any other file!

If you define the symbol, make sure that you define the symbol **before** including any ACPLT/OV header files!

4 Debugging macros used in the library

In debugging mode (`OV_DEBUG` defined) the following macros are available, otherwise they do not take effect:

4.1 Print debug information

If `OV_DEBUG` is defined, this macro logs a user-defined debugging message:
`[ACPLT/OV Debug]` *file*`:`*line*`: Info:` *info*

Header file

```
#include "libov/ov_debug.h"
```

Macro usage

```
Ov_Info(info);
```

Parameters

`info` String containing the info message.

Return value

none

4.2 Print a warning and continue

In debugging mode (`OV_DEBUG` defined) this macro logs a user-defined debugging message:
`[ACPLT/OV Debug]` *file*`:`*line*`: Warning:` *warning*

Header file

```
#include "libov/ov_debug.h"
```

Macro usage

```
Ov_Warning(warning);
```

Parameters

`warning` String containing the warning message.

Return value

none

4.3 Print an error and abort

In debugging mode (`OV_DEBUG` defined) this macro logs a user-defined debugging message:
`[ACPLT/OV Debug]` *file*`:`*line*`: Error:` *err*

Header file

```
#include "libov/ov_debug.h"
```

Macro usage

```
Ov_Error(err);
```

Parameters

`err` String containing the error message.

Return value

none

4.4 Print a warning if a condition holds

In debugging mode (`OV_DEBUG` defined) this macro logs a user-defined debugging message, if condition specified is TRUE:

`[ACPLT/OV Debug]` *file*`:`*line*`: Warning: Assertion failed: "!(`*condition*`)".`

Header file

```
#include "libov/ov_debug.h"
```

Macro usage

```
Ov_WarnIf(condition);
```

Parameters

`condition`
: Conditional expression; if TRUE, a warning is printed.

Return value

none

4.5 Print a warning if a condition does not hold

Header file

In debugging mode (`OV_DEBUG` defined) this macro logs a user-defined debugging message, if condition specified is FALSE:

`[ACPLT/OV Debug]` *file*`:`*line*`: Warning: Assertion failed: "(`*condition*`)".`

```
#include "libov/ov_debug.h"
```

Macro usage

```
Ov_WarnIfNot(condition);
```

Parameters

`condition`
: Conditional expression; if FALSE, a warning is printed.

Return value

none

4.6 Print an error if a condition holds and abort

In debugging mode (`OV_DEBUG` defined) this macro logs a user-defined debugging message, if condition specified is TRUE and aborts the process:

`[ACPLT/OV Debug]` *file*`:`*line*`: Error: Assertion failed: "!(`*condition*`)".`

Header file

```
#include "libov/ov_debug.h"
```

Macro usage

```
Ov_AbortIf(condition);
```

Parameters

`condition`
: Conditional expression; if TRUE, the process is aborted with a message.

Return value

none

4.7 Print an error if a condition does not hold and abort

In debugging mode (`OV_DEBUG` defined) this macro logs a user-defined debugging message, if condition specified is FALSE and aborts the process:

`[ACPLT/OV Debug]` *file*`:`*line*`: Error: Assertion failed: "(`*condition*`)".`

Header file

```
#include "libov/ov_debug.h"
```

Macro usage

```
Ov_AbortIfNot(condition);
```

Parameters

`condition`
: Conditional expression; if FALSE, the process is aborted with a message.

Return value

none

5 Useful macros easing the use of the library

5.1 Upcast of an instance pointer of the parent class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_ParentPtrUpCast(assoc, pparent);
```

Parameters

`assoc`

`pparent`

Return value

Remarks

5.2 Upcast of an instance pointer of the child class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_ChildPtrUpCast(assoc, pchild);
```

Parameters

`assoc`

`pchild`

Return value

Remarks

5.3 Get first child in a 1:n association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pchild = Ov_GetFirstChild(assoc, pparent);
```

Parameters

`assoc`

`pparent`

Return value

Remarks

5.4 Get last child in a 1:n association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pchild = Ov_GetLastChild(assoc, pparent);
```

Parameters

`assoc`

`pparent`

Return value

Remarks

5.5 Get next child in a 1:n association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pchild = Ov_GetNextChild(assoc, pchild);
```

Parameters

`assoc`

`pchild`

Return value

Remarks

5.6 Get previous child a 1:n association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pchild = Ov_GetPreviousChild(assoc, pchild);
```

Parameters

`assoc`

`pchild`

Return value

Remarks

5.7 Get parent in a 1:n association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pparent = Ov_GetParent(assoc, pchild);
```

Parameters

`assoc`

`pchild`

Return value

Remarks

5.8 Iterate over all child objects in a 1:n association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
Ov_ForEachChild(assoc, pparent, pchild);
```

Parameters

`assoc`

`pparent`

`pchild`

Return value

Remarks

5.9 Iterate over all child objects in a 1:n association and dynamically cast to a given child class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
Ov_ForEachChildEx(assoc, pparent, pchild, childclass);
```

Parameters

`assoc`

`pparent`

`pchild`

`childclass`

Return value

Remarks

5.10 Search a child with given identifier in a 1:n association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pchild = Ov_SearchChild(assoc, pparent, ident);
```

Parameters

`assoc`

`pparent`

`ident`

Return value

Remarks

5.11 Search a child with given identifier and cast to child class in a 1:n association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pchild = Ov_SearchChildEx(assoc, pparent, ident, childclass);
```

Parameters

`assoc`

`pparent`

`ident`

`childclass`

Return value

Remarks

5.12 Define an iterator for iterating over n:m associations

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_DefineIteratorNM(assoc, pit);
```

Parameters

`assoc`

`pit`

Return value

Remarks

5.13 Get first child in an n:m association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pchild = Ov_GetFirstChildNM(assoc, pit, pparent);
```

Parameters

`assoc`

`pit`

`pparent`

Return value

Remarks

5.14 Get last child in an n:m association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pchild = Ov_GetLastChildNM(assoc, pit, pparent);
```

Parameters

`assoc`

`pit`

`pparent`

Return value

Remarks

5.15 Get next child in an n:m association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pchild = Ov_GetNextChildNM(assoc, pit);
```

Parameters

`assoc`

`pit`

Return value

Remarks

5.16 Get previous child in an n:m association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pchild = Ov_GetPreviousChildNM(assoc, pit);
```

Parameters

`assoc`

`pit`

Return value

Remarks

5.17 Get first parent in an n:m association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pparent = Ov_GetFirstParentNM(assoc, pit, pchild);
```

Parameters

`assoc`

`pit`

`pchild`

Return value

Remarks

5.18 Get last parent in an n:m association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pparent = Ov_GetLastParentNM(assoc, pit, pchild);
```

Parameters

`assoc`

`pit`

`pchild`

Return value

Remarks

5.19 Get next parent in an n:m association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pparent = Ov_GetNextParentNM(assoc, pit);
```

Parameters

`assoc`

`pit`

Return value

Remarks

5.20 Get previous parent in an n:m association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
pparent = Ov_GetPreviousParentNM(assoc, pit);
```

Parameters

`assoc`

`pit`

Return value

Remarks

5.21 Iterate over all child objects in an n:m association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
Ov_ForEachChildNM(assoc, pit, pparent, pchild);
```

Parameters

`assoc`

`pit`

`pparent`

`pchild`

Return value

Remarks

5.22 Iterate over all child objects in an n:m association and dynamically cast to a given child class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
Ov_ForEachChildNMEx(assoc, pit, pparent, pchild, childclass);
```

Parameters

`assoc`

`pit`

`pparent`

`pchild`

`childclass`

Return value

Remarks

5.23 Iterate over all parent objects in an n:m association

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
Ov_ForEachParentNM(assoc, pit, pchild, pparent);
```

Parameters

`assoc`

`pit`

`pchild`

`pparent`

Return value

Remarks

5.24 Iterate over all parent objects in an n:m association and dynamically cast to a given parent class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
Ov_ForEachParentNMEx(assoc, pit, pchild, pparent, parentclass);
```

Parameters

`assoc`

`pit`

`pchild`

`pparent`

`parentclass`

Return value

Remarks

5.25 Link parent and child object (1:n or n:m association)

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_Link(assoc, pparent, pchild);
```

Parameters

`assoc`

`pparent`

`pchild`

Return value

Remarks

5.26 Link parent and child object with given child placement hint (1:n association)

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_LinkPlaced(assoc, pparent, pchild, childhint);
```

Parameters

`assoc`

`pparent`

`pchild`

`childhint`

Return value

Remarks

5.27 Link parent and child object with given relative child placement hint (1:n association)

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_LinkRelativePlaced(assoc, pparent, pchild, childhint, prelchild);
```

Parameters

`assoc`

`pparent`

`pchild`

`childhint`

`prelchild`

Return value

Remarks

5.28 Unlink parent and child object (1:n or n:m association)

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
Ov_Unlink(assoc, pparent, pchild);
```

Parameters

`assoc`

`pparent`

`pchild`

Return value

Remarks

5.29 Link parent and child object (1:n or n:m association)

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_LinkNM(assoc, pparent, pchild);
```

Parameters

`assoc`

`pparent`

`pchild`

Return value

Remarks

5.30 Link parent and child object with given child placement hint (n:m association)

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_LinkPlacedNM(assoc, pparent, pchild, parenthint, childhint);
```

Parameters

`assoc`

`pparent`

`pchild`

`parenthint`

`childhint`

Return value

Remarks

5.31 Link parent and child object with given relative placement hints (n:m association)

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_LinkRelativePlacedNM(assoc, pparent, pchild, parenthint, prelparent,
childhint, prelchild);
```

Parameters

`assoc`

`pparent`

`pchild`

`parenthint`

`prelparent`

`childhint`

`prelchild`

Return value

Remarks

5.32 Unlink parent and child object (1:n or n:m association)

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
Ov_UnlinkNM(assoc, pparent, pchild);
```

Parameters

`assoc`

`pparent`

`pchild`

Return value

Remarks

5.33 Upcast to a pointer of a given base class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_PtrUpCast(class, pobj);
```

Parameters

`class`

`pobj`

Return value

Remarks

5.34 Static cast to a pointer of a given class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_StaticPtrCast(class, pobj);
```

Parameters

`class`

`pobj`

Return value

Remarks

5.35 Test if it is allowed to cast to a given class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_CanCastTo(class, pobj);
```

Parameters

`class`

`pobj`

Return value

Remarks

5.36 Dynamic cast to a pointer of a given class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_DynamicPtrCast(class, pobj);
```

Parameters

`class`

`pobj`

Return value

Remarks

5.37 Create an object of a given class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_CreateObject(class, pobj, pparent, ident);
```

Parameters

`class`

`pobj`

`pparent`

`ident`

Return value

Remarks

5.38 Delete an object

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
Ov_DeleteObject(pobj);
```

Parameters

`pobj`

Return value

Remarks

5.39 Get a pointer to the static part of an object

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_GetStaticInstPtr(class, pobj);
```

Parameters

`class`

`pobj`

Return value

Remarks

5.40 Get a pointer to a part object

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_GetPartPtr(part, pobj);
```

Parameters

`part`

`pobj`

Return value

Remarks

5.41 Get pointer to the class object of an instance

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_GetClassPtr(pobj);
```

Parameters

`pobj`

Return value

Remarks

5.42 Get the vtable pointer to an object of a given class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_GetVTablePtr(class, pvtable, pobj);
```

Parameters

`class`

`pvtable`

`pobj`

Return value

Remarks

5.43 Get the vtable pointer of the direct base class of an object of a given class

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_GetBaseclassVTablePtr(class, pobj);
```

Parameters

`class`

`pobj`

Return value

Remarks

5.44 Test, if a variable definition object defines a variable with a given name

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_IsVariable(class, name, pvar);
```

Parameters

`class`

`name`

`pvar`

Return value

Remarks

5.45 Set the value of a static vector variable

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_SetStaticVectorValue(pvector, pvalue, veclen, type);
```

Parameters

`pvector`

`pvalue`

`veclen`

`type`

Return value

Remarks

5.46 Set the value of a dynamic vector variable

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_SetDynamicVectorValue(pvector, pvalue, veclen, type);
```

Parameters

`pvector`

`pvalue`

`veclen`

`type`

Return value

Remarks

5.47 Set the vector length of a dynamic vector variable

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_SetDynamicVectorLength(pvector, veclen, type);
```

Parameters

`pvector`

`veclen`

`type`

Return value

Remarks

5.48 Compare two vector variable values

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_CompareVectorValues(pvalue1, pvalue2, veclen, type);
```

Parameters

`pvalue1`

`pvalue2`

`veclen`

`type`

Return value

Remarks

5.49 Convert a time (span) into a double variable

Header file
`#include "libov/ov_macros.h"`
Macro usage
`= Ov_TimeToDouble(time, dbl);`
Parameters
`time`
`dbl`
Return value
Remarks

5.50 Convert a double into a time (span) variable

Header file
`#include "libov/ov_macros.h"`
Macro usage
`= Ov_DoubleToTime(dbl, time);`
Parameters
`dbl`
`time`
Return value
Remarks

5.51 Allocate memory in the database

Header file
`#include "libov/ov_macros.h"`
Macro usage
`= Ov_DbAlloc(type);`
Parameters
`type`
Return value
Remarks

5.52 Allocate memory in the database

Header file
`#include "libov/ov_macros.h"`
Macro usage
`= Ov_DbMalloc(size);`
Parameters
`size`
Return value
Remarks

5.53 Free memory allocated in the database

Header file
`#include "libov/ov_macros.h"`
Macro usage
`Ov_DbFree(ptr);`
Parameters
`ptr`
Return value
Remarks

5.54 Allocate memory on the stack memory

Header file
`#include "libov/ov_macros.h"`
Macro usage
`= Ov_MemStackAlloc(type);`
Parameters
`type`
Return value
Remarks

5.55 Allocate memory on the heap

Header file
`#include "libov/ov_macros.h"`
Macro usage
`= Ov_HeapAlloc(type);`
Parameters
`type`
Return value
Remarks

5.56 Allocate memory on the heap

Header file
`#include "libov/ov_macros.h"`
Macro usage
`= Ov_HeapMalloc(size);`
Parameters
`size`
Return value
Remarks

5.57 Reallocate memory on the heap

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_HeapRealloc(ptr, size);
```

Parameters

`ptr`

`size`

Return value

Remarks

5.58 Free memory allocated on the heap

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
Ov_HeapFree(ptr);
```

Parameters

`ptr`

Return value

Remarks

5.59 Duplicate a string on the heap

Header file

```
#include "libov/ov_macros.h"
```

Macro usage

```
= Ov_HeapStrdup(ptr);
```

Parameters

`ptr`

Return value

Remarks

6 Fundamental datatypes used in the library

Datatypes of object variables supported by the ACPLT/OV modeling language:
Datatypes associated with objects:
Datatypes associated with variables of an object:
Datatypes associated with functions or methods of an object:
Datatypes associated with links of an object:
Datatypes associated with access rights and authentification/verification:
Generic datatypes for different purposes:
Vector datatypes:
PV (Process Value) datatypes:
Datatypes holding properties of the ACPLT/OV metamodel:
Datatypes associated with ACPLT/KS histories:
Datatypes associated with logfiles:

6.1 Bool value

A bool value may have one out of two possible values, `TRUE` or `FALSE`.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef bool_t OV_BOOL;
```

Remarks

The datatype `bool_t` is declared in the Sun ONC/RPC header `rpc/types.h` and falls back on the `int` datatype.

The reason for using this datatype in ACPLT/OV is that using this declaration we do not need any conversion when encoding/decoding a bool value in ACPLT/OV into/from an XDR stream.

6.2 Signed integer value

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef long OV_INT;
```

6.3 Unsigned integer value

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef u_long OV_UINT;
```

Remarks

The datatype `u_long` is declared in the Sun ONC/RPC header `rpc/types.h` and falls back on the `unsigned long` datatype.

The reason for using this datatype in ACPLT/OV is that using this declaration we do not need any conversion when encoding/decoding an unsigned integer value in ACPLT/OV into/from an XDR stream.

6.4 Single precision floating point value

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef single OV_SINGLE;
```

6.5 Double precision floating point value

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef double OV_DOUBLE;
```

6.6 String value

A string value is a pointer a `NULL`-terminated `char` array.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef char* OV_STRING;
```

Remarks

`OV_STRING` does not contain the actual string but a pointer to a `NULL`-terminated `char` array. This means that you must pay careful attention to the question who the owner of the string is.

If the string belongs to the instance of an ACPLT/OV class, the instance is the owner of the string and the string must **always** be stored in the database unless the string pointer is `NULL`. The ACPLT/OV library provides the functions `ov_string_setvalue()` and `ov_string_setvecvalue()` which take care of setting such string values of instances for you. Do **not** use this function for string values that do not belong to instances of ACPLT/OV classes!

In all other cases the ownership of the string must to specially indicated. ACPLT/OV Object Management Library functions returning string values, such as `ov_path_getcanonicalpath()`, allocated the string on the memory stack. If you call such a function, you must call the function `ov_memstack_lock()` before call the desired function and call the function `ov_memstack_unlock()` when you do not need the string anymore (but always in the same routine). This way strings may be located persistently in the database or temporary on the memory stack and you will not have to care too much about the ownership.

6.7 Time or date value

A time or date value is represented by a structure which contains the time in seconds and microseconds elapsed since January 1, 1970, 00:00:00.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_UINT secs;  /* seconds */
    OV_UINT usecs; /* microseconds */
}   OV_TIME;
```

Remarks

The `usecs` member must always contain a value between `0` and `999999`.

6.8 Time span value

A time span is a time difference (duration) and represented by a structure which contains the time span in seconds and microseconds.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_INT secs;  /* seconds */
    OV_INT usecs; /* microseconds */
}   OV_TIME_SPAN;
```

Remarks

The `usecs` member must always contain a value between `-999999` and `999999`. If both members of the structure contain a value not equal to `0`, both members must have the same sign.

6.9 Macro state of an object

`OV_OBJ_STATE` is an enumeration value defining the macro state of an object.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_OS_NONE    0x00000000 /* object is not yet initialized */
#define OV_OS_INIT    0x00000001 /* object is initialized */
#define OV_OS_STARTED 0x00000002 /* object is started up */
#define OV_OS_ACTIVE  0x00000004 /* object is active */

typedef OV_ENUM OV_OBJ_STATE;
```

6.10 Type of a variable

`OV_VAR_VALUE` is an enumeration value defining the datatype of an ACPLT/OV variable.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_VT_BOOL          KS_VT_BOOL          /* bool */
#define OV_VT_INT           KS_VT_INT           /* signed integer */
#define OV_VT_UINT          KS_VT_UINT          /* unsigned integer */
#define OV_VT_SINGLE        KS_VT_SINGLE        /* single */
#define OV_VT_DOUBLE        KS_VT_DOUBLE        /* double */
#define OV_VT_STRING        KS_VT_STRING        /* string */
#define OV_VT_TIME          KS_VT_TIME          /* time */
#define OV_VT_TIME_SPAN     KS_VT_TIME_SPAN     /* time span */

#define OV_VT_BOOL_VEC      KS_VT_BOOL_VEC      /* bool vector */
#define OV_VT_INT_VEC       KS_VT_INT_VEC       /* signed integer vector */
#define OV_VT_UINT_VEC      KS_VT_UINT_VEC      /* unsigned integer vector */
#define OV_VT_SINGLE_VEC    KS_VT_SINGLE_VEC    /* single vector */
#define OV_VT_DOUBLE_VEC    KS_VT_DOUBLE_VEC    /* double vector */
#define OV_VT_STRING_VEC    KS_VT_STRING_VEC    /* string vector */
```

```
#define OV_VT_TIME_VEC      KS_VT_TIME_VEC      /* time vector */
#define OV_VT_TIME_SPAN_VEC KS_VT_TIME_SPAN_VEC /* time span vector */

#define OV_VT_STRUCT        KS_VT_STRUCT        /* structure */

#define OV_VT_VOID          KS_VT_VOID          /* void */
#define OV_VT_BYTE_VEC      KS_VT_BYTE_VEC      /* opaque byte vector */

typedef OV_ENUM OV_VAR_TYPE;
```

Remarks

Note, that the enumeration values are identical to the corresponding ACPLT/KS values of the datatype `KS_VAR_TYPE`.

6.11 Value of a variable

The value of a variable is a structure containing the datatype, the vector length and the actual value of a variable.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_VAR_TYPE vartype;
    OV_UINT     veclen;
    union {
        OV_BOOL      val_bool;           /* if vartype == OV_VT_BOOL */
        OV_INT       val_int;            /* if vartype == OV_VT_INT */
        OV_UINT      val_uint;           /* if vartype == OV_VT_UINT */
        OV_SINGLE    val_single;         /* if vartype == OV_VT_SINGLE */
        OV_DOUBLE    val_double;         /* if vartype == OV_VT_DOUBLE */
        OV_STRING    val_string;         /* if vartype == OV_VT_STRING */
        OV_TIME      val_time;           /* if vartype == OV_VT_TIME */
        OV_TIME_SPAN val_time_span;      /* if vartype == OV_VT_TIME_SPAN */
        OV_BOOL      *val_bool_vec;      /* if vartype == OV_VT_BOOL_VEC */
        OV_INT       *val_int_vec;       /* if vartype == OV_VT_INT_VEC */
        OV_UINT      *val_uint_vec;      /* if vartype == OV_VT_UINT_VEC */
        OV_SINGLE    *val_single_vec;    /* if vartype == OV_VT_SINGLE_VEC */
        OV_DOUBLE    *val_double_vec;    /* if vartype == OV_VT_DOUBLE_VEC */
        OV_STRING    *val_string_vec;    /* if vartype == OV_VT_STRING_VEC */
        OV_TIME      *val_time_vec;      /* if vartype == OV_VT_TIME_VEC */
        OV_TIME_SPAN *val_time_span_vec; /* if vartype == OV_VT_TIME_SPAN_VEC */
        char         *val_byte_vec;      /* if vartype == OV_VT_BYTE_VEC */
    }   valueunion;
}   OV_VAR_VALUE;
```

6.12 State of a variable

`OV_STATE` is an enumeration value defining the state of an ACPLT/OV variable.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_ST_NOTSUPPORTED KS_ST_NOTSUPPORTED /* no state available */
#define OV_ST_UNKNOWN      KS_ST_UNKNOWN      /* state unknown at this time */
#define OV_ST_BAD          KS_ST_BAD          /* information is bad */
#define OV_ST_QUESTIONABLE KS_ST_QUESTIONABLE /* information is questionable */
#define OV_ST_GOOD         KS_ST_GOOD         /* information is good */

typedef OV_ENUM OV_STATE;
```

Remarks

Note, that the enumeration values are identical to the corresponding ACPLT/KS values of the datatype `KS_STATE`.

6.13 Current properties of a variable

The current properties of a variable are a structure containing the value, the state and the timestamp of a variable.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_VAR_VALUE value;
    OV_STATE     state;
    OV_TIME      time;
}   OV_VAR_CURRENT_PROPS;
```

Remarks

Compare the definition of the current properties of a variable in ACPLT/KS.

6.14 Result of a function call

`OV_RESULT` is an enumeration value defining the result of a function call.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_ERR_OK               KS_ERR_OK             /* ok, no error */
#define OV_ERR_GENERIC          KS_ERR_GENERIC        /* generic error */
#define OV_ERR_TARGETGENERIC    KS_ERR_TARGETGENERIC  /* generic error in target */
#define OV_ERR_BADAUTH          KS_ERR_BADAUTH        /* bad authentification */
#define OV_ERR_UNKNOWNAUTH      KS_ERR_UNKNOWNAUTH    /* unknown authentification */
#define OV_ERR_NOTIMPLEMENTED   KS_ERR_NOTIMPLEMENTED /* not implemented */
#define OV_ERR_BADPARAM         KS_ERR_BADPARAM       /* bad parameter */
#define OV_ERR_BADOBJTYPE       KS_ERR_BADOBJTYPE     /* bad object type */

#define OV_ERR_BADNAME          KS_ERR_BADNAME        /* bad name */
#define OV_ERR_BADPATH          KS_ERR_BADPATH        /* bad path */
#define OV_ERR_BADMASK          KS_ERR_BADMASK        /* bad mask */
#define OV_ERR_NOACCESS         KS_ERR_NOACCESS       /* no access */
#define OV_ERR_BADTYPE          KS_ERR_BADTYPE        /* bad type */
#define OV_ERR_BADVALUE         KS_ERR_BADVALUE       /* bad value */

#define OV_ERR_BADFACTORY       KS_ERR_BADFACTORY     /* bad factory */
```

```
#define OV_ERR_ALREADYEXISTS     KS_ERR_ALREADYEXISTS  /* object already exists */
#define OV_ERR_BADINITPARAM      KS_ERR_BADINITPARAM   /* bad initialization parameter */
#define OV_ERR_BADPLACEMENT      KS_ERR_BADPLACEMENT   /* bad placement */

#define OV_ERR_CANTCREATEFILE    0x00010000            /* can't create file */
#define OV_ERR_CANTOPENFILE      0x00010001            /* can't open file */
#define OV_ERR_CANTLOCKFILE      0x00010002            /* can't lock file */
#define OV_ERR_CANTREADFROMFILE  0x00010003            /* can't read from file */
#define OV_ERR_CANTWRITETOFILE   0x00010004            /* can't write to file */
#define OV_ERR_CANTMAPFILE       0x00010005            /* can't map file to memory */
#define OV_ERR_BADDATABASE       0x00010006            /* bad database */

#define OV_ERR_CANTOPENLIBRARY   0x00010010            /* can't open library */

#define OV_ERR_LIBDEFMISMATCH    0x00010020            /* library def. mismatch */
#define OV_ERR_STRUCTDEFMISMATCH 0x00010021            /* structure def. mismatch */
#define OV_ERR_CLASSDEFMISMATCH  0x00010022            /* class def. mismatch */
#define OV_ERR_ASSOCDEFMISMATCH  0x00010023            /* association def. mismatch */
#define OV_ERR_VARDEFMISMATCH    0x00010024            /* variable def. mismatch */
#define OV_ERR_PARTDEFMISMATCH   0x00010025            /* part def. mismatch */
#define OV_ERR_OPDEFMISMATCH     0x00010026            /* operation def. mismatch */
#define OV_ERR_UNKNOWNSTRUCTDEF  0x00010027            /* unknown structure def. */
#define OV_ERR_UNKNOWNCLASSDEF   0x00010028            /* unknown class def. */
#define OV_ERR_UNKNOWNASSOCDEF   0x00010029            /* unknown association def. */

#define OV_ERR_DBOUTOFMEMORY     0x00010030            /* database is out of memory */
#define OV_ERR_HEAPOUTOFMEMORY   0x00010031            /* heap is out of memory */

typedef OV_ENUM OV_RESULT;
```

Remarks

Note, that the some enumeration values are identical to the corresponding ACPLT/KS values of the datatype `KS_RESULT`.

6.15 Type of an association

`OV_STATE` is an enumeration value defining the type of an ACPLT/OV association.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_AT_ORDERED_LIST 0x00000001 /* ordered list */

typedef OV_ENUM OV_ASSOC_TYPE;
```

6.16 Placement hint used with links

`OV_PLACEMENT_HINT` is an enumeration value defining a hint for the placement of a new member in a link when creating a new link between to objects.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_PMH_DEFAULT KS_PMH_DEFAULT /* default placement */
#define OV_PMH_BEGIN   KS_PMH_BEGIN   /* at the beginning */
#define OV_PMH_END     KS_PMH_END     /* at the end */
#define OV_PMH_BEFORE  KS_PMH_BEFORE  /* before a given object */
#define OV_PMH_AFTER   KS_PMH_AFTER   /* after a given object */

typedef OV_ENUM OV_PLACEMENT_HINT;
```

Remarks

Note, that the enumeration values are identical to the corresponding ACPLT/KS values of the datatype `KS_PLACEMENT_HINT`.

6.17 Access rights of an object

`OV_ACCESS` is an enumeration value defining the access rights of an ACPLT/OV object or any of its parts. The access rights also indicate if an object or part is a part of another object or part.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_AC_NONE         KS_AC_NONE         /* no access at all, element is not visible
#define OV_AC_READ         KS_AC_READ         /* read access */
#define OV_AC_WRITE        KS_AC_WRITE        /* write access */
#define OV_AC_READWRITE    KS_AC_READWRITE    /* both read and write access */
#define OV_AC_INSTANTIABLE KS_AC_INSTANTIABLE /* object can act as a factory */
#define OV_AC_PART         KS_AC_PART         /* object is part of another object */
#define OV_AC_DELETEABLE   0x00010000         /* object can be deleted */
#define OV_AC_LINKABLE     0x00010010         /* parent/child can be linked */
#define OV_AC_UNLINKABLE   0x00010011         /* parent/child can be unlinked */

typedef OV_ENUM OV_ACCESS;
```

Remarks

Note, that the some enumeration values are identical to the corresponding ACPLT/KS values of the datatype `KS_ACCESS`.

6.18 Type of an A/V-ticket

`OV_TICKET_TYPE` is an enumeration value defining the type of an ACPLT/KS authentification/verification ticket (compare ACPLT/KS documentation).

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_TT_NONE   KS_AUTH_NONE   /* no ticket */
#define OV_TT_SIMPLE KS_AUTH_SIMPLE /* a simple ticket */

typedef OV_ENUM OV_TICKET_TYPE;
```

Remarks

Note, that the enumeration values are identical to the corresponding ACPLT/KS values of the datatype `KS_AUTH_TYPE`.

6.19 Virtual function table associated with a ticket

`OV_TICKET_VTBL` is a virtual function table associated with an ACPLT/KS authentification/verification ticket of a given ticket type (compare ACPLT/KS documentation).

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_TICKET *(* createticket)(XDR *xdr, OV_TICKET_TYPE type);
    void      (* deleteticket)(OV_TICKET *pticket);
    OV_BOOL   (* encodereply)(XDR *xdr, OV_TICKET *pticket);
    OV_ACCESS (* getaccess)(const OV_TICKET *pticket);
}   OV_TICKET_VTBL;
```

Remarks

Please see the file `ov_server.c` for an example how to use virtual function tables of ACPLT/KS tickets in an ACPLT/OV server.

6.20 A/V-ticket

An A/V-ticket (authentification/verification ticket) is a structure containing the type of the ticket and, depending on this type, further parameters.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    struct OV_TICKET_VTBL *vtbl; /* not used when acting as a ACPLT/KS client */
    OV_TICKET_TYPE        type;
    union {
        struct {
            OV_STRING     id;    /* only, if tickettype == OV_TT_SIMPLE */
        }   simpleticket;
    }   ticketunion;
}   OV_TICKET;
```

Remarks

Compare the definition of A/V-tickets in ACPLT/KS.

6.21 Generic byte value

A byte value is a generic value which can store 8 bits.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef char OV_BYTE;
```

Remarks

Byte values are not used in ACPLT/OV; however, opaque values are expressed as byte vectors in ACPLT/KS. In ACPLT/OV this feature is used to allow to generically transport variable values over ACPLT/KS which are not supported by ACPLT/KS (e.g. user-defined datatypes).

Furthermore, sometimes it is necessary to calculate the difference between to pointers in bytes. As ANSI C does not define a difference of `void*` pointers, for this purpose `OV_BYTE*` pointers are used.

6.22 Generic enumeration value

A generic enumeration value can contain any possible value of an `enum`.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef enum_t OV_ENUM;
```

Remarks

The datatype `enum_t` is declared in the Sun ONC/RPC header `rpc/types.h` and falls back on the `int` datatype.

The reason for using this datatype in ACPLT/OV is that using this declaration we do not need any conversion when encoding/decoding an enumeration value in ACPLT/OV into/from an XDR stream.

6.23 Generic pointer value

A generic pointer value can contain any possible value of a pointer to any datatype.

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef void* OV_POINTER;
```

Remarks

Note, that ANSI C does not define a difference between `void*` pointers. Even if some compilers (e.g. the GNU compiler) allow to calculate a `void`-pointer difference, you should **never** do this with `OV_POINTER` pointers. Use `OV_BYTE*` pointers instead.

6.24 Dynamic bool value vector

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_UINT   veclen;
    OV_BOOL   *value;
}   OV_BOOL_VEC;
```

Remarks

6.25 Dynamic signed integer value vector

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_UINT   veclen;
    OV_INT    *value;
}   OV_INT_VEC;
```

Remarks

6.26 Dynamic unsigned integer value vector

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_UINT   veclen;
    OV_UINT   *value;
}   OV_UINT_VEC;
```

Remarks

6.27 Dynamic single precision floating point value vector

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_UINT     veclen;
    OV_SINGLE   *value;
}   OV_SINGLE_VEC;
```

Remarks

6.28 Dynamic double precision floating point value vector

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_UINT     veclen;
    OV_DOUBLE   *value;
}   OV_DOUBLE_VEC;
```

Remarks

6.29 Dynamic string value vector

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_UINT     veclen;
    OV_STRING   *value;
}   OV_STRING_VEC;
```

Remarks

6.30 Dynamic time/date value vector

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_UINT   veclen;
    OV_TIME   *value;
}   OV_TIME_VEC;
```

Remarks

6.31 Dynamic time span value vector

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_UINT        veclen;
    OV_TIME_SPAN   *value;
}   OV_TIME_SPAN_VEC;
```

Remarks

6.32 Generic dynamic value vector (internal use)

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_UINT      veclen;
    OV_POINTER   value;
}   OV_GENERIC_VEC;
```

Remarks

6.33 Boolean process value

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_BOOL    value;
    OV_STATE   state;
    OV_TIME    time;
}   OV_BOOL_PV;
```

Remarks

6.34 Integer process value

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_INT     value;
    OV_STATE   state;
    OV_TIME    time;
}   OV_INT_PV;
```

Remarks

6.35 Single precision floating point process value

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
typedef struct {
    OV_SINGLE   value;
    OV_STATE    state;
    OV_TIME     time;
}   OV_SINGLE_PV;
```

Remarks

6.36 Variable properties

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_VP_GETACCESSOR   0x00000001       /* variable has a get accessor */
#define OV_VP_SETACCESSOR   0x00000002       /* variable has a set accessor */
#define OV_VP_ACCESSORS     (OV_VP_GETACCESSOR | OV_VP_SETACCESSOR)
#define OV_VP_DERIVED       0x00000004       /* variable is derived (virtual) */
#define OV_VP_STATIC        0x00000008       /* variable is static (class variable) */

typedef OV_ENUM   OV_VAR_PROPS;
```

Remarks

6.37 Class properties

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_CP_INSTANTIABLE   0x00000001       /* class is instantiable */
#define OV_CP_FINAL          0x00000002       /* class cannot be subclassed */

typedef OV_ENUM   OV_CLASS_PROPS;
```

Remarks

6.38 Association properties

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_AP_LOCAL   0x00000001       /* association has local name scope */

typedef OV_ENUM   OV_ASSOC_PROPS;
```

Remarks

6.39 Operation properties

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_OP_ABSTRACT   0x00000001       /* operation is not implemented */

typedef OV_ENUM   OV_OP_PROPS;
```

Remarks

6.40 Time types for use with ACPLT/KS histories

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define KS_TT_ABSOLUTE   0x00000000       /* TODO! should be defined in ks.h... */
#define KS_TT_RELATIVE   0x00000001       /* ...but is currently defined in selector.h */

#define OV_TT_ABSOLUTE   KS_TT_ABSOLUTE
#define OV_TT_RELATIVE   KS_TT_RELATIVE

typedef OV_ENUM   OV_TIME_TYPE;
```

Remarks

6.41 Selector types for use with ACPLT/KS histories

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define KS_HSELT_NONE     0x00000000       /* TODO! should be defined in ks.h... */
#define KS_HSELT_TIME     0x00000001       /* ...but is currently defined in selector.h */
#define KS_HSELT_STRING   0x00000002       /* dto. */

#define OV_HSELT_NONE     KS_HSELT_NONE
#define OV_HSELT_TIME     KS_HSELT_TIME
#define OV_HSELT_STRING   KS_HSELT_STRING

typedef OV_ENUM   OV_HSEL_TYPE;
```

Remarks

6.42 History types for use with ACPLT/KS histories

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_HT_LOG             KS_HT_LOG
#define OV_HT_BOOL            KS_HT_BOOL
#define OV_HT_INT             KS_HT_INT
#define OV_HT_UINT            KS_HT_UINT
#define OV_HT_SINGLE          KS_HT_SINGLE
#define OV_HT_DOUBLE          KS_HT_DOUBLE
#define OV_HT_STRING          KS_HT_STRING
#define OV_HT_TIME            KS_HT_TIME
#define OV_HT_TIME_SPAN       KS_HT_TIME_SPAN
#define OV_HT_TYPE_MASK       KS_HT_TYPE_MASK
#define OV_HT_TIME_DRIVEN     KS_HT_TIME_DRIVEN
#define OV_HT_CHANGE_DRIVEN   KS_HT_CHANGE_DRIVEN

typedef OV_ENUM   OV_HIST_TYPE;
```

Remarks

6.43 Interpolation modes for use with ACPLT/KS histories

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_IPM_NONE      KS_IPM_NONE
#define OV_IPM_LINEAR    KS_IPM_LINEAR
#define OV_IPM_MIN       KS_IPM_MIN
#define OV_IPM_MAX       KS_IPM_MAX
#define OV_IPM_HOLD      KS_IPM_HOLD
#define OV_IPM_DEFAULT   KS_IPM_DEFAULT

typedef OV_ENUM   OV_INTERPOLATION_MODE;
```

Remarks

6.44 Type of a logfile message

Header file

```
#include "libov/ov_ov.h"
```

Declaration

```
#define OV_MT_UNKNOWN   0x00000000
#define OV_MT_INFO      0x00000001
#define OV_MT_DEBUG     0x00000002
#define OV_MT_WARNING   0x00000003
#define OV_MT_ERROR     0x00000004
#define OV_MT_ALERT     0x00000005

typedef OV_ENUM   OV_MSG_TYPE;
```

Remarks

7 Functions and macros associated with associations

An association defines a relationship between parent and child objects of given classes. It can be regarded as the class of all links of the same type and all links of this type as instances of that class.

This means, that the functionality for iterating over or manipulating links is associated with the association defining the link.

The macros and functions implementing this functionality are generic and **not type safe**. Whenever possible, you should use the macros defined in `libov/ov_macros.h` for improved type checking during compilation.

Macros for iterating over links of an association:

Functions associated with links of an association:

7.1 Get first child in a 1:n association

The `Ov_Association_GetFirstChild()` macro returns a pointer of the first child object of a link.

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pchild = Ov_Association_GetFirstChild(passoc, pparent);
```

Parameters

`passoc` Pointer to the association object defining the link.

`pparent` Pointer to the parent object of the link.

Return value

The macro returns a pointer to the first child object or `NULL` if no child objects exist.

Remarks

This generic macro is **not type safe**. Whenever possible, use the macro `Ov_GetFirstChild()` instead!

7.2 Get last child in a 1:n association

The `Ov_Association_GetLastChild()` macro returns a pointer of the last child object of a link.

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pchild = Ov_Association_GetLastChild(passoc, pparent);
```

Parameters

`passoc` Pointer to the association object defining the link.

`pparent` Pointer to the parent object of the link.

Return value

The macro returns a pointer to the last child object or `NULL` if no child objects exist.

Remarks

This generic macro is **not type safe**. Whenever possible, use the macro `Ov_GetLastChild()` instead!

7.3 Get next child in a 1:n association

The `Ov_Association_GetNextChild()` macro returns a pointer of the next child object of a link.

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pchild = Ov_Association_GetNextChild(passoc, pchild);
```

Parameters

`passoc` Pointer to the association object defining the link.

`pchild` Pointer to the current child object of the link.

Return value

The macro returns a pointer to the next child object or `NULL` if the current child object is the last object of the link.

Remarks

This generic macro is **not type safe**. Whenever possible, use the macro `Ov_GetNextChild()` instead!

7.4 Get previous child in a 1:n association

The `Ov_Association_GetPreviousChild()` macro returns a pointer of the previous child object of a link.

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pchild = Ov_Association_GetPreviousChild(passoc, pchild);
```

Parameters

`passoc` Pointer to the association object defining the link.

`pchild` Pointer to the current child object of the link.

Return value

The macro returns a pointer to the previous child object or `NULL` if the current child object is the first object of the link.

Remarks

This generic macro is **not type safe**. Whenever possible, use the macro `Ov_GetPrevChild()` instead!

7.5 Get parent in a 1:n association

The `Ov_Association_GetParent()` macro returns a pointer of the parent object of a link.

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pparent = Ov_Association_GetParent(passoc, pchild);
```

Parameters

`passoc` Pointer to the association object defining the link.

`pchild` Pointer to a child object of the link.

Return value

The macro returns a pointer to the parent object or `NULL` if child object does not have a parent object.

Remarks

This generic macro is **not type safe**. Whenever possible, use the macro `Ov_GetParent()` instead!

7.6 Iterate over all children in a 1:n association

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
Ov_Association_ForEachChild(passoc, pparent, pchild);
```

Parameters

`passoc`

`pparent`

`pchild`

Return value

Remarks

7.7 Define an iterator for iterating over n:m associations

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
Ov_Association_DefineIteratorNM(pit);
```

Parameters

`pit`

Return value

Remarks

7.8 Get first child in an n:m assocation

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pchild = Ov_Association_GetFirstChildNM(passoc, pit, pparent);
```

Parameters

`passoc`

`pit`

`pparent`

Return value

Remarks

7.9 Get last child in an n:m assocation

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pchild = Ov_Association_GetLastChildNM(passoc, pit, pparent);
```

Parameters

`passoc`

`pit`

`pparent`

Return value

Remarks

7.10 Get next child in an n:m association

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pchild = Ov_Association_GetNextChildNM(passoc, pit);
```

Parameters

`passoc`

`pit`

Return value

Remarks

7.11 Get previous child in an n:m association

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pchild = Ov_Association_GetPreviousChildNM(passoc, pit);
```

Parameters

`passoc`

`pit`

Return value

Remarks

7.12 Iterate over all children in an n:m association

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
Ov_Association_ForEachChildNM(passoc, pit, pparent, pchild);
```

Parameters

`passoc`

`pit`

`pparent`

`pchild`

Return value

Remarks

7.13 Get first parent in an n:m assocation

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pparent = Ov_Association_GetFirstParentNM(passoc, pit, pchild);
```

Parameters

`passoc`

`pit`

`pchild`

Return value

Remarks

7.14 Get last parent in an n:m assocation

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pparent = Ov_Association_GetLastParentNM(passoc, pit, pchild);
```

Parameters

`passoc`

`pit`

`pchild`

Return value

Remarks

7.15 Get next parent in an n:m association

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pparent = Ov_Association_GetNextParentNM(passoc, pit);
```

Parameters

`passoc`

`pit`

Return value

Remarks

7.16 Get previous parent in an n:m association

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
pparent = Ov_Association_GetPreviousParentNM(passoc, pit);
```

Parameters

`passoc`

`pit`

Return value

Remarks

7.17 Iterate over all parents in an n:m association

Header file

```
#include "libov/ov_association.h"
```

Macro usage

```
Ov_Association_ForEachParentNM(passoc, pit, pchild, pparent);
```

Parameters

`passoc`

`pit`

`pchild`

`pparent`

Return value

Remarks

7.18 Search for a child object with a given identifier in a 1:n association

The `ov_association_searchchild()` function returns a pointer to the child object of a namebinding link with a given identifier.

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_INSTPTR_ov_object ov_association_searchchild(
    const OV_INSTPTR_ov_association passoc,
    const OV_INSTPTR_ov_object      pparent,
    const OV_STRING                 identifier
);
```

Parameters

`passoc` Pointer to the association object defining the link. Note, that the association must be namebinding; a `NULL` pointer is returned otherwise.

`pparent` Pointer to the parent object of the link.

`identifier`
Pointer to a string which must match the identifier of the child object.

Return value

The function returns a pointer to the child object or `NULL` no child is found or the association is not namebinding.

Remarks

This generic function is **not type safe**. Whenever possible, use the macro `Ov_SearchChild()` instead!

7.19 Create a link between a child and a parent object

The `ov_association_link()` function creates a link between a child and a parent object.

The link can not be created, if the child object already has a parent object of the type given by the association or if the link is namebinding and a the child object has the same identifier as another child object of the parent object.

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_association_link(
    const OV_INSTPTR_ov_association passoc,
    const OV_INSTPTR_ov_object      pparent,
    const OV_INSTPTR_ov_object      pchild,
    const OV_PLACEMENT_HINT         hint,
    const OV_INSTPTR_ov_object      prelchild
);
```

Parameters

`passoc` Pointer to the association object defining the link.

`pparent` Pointer to the object which will be the parent object of the new link.

`pchild` Pointer to the object which will be the child object of the new link.

`hint` Placement hint indicating the position of the child object compared to other child objects of the parent of the new link.

`prelchild`
If `hint == OV_PMH_BEFORE` or `hint == OV_PMH_AFTER`, this parameter is a pointer to a child object before/after which the child object of the new link will be placed; otherwise this parameter is ignored. The child object pointed to by `prelchild` must alreay be a child of the parent object.

Return value

The function returns `OV_ERR_OK` if it succeeds or the error code otherwise.

Remarks

This generic function is **not type safe**. Whenever possible, use the macro `Ov_Link()` instead!

7.20 Remove a link between a child and a parent object

The `ov_association_unlink()` function removes a link between a child and a parent object.

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_association_unlink(
    const OV_INSTPTR_ov_association passoc,
    const OV_INSTPTR_ov_object      pparent,
    const OV_INSTPTR_ov_object      pchild
);
```

Parameters

`passoc` Pointer to the association object defining the link.

`pparent` Pointer to the parent object of the link to be removed.

`pchild` Pointer to the child object of the link to be removed.

Remarks

This generic function is **not type safe**. Whenever possible, use the macro `Ov_Unlink()` instead!

7.21 Get the number of child objects

The `ov_association_getchildcount()` function returns the number of child objects linked to a given parent object.

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_UINT ov_association_getchildcount(
    const OV_INSTPTR_ov_association passoc,
    const OV_INSTPTR_ov_object      pparent
);
```

Parameters

`passoc` Pointer to the association object defining the link.

`pparent` Pointer to the parent object.

Return value

The function returns the number of child objects linked to the parent object.

7.22 Test whether a head of a link is used

The `ov_association_isusedhead()` function tests whether a head of a link is used (i.e. the parent object owning the head has child objects) or not.

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_association_isusedhead(
    const OV_INSTPTR_ov_association passoc,
    const OV_INSTPTR_ov_object      pparent
);
```

Parameters

`passoc` Pointer to the association object defining the link.

`pparent` Pointer to the parent object owning the head of the link.

Return value

The function returns `TRUE` if the head is used and `FALSE` otherwise.

7.23 Test whether an anchor of a link is used

The `ov_association_isusedanchor()` function tests whether an anchor of a link is used (i.e. the child object owning the anchor has a parent object) or not.

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_association_isusedanchor(
    const OV_INSTPTR_ov_association passoc,
    const OV_INSTPTR_ov_object      pchild
);
```

Parameters

`passoc` Pointer to the association object defining the link.

`pchild` Pointer to the child object owning the anchor of the link.

Return value

The function returns `TRUE` if the anchor is used and `FALSE` otherwise.

7.24 Load an association into the database

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_RESULT ov_association_load(
    OV_ASSOCIATION_DEF     *passocdef,
    OV_INSTPTR_ov_domain   pparent
);
```

Parameters

`passocdef`

pparent

Return value

Remarks

7.25 Compare an association with its definition

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_RESULT ov_association_compare(
    OV_INSTPTR_ov_association   passoc,
    OV_ASSOCIATION_DEF          *passocdef
);
```

Parameters

passoc

passocdef

Return value

Remarks

7.26 Test if we can unload an association from the database

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_BOOL ov_association_canunload(
    OV_INSTPTR_ov_association   passoc
);
```

Parameters

passoc

Return value

Remarks

7.27 Get the number of parents of an association

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_UINT ov_association_getparentcount(
    const OV_INSTPTR_ov_association   passoc,
    const OV_INSTPTR_ov_object        pchild
);
```

Parameters

passoc

pchild

Return value

Remarks

7.28 Test if a parent link is used

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_association_isusedparentlink(
    const OV_INSTPTR_ov_association   passoc,
    const OV_INSTPTR_ov_object        pparent
);
```

Parameters

`passoc`

`pparent`

Return value

Remarks

7.29 Test if a child link is used

Header file

```
#include "libov/ov_association.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_association_isusedchildlink(
    const OV_INSTPTR_ov_association   passoc,
    const OV_INSTPTR_ov_object        pchild
);
```

Parameters

`passoc`

`pchild`

Return value

Remarks

8 Datatypes and functions associated with classes

A class is an abstraction of objects with the same data structure and the same operations; these objects are called instances of the class.

A class can be regarded as a factory responsible for creating and deleting instances.

Datatypes associated with classes:

Functions associated with classes:

8.1 Function prototype of an initialization function

The function prototype `OV_FNC_INITOBJ` is used for defining initialization functions, which are called during instantiation of an object using the function `ov_class_createobject()`.

Header file

```
#include "libov/ov_class.h"
```

Declaration

```
typedef OV_DLLFNCEXPORT OV_RESULT OV_FNC_INITOBJ(
    OV_INSTPTR_ov_object pobj,
    OV_POINTER           userdata
);
```

Parameters

`pobj`	Pointer to the instance object to be initialized during instantiation.
`userdata`	Generic pointer to a user defined data structure containing information which is necessary for initialization of the object to be initialized.

Return value

Functions of this type must return `OV_ERR_OK` if the initialization succeeds or an error code otherwise.

8.2 Create an instance of a class

The `ov_class_createobject()` function creates a new instance object of a class, initializes it and starts it up.

The instance can not be created if there is not enough memory available in the database, if the class is not instantiable, if an object with the given identifier already exists in the parent domain, if the initialization of the object fails or if the constructor of the object returns an error code other than `OV_ERR_OK`.

Header file

```
#include "libov/ov_class.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_class_createobject(
    const OV_INSTPTR_ov_class  pclass,
    const OV_INSTPTR_ov_domain pparent,
    const OV_STRING            identifier,
    const OV_PLACEMENT_HINT    hint,
    const OV_INSTPTR_ov_object prelchild,
    OV_FNC_INITOBJ             *initobjfnc,
    OV_POINTER                 userdata,
    OV_INSTPTR_ov_object       *ppobj
);
```

Parameters

`pclass` Pointer to the class object defining the class to be instantiated. The class must be instantiable.

`pparent` Pointer to the parent domain object which will contain the new instance object.

`identifier`
String containing the identifier of the new instance object.

`hint` Placement hint indicating the position of the new instance object compared to other objects contained in the parent domain object.

`prelchild`
If `hint == OV_PMH_BEFORE` or `hint == OV_PMH_AFTER`, this parameter is a pointer to a child object contained in the parent domain before/after which the new instance object will be placed; otherwise this parameter is ignored. The child object pointed to by `prelchild` must alreay be a child of the parent domain object.

`initobjfnc`
Pointer to a function with the function prototype `OV_FNC_INITOBJ`, which will be called for initialization of the new instance object or `NULL`. When this function is called during instantiation, the pointer to the instance object and a generic pointer to a user defined data structure is passed to the function. The function must then return a function result. If this result is not `OV_ERR_OK` instantiation of the object fails and the new instance object will be deleted.

`userdata` Generic pointer to a user defined data structure containing information which is necessary for initialization of the new instance object. This parameter is passed to the initialization function specified by `initobjfnc`.

`ppobj` Pointer to the instance object pointer of the new instance object.

Return value

The function returns `OV_ERR_OK` if it succeeds or the error code otherwise. If the function succeeds, the instance object pointer pointed to by the `ppobj` parameter contains the address of the new instance object or `NULL` otherwise.

8.3 Delete an instance of a class

The `ov_class_deleteobject()` function deletes an instance object. Before the instance object is actually deleted, it is shut down and its destructor is called.

An instance object can not be deleted if it is a part of another object.

Header file

```
#include "libov/ov_class.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_class_deleteobject(
    const OV_INSTPTR_ov_object pobj
);
```

Parameters

`pobj` Pointer to the instance object to be deleted.

Return value

The function returns `OV_ERR_OK` if it succeeds or the error code otherwise.

8.4 Test if a pointer cast of an instance pointer is allowed

The `ov_class_cancastto()` dynamically tests if a typecast from a pointer to an instance of one class, the "from" class, can be to a pointer to an instance of another class, the "to" class, is allowed or not. Such a typecast is allowed, if the "from" class is derived from the "to" class.

Header file

```
#include "libov/ov_class.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_class_cancastto(
    const OV_INSTPTR_ov_class pclassfrom,
    const OV_INSTPTR_ov_class pclassto
);
```

Parameters

`pclassfrom`
: Pointer to the class object defining the "from" class.

`pclassto`
: Pointer to the class object defining the "to" class.

Return value

The function returns `TRUE` if the typecast is allowed or `FALSE` otherwise.

8.5 Search for a class object with given identifier

Header file

```
#include "libov/ov_class.h"
```

Declaration

```
OV_INSTPTR_ov_class ov_class_search(
    OV_STRING   identifier
);
```

Parameters

`identifier`

Return value

Remarks

8.6 Load a class into the database

Header file

```
#include "libov/ov_class.h"
```

Declaration

```
OV_RESULT ov_class_load(
    OV_CLASS_DEF           *pclassdef,
    OV_INSTPTR_ov_domain   pparent
);
```

Parameters

`pclassdef`

`pparent`

Return value

Remarks

8.7 Compare a class with its definition

Header file

```
#include "libov/ov_class.h"
```

Declaration

```
OV_RESULT ov_class_compare(
    OV_INSTPTR_ov_class   pclass,
    OV_CLASS_DEF          *pclassdef
);
```

Parameters

`pclass`

`pclassdef`

Return value

Remarks

8.8 Test if we can unload a class from the database

Header file

```
#include "libov/ov_class.h"
```

Declaration

```
OV_BOOL ov_class_canunload(
    OV_INSTPTR_ov_class   pclass
);
```

Parameters

Return value

Remarks

8.9 Rename an instance of a class

Header file

```
#include "libov/ov_class.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_class_renameobject(
    const OV_INSTPTR_ov_object   pobj,
    const OV_INSTPTR_ov_domain   pparent,
    const OV_STRING              identifier,
    const OV_PLACEMENT_HINT      hint,
    const OV_INSTPTR_ov_object   prelobj
);
```

Parameters

`pobj`

`pparent`

`identifier`

`hint`

`prelobj`

Return value

Remarks

9 Functions associated with the database

ACPLT/OV applications always run on exactly one database. The database usually is a memory mapped file and stores the application's objects as well as the meta objects defined in the libraries of the application. The objects stored in the database are persistent, i.e. they survive if you stop the application process and restart it later.

Functions associated with the database file:

Functions for startup and shutdown of the objects:

Database memory management functions:

Database memory usage statistics functions:

9.1 Create a new database

The `ov_database_create()` function creates a new database file, maps the file into the application's address space and loads the OV library (the OV meta model).

The function can only succeed if no database is mapped.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_database_create(
    OV_STRING filename,
    OV_UINT   size
);
```

Parameters

`filename` The filename of the new database. The file must not yet exist.

`size` The initial size of the new database (in bytes). The actual database size is rounded up to a multiple of the memory page size.

Return value

The function returns `OV_ERR_OK` if it succeeds or the error code otherwise.

9.2 Map an existing database

The `ov_database_map()` function opens an existing database file and maps it into the application's address space.

The function can only succeed if no database is mapped and the database file is not used by another application.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_database_map(
    OV_STRING filename
);
```

Parameters

`filename` The filename of the existing database. The file must already exist and at least contain a consistent OV library.

Return value

The function returns `OV_ERR_OK` if it succeeds or the error code otherwise.

9.3 Unmap the mapped database

The `ov_database_unmap()` function unmaps the current database file from the application's address space and closes it.

The function can only succeed if no database is mapped.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_database_unmap(void);
```

Remarks

Before a database which was started up using `ov_database_startup()` is unmapped, it has to be shut down by calling `ov_database_shutdown()`.

If no database is mapped, the function has no effect.

9.4 Flush the contents of the mapped database

The `ov_database_flush()` function synchronously flushes the the contents of the database memory into the database file.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_database_flush(void);
```

Remarks

If no database is mapped, the function has no effect.

9.5 Start up the objects in the database

The `ov_database_startup()` function loads all libraries and starts up the objects stored in the mapped database by calling the root object's `startup` method.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_database_startup(void);
```

Return value

The function returns `OV_ERR_OK` if it succeeds or the error code otherwise.

Remarks

If no database is mapped, the function has no effect.

9.6 Shut down the objects in the database

The `ov_database_shutdown()` function shuts down the objects stored in the mapped database by calling the root object's `shutdown` method.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_database_shutdown(void);
```

Remarks

If no database is mapped, the function has no effect.

9.7 Allocate persistent database memory

Similar to the ANSI C `malloc()` function, `ov_database_malloc()` allocates a memory block of a given size and returns its pointer. While `malloc()` allocates the memory on the system heap, `ov_database_malloc()` allocates the memory in the persistent database.

If there's no memory block available in the database, the system tries to increase the database file and map the new part of the file to memory. If does not succeed, the function fails.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_POINTER ov_database_malloc(
    OV_UINT size
);
```

Parameters

`size` The amount of persistent memory to allocate in the database (in bytes).

Return value

The function returns a pointer to the allocated memory block if it succeeds or `NULL` otherwise.

Remarks

Even if `ov_database_getfree()` returns a value greater than `size` the function may return `NULL` if no block of the given size is available due to fragmentation.

If no database is mapped, the function always returns `NULL`.

9.8 Reallocate persistent database memory

Similar to the ANSI C `realloc()` function, `ov_database_realloc()` reallocates a memory block previously allocated using `ov_database_malloc()` or `ov_database_realloc()` of a given size and returns its pointer. While `realloc()` reallocates the memory on the system heap, `ov_database_realloc()` reallocates the memory in the persistent database.

If the new memory block is at least as large as the old block, the full contents of the old block is copied to the new block; if it is smaller than the old block, only the `size` first bytes from the old block are copied to the new block.

If there's no memory block available in the database, the system tries to increase the database file and map the new part of the file to memory. If does not succeed, the function fails.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_POINTER ov_database_realloc(
    OV_POINTER ptr,
    OV_UINT    size
);
```

Parameters

`ptr` Pointer to the memory block previously allocated in the database. If `ptr` is `NULL`, this function is equivalent to `ov_database_malloc(size)`.

`size` The amount of persistent memory to reallocate in the database (in bytes).

Return value

The function returns a pointer to the reallocated memory block if it succeeds or `NULL` otherwise.

Remarks

Even if `ov_database_getfree()` returns a value greater than `size` the function may return `NULL` if no block of the given size is available due to fragmentation.

If no database is mapped, the function always returns `NULL`.

9.9 Free persistent database memory

Similar to the ANSI C `free()` function, `ov_database_free()` frees a memory block previously allocated using `ov_database_malloc()` or `ov_database_realloc()`. While `free()` frees memory allocated on the system heap, `ov_database_free()` frees memory allocated in the persistent database.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_database_free(
    OV_POINTER ptr
);
```

Parameters

`ptr` Pointer to the memory block in the database that has to be freed.

Remarks

If no database is mapped, the function has no effect.

9.10 Get the total size of the database memory

The `ov_database_getsize()` function returns the total amount of memory the mapped database contains. Usually this size is equal the file size of the database file.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_UINT ov_database_getsize(void);
```

Return value

The function returns the total amount of memory in the mapped database (in bytes).

Remarks

If no database is mapped, the function returns `0`.

9.11 Get the size of the free database memory

The `ov_database_getfree()` function returns the amount of free (unused) memory the mapped database contains.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_UINT ov_database_getfree(void);
```

Return value

The function returns the amount of free memory in the mapped database (in bytes).

Remarks

If no database is mapped, the function returns `0`.

9.12 Get the size of the used database memory

The `ov_database_getused()` function returns the amount of used (allocated) memory the mapped database contains.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_UINT ov_database_getused(void);
```

Return value

The function returns the amount of used memory in the mapped database (in bytes).

Remarks

If no database is mapped, the function returns `0`.

9.13 Get the fragmentation of the database memory

The `ov_database_getfrag()` function returns a rough indication of the degree of fragmentation of the database memory in percent (0..100). If this value is low, chances are high that it is possible to allocate a new database memory block without increasing the database size.

Basically, the underlying algorithm determines the precentage of free memory that can not be used to allocate memory blocks of average object size.

Header file

```
#include "libov/ov_database.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_UINT ov_database_getfrag(void);
```

Return value

The function returns the degree of memory fragmentation of the mapped database (in percent).

Remarks

If no database is mapped, the function returns `100`.

10 Datatypes and functions associated with elements

An element is either an object or a part of an object, i.e. a variable, an end of a link (head or anchor), a part object, an operation or a member of a structured variable and can be identified by a unique path. Each element is either a child or a part of another element except for the element corresponding to the root object. This relationship is expressed in the path identifying the element, e.g. in `parent/child` the `child` element is a child of the `parent` element and in `composite.part` the `part` element is a part of the `composite` element.

Datatypes associated with elements:

Functions associated with elements:

10.1 Type of an element

Header file

```
#include "libov/ov_element.h"
```

Declaration

```
enum OV_ELEM_TYPE_ENUM {
    OV_ET_NONE          = 0x00,       /* invalid element */
    OV_ET_OBJECT        = 0x01,
    OV_ET_VARIABLE      = 0x02,
    OV_ET_MEMBER        = 0x04,
    OV_ET_PARENTLINK    = 0x08,
    OV_ET_CHILDLINK     = 0x10,
    OV_ET_OPERATION     = 0x20,
    OV_ET_ANY           = 0x3F        /* used for search masks only */
};

typedef enum_t   OV_ELEM_TYPE;
```

Remarks

10.2 Information associated with an element

Header file

```
#include "libov/ov_element.h"
```

Declaration

```
typedef struct {
    OV_ELEM_TYPE                        elemtype;       /* the type of the element */
    struct OV_INST_ov_object            *pobj;          /* object this element belongs to
    /* in case element is a variable or member of a variable: */
    OV_BYTE                             *pvalue;        /* pointer to variable value */
    /* in different cases: */
    union {
        /* generic definition object pointer */
        struct OV_INST_ov_object        *pobj;
        /* in case element is a variable or member of a variable: */
        struct OV_INST_ov_variable      *pvar;
        /* in case element is a part object: */
        struct OV_INST_ov_part          *ppart;
        /* in case element is a parent or child link: */
```

```
        struct OV_INST_ov_association   *passoc;
        /* in case element is an operation: */
        struct OV_INST_ov_operation     *pop;
        /* in case element is a class: */
        struct OV_INST_ov_class         *pclass;
    }   elemunion;
}   OV_ELEMENT;
```

Remarks

10.3 Search a child element of an element

Header file

```
#include "libov/ov_element.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_element_searchchild(
    const OV_ELEMENT   *pparent,
    OV_ELEMENT         *pchild,
    OV_STRING          identifier
);
```

Parameters

`pparent`

`pchild`

`identifier`

Return value

Remarks

10.4 Search a part element of an element

Header file

```
#include "libov/ov_element.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_element_searchpart(
    const OV_ELEMENT   *pparent,
    OV_ELEMENT         *ppart,
    OV_ELEM_TYPE       mask,
    OV_STRING          identifier
);
```

Parameters

`pparent`

`ppart`

`mask`

`identifier`

Return value

Remarks

10.5 Get next child element of an element

Header file

```
#include "libov/ov_element.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_element_getnextchild(
    const OV_ELEMENT   *pparent,
    OV_ELEMENT         *pchild
);
```

Parameters

`pparent`

`pchild`

Return value

Remarks

10.6 Get next part element of an element

Header file

```
#include "libov/ov_element.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_element_getnextpart(
    const OV_ELEMENT   *pparent,
    OV_ELEMENT         *ppart,
    OV_ELEM_TYPE       mask
);
```

Parameters

`pparent`

`ppart`

`mask`

Return value

Remarks

10.7 Get the identifier of an element

Header file

```
#include "libov/ov_element.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_STRING ov_element_getidentifier(
    const OV_ELEMENT   *pelem
);
```

Parameters

`pelem`

Return value

Remarks

11 Functions associated with libraries

11.1 Search for a library object with given identifier

Header file

```
#include "libov/ov_library.h"
```

Declaration

```
OV_INSTPTR_ov_library ov_library_search(
    OV_STRING   identifier
);
```

Parameters

`identifier`

Return value

Remarks

11.2 Open a library which is either a DLL/shared library or statically linked

Header file

```
#include "libov/ov_library.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_LIBRARY_DEF *ov_library_open(
    OV_INSTPTR_ov_library   plib
);
```

Parameters

`plib`

Return value

Remarks

11.3 Close a library file if it is a DLL/shared library

Header file

```
#include "libov/ov_library.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_library_close(
    OV_INSTPTR_ov_library   plib
);
```

Parameters

`plib`

Return value

Remarks

11.4 Load a library and its definitions into the database

Header file

```
#include "libov/ov_library.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_library_load(
    OV_INSTPTR_ov_library   plib,
    OV_LIBRARY_DEF          *plibdef
);
```

Parameters

`plib`

`plibdef`

Return value

Remarks

11.5 Compare a library with its definition

Header file

```
#include "libov/ov_library.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_library_compare(
    OV_INSTPTR_ov_library   plib,
    OV_LIBRARY_DEF          *plibdef
);
```

Parameters

`plib`

`plibdef`

Return value

Remarks

11.6 Test if we can unload a library and its definitions from the database

Header file

```
#include "libov/ov_library.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_library_canunload(
    OV_INSTPTR_ov_library   plib
);
```

Parameters

`plib`

Return value

Remarks

11.7 Get environment variable with library path

Header file

```
#include "libov/ov_library.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_STRING ov_library_getenv(void);
```

Parameters

none

Return value

Remarks

11.8 Set environment variable with library path

Header file

```
#include "libov/ov_library.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_library_setenv(
    OV_STRING   path
);
```

Parameters

`path`

Return value

Remarks

12 Functions associated with logfiles

12.1 Open/create a logfile

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_logfile_open(
    const OV_STRING   ident,
    OV_STRING         filename,
    OV_STRING         mode
);
```

Parameters

`ident`

`filename`

`mode`

Return value

Remarks

12.2 Close the logfile

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_logfile_close(void);
```

Parameters

none

Return value

Remarks

12.3 Log to stdout

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_logfile_logtostdout(
    const OV_STRING   ident
);
```

Parameters

`ident`

Return value

Remarks

12.4 Log to stderr

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_logfile_logtostderr(
    const OV_STRING   ident
);
```

Parameters

`ident`

Return value

Remarks

12.5 Log to the NT logger (Windows NT only)

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
#if OV_SYSTEM_NT

OV_DLLFNCEXPORT void ov_logfile_logtontlog(
    const OV_STRING   ident
);

#endif
```

Parameters

`ident`

Return value

Remarks

12.6 Print text to logfile

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_logfile_print(
    OV_MSG_TYPE       msgtype,
    const OV_STRING   msg
);
```

Parameters

`msgtype`

`msg`

Return value

Remarks

12.7 Print info to logfile

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_logfile_info(
    const OV_STRING   format,
    ...
);
```

Parameters

`format`

`...`

Return value

Remarks

12.8 Print debug info to logfile

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_logfile_debug(
    const OV_STRING   format,
    ...
);
```

Parameters

`format`

`...`

Return value

Remarks

12.9 Print warning to logfile

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_logfile_warning(
    const OV_STRING   format,
    ...
);
```

Parameters

`format`

`...`

Return value

Remarks

12.10 Print error to logfile

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_logfile_error(
    const OV_STRING   format,
    ...
);
```

Parameters

`format`

`...`

Return value

Remarks

12.11 Print alert to logfile

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_logfile_alert(
    const OV_STRING   format,
    ...
);
```

Parameters

`format`

`...`

Return value

Remarks

12.12 Get messages from the logfile

Header file

```
#include "libov/ov_logfile.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_logfile_getmessages(
    OV_TIME     *from,
    OV_TIME     *to,
    OV_UINT     max_no_messages,
    OV_STRING   **messages,
    OV_TIME     **times,
    OV_UINT     *no_messages
);
```

Parameters

`from`

`to`

`max_no_messages`

`messages`

`times`

`no_messages`

Return value

Remarks

You must call `ov_memstack_lock()` and `ov_memstack_unlock()` outside.

13 Memory management functions for the system heap memory

13.1 Allocate memory on the heap

Header file

```
#include "libov/ov_malloc.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_POINTER ov_malloc(
    OV_UINT   size
);
```

Parameters

`size`

Return value

Remarks

13.2 Free memory allocated in the heap

Header file

```
#include "libov/ov_malloc.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_free(
    OV_POINTER   ptr
);
```

Parameters

`ptr`

Return value

Remarks

13.3 Reallocate memory on the heap

Header file

```
#include "libov/ov_malloc.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_POINTER ov_realloc(
    OV_POINTER   ptr,
    OV_UINT      size
);
```

Parameters

`ptr`

`size`

Return value

Remarks

13.4 Duplicate a string on the heap using malloc

Header file

```
#include "libov/ov_malloc.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_STRING ov_strdup(
    OV_STRING   string
);
```

Parameters

`string`

Return value

Remarks

14 Memory management functions for the memory stack

14.1 Increment the reference count of the stack and initialize if necessary

Header file

```
#include "libov/ov_memstack.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_memstack_lock(void);
```

Parameters

none

Return value

Remarks

14.2 Allocate memory on the stack

Header file

```
#include "libov/ov_memstack.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_POINTER ov_memstack_alloc(
    OV_UINT   size
);
```

Parameters

`size`

Return value

Remarks

14.3 Decrement the reference count of the stack and free the stack memory if necessary

Header file

```
#include "libov/ov_memstack.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_memstack_unlock(void);
```

Parameters

none

Return value

Remarks

15 Datatypes and functions associated with objects (top level class)

Datatypes associated with objects:

Functions associated with objects:

15.1 Function prototype for constructor of an object

The function prototype `OV_FNC_CONSTRUCTOR` is used for defining the constructor function of an object, which is called during instantiation of an object using the function `ov_class_createobject()`. If the function does not return `OV_ERR_OK` the object is destroyed immediately and no object is created.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT OV_RESULT OV_FNC_CONSTRUCTOR(
    OV_INSTPTR_ov_object pobj
);
```

Parameters

`pobj` Pointer to the object to be constructed.

Return value

Functions of this type must return `OV_ERR_OK` if the construction succeeds or an error code otherwise.

15.2 Function prototype for checking the initialization

The function prototype `OV_FNC_CHECKINIT` is used for defining a function of an object, which checks if the initialization resulted in a valid state of the object. This function is called during instantiation of an object using the function `ov_class_createobject()`. If the function does not return `OV_ERR_OK` the object is destroyed immediately and no object is created.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT OV_RESULT OV_FNC_CHECKINIT(
    OV_INSTPTR_ov_object pobj
);
```

Parameters

`pobj` Pointer to the object of which the initialization is checked.

Return value

Functions of this type must return `OV_ERR_OK` if the initialzation was valid or an error code otherwise.

15.3 Function prototype for destructor of an object

The function prototype `OV_FNC_DESTRUCTOR` is used for defining the destructor function of an object, which is called during deletion of an object using the function `ov_class_deleteobject()`.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT void OV_FNC_DESTRUCTOR(
    OV_INSTPTR_ov_object pobj
);
```

Parameters

`pobj` Pointer to the object to be destructed.

Return value

none.

15.4 Function prototype for method starting up an object

The function prototype `OV_FNC_STARTUP` is used for defining the function of an object, which is called when the object is started up, either at system startup or immediately after construction (instantiation) of the object.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT void OV_FNC_STARTUP(
    OV_INSTPTR_ov_object pobj
);
```

Parameters

`pobj` Pointer to the object to be started up.

Return value

none.

15.5 Function prototype for method shutting down an object

The function prototype `OV_FNC_SHUTDOWN` is used for defining the function of an object, which is called when the object is shut down, either at system shutdown or immediately before destruction (deletion) of the object.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT void OV_FNC_SHUTDOWN(
    OV_INSTPTR_ov_object pobj
);
```

Parameters

`pobj` Pointer to the object to be shut down.

Return value

none.

15.6 Function prototype for method reading access rights

The function prototype `OV_FNC_GETACCESS` is used for defining the function of an object, which returns the access rights of an object or one of its elements.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT OV_ACCESS OV_FNC_GETACCESS(
    OV_INSTPTR_ov_object pobj,
    const OV_ELEMENT     *pelem,
    const OV_TICKET      *pticket
);
```

Parameters

`pobj` Pointer to the object to be asked about the access rights.

`pelem` Pointer to a structure containing information about the element of the object the access rights are read from.

`pticket` Pointer to the ticket structure providing authentification/verification information. The information included in this structure may be used to decide which access rights are granted to the specified element of the object.

Return value

Access rights of the object or object element respectively.

15.7 Function prototype for method reading semantical flags

The function prototype `OV_FNC_GETFLAGS` is used for defining the function of an object, which returns the semantic flags of an object or one of its elements.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT OV_UINT OV_FNC_GETFLAGS(
    OV_INSTPTR_ov_object pobj,
    const OV_ELEMENT     *pelem
);
```

Parameters

`pobj` Pointer to the object to be asked about the semantic flags.

`pelem` Pointer to a structure containing information about the element of the object the semantic flags are read from.

Return value

Semantic flags of the object or object element respectively.

15.8 Function prototype for method reading comments

The function prototype `OV_FNC_GETCOMMENT` is used for defining the function of an object, which returns the comment associated with an object or one of its elements.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT OV_STRING OV_FNC_GETCOMMENT(
    OV_INSTPTR_ov_object pobj,
    const OV_ELEMENT     *pelem
);
```

Parameters

`pobj` Pointer to the object to be asked about the associated comment.

`pelem` Pointer to a structure containing information about the element of the object the comment to be read is associated with.

Return value

Comment associated with the object or object element respectively.

15.9 Function prototype for method reading variable units

The function prototype `OV_FNC_GETACCESS` is used for defining the function of an object, which returns the technical unit of a variable element of the object.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT OV_STRING OV_FNC_GETTECHUNIT(
    OV_INSTPTR_ov_object pobj,
    const OV_ELEMENT     *pelem
);
```

Parameters

`pobj` Pointer to the object to be asked about the technical unit.

`pelem` Pointer to a structure containing information about the variable element of the object the comment is read from.

Return value

Technical unit associated with the variable element of the object.

15.10 Function prototype for method reading current variable properties

The function prototype `OV_FNC_GETVAR` is used for defining the function of an object, which reads the current properties of a variable of the object.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT OV_RESULT OV_FNC_GETVAR(
    OV_INSTPTR_ov_object pobj,
    const OV_ELEMENT     *pelem,
    OV_ANY               *pvarcurrprops
);
```

Parameters

`pobj` Pointer to the object containing the variable element to be read.

`pelem` Pointer to a structure containing information about the variable element of the object the current properties are read from.

`pvarcurrprops`
Pointer to a structure containing the current properties of the variable. This structure has to be filled by the function when reading the current properties succeeds.

Return value

Functions of this type must return `OV_ERR_OK` if reading the current properties of the variable succeeds or an error code otherwise.

15.11 Function prototype for method writing current variable properties

The function prototype `OV_FNC_SETVAR` is used for defining the function of an object, which sets the current properties of a variable of the object.

Header file

```
#include "ov.ovf"
```

Declaration

```
typedef OV_DLLFNCEXPORT OV_RESULT OV_FNC_SETVAR(
    OV_INSTPTR_ov_object pobj,
    const OV_ELEMENT     *pelem
    const OV_ANY         *pvarcurrprops
);
```

Parameters

`pobj` Pointer to the object containing the variable element to be set.

`pelem` Pointer to a structure containing information about the variable element of the object of which the current properties are set.

`pvarcurrprops`
Pointer to a structure containing the current properties of the variable which the object is asked to set.

Return value

Functions of this type must return `OV_ERR_OK` if setting the current properties of the variable succeeds or an error code otherwise. You should use `OV_ERR_BADVALUE` to indicate that the variable value is not accepted (e.g. out of bounds) and `OV_ERR_BADTYPE` to indicate that the given variable type is not accepted by the object.

15.12 Test, if an object owns links (except for the parent domain and class)

Header file

```
#include "libov/ov_object.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_object_haslinks(
    OV_INSTPTR_ov_object   pobj
);
```

Parameters

`pobj`

Return value

Remarks

15.13 Test, if a string is a valid identifier for an object

Header file

```
#include "libov/ov_object.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_object_identifierok(
    OV_STRING   identifier
);
```

Parameters

`identifier`

Return value

Remarks

16 Functions associated with operations

16.1 Load an operation into the database

Header file

```
#include "libov/ov_operation.h"
```

Declaration

```
OV_RESULT ov_operation_load(
    OV_OPERATION_DEF       *popdef,
    OV_INSTPTR_ov_domain   pparent
);
```

Parameters

`popdef`

`pparent`

Return value

Remarks

16.2 Compare an operation with its definition

Header file

```
#include "libov/ov_operation.h"
```

Declaration

```
OV_RESULT ov_operation_compare(
    OV_INSTPTR_ov_operation   pop,
    OV_OPERATION_DEF          *popdef
);
```

Parameters

`pop`

`popdef`

Return value

Remarks

16.3 Test if we can unload an operation from the database

Header file

```
#include "libov/ov_operation.h"
```

Declaration

```
OV_BOOL ov_operation_canunload(
    OV_INSTPTR_ov_operation   pop
);
```

Parameters

`pop`

Return value

Remarks

17 Functions associated with parts

17.1 Load a part into the database

Header file

```
#include "libov/ov_part.h"
```

Declaration

```
OV_RESULT ov_part_load(
    OV_PART_DEF            *ppartdef,
    OV_INSTPTR_ov_domain   pparent
);
```

Parameters

ppartdef

pparent

Return value

Remarks

17.2 Compare a part with its definition

Header file

```
#include "libov/ov_part.h"
```

Declaration

```
OV_RESULT ov_part_compare(
    OV_INSTPTR_ov_part   ppart,
    OV_PART_DEF          *ppartdef
);
```

Parameters

ppart

ppartdef

Return value

Remarks

17.3 Test if we can unload a part from the database

Header file

```
#include "libov/ov_part.h"
```

Declaration

```
OV_BOOL ov_part_canunload(
    OV_INSTPTR_ov_part   ppart
);
```

Parameters

ppart

Return value

Remarks

18 Datatypes and functions associated with paths

Datatypes associated with paths:

Functions associated with paths:

18.1 Array of elements corresponding to the identifiers of a path name

Header file

```
#include "libov/ov_path.h"
```

Declaration

```
typedef struct {
    OV_UINT      size;            /* number of path elements */
    OV_ELEMENT   *elements;       /* pointer to array of elements */
}   OV_PATH;
```

Remarks

18.2 Resolve a path using a given path name

Header file

```
#include "libov/ov_path.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_path_resolve(
    OV_PATH           *ppath,
    const OV_PATH     *prelpath,
    const OV_STRING   pathname,
    const OV_UINT     version
);
```

Parameters

`ppath`

`prelpath`

`pathname`

`version`

Return value

Remarks

The memory for the path elements is allocated on the memory stack, use `ov_memstack_lock()`/`unlock()` outside of this function.

18.3 Get the canonical path of an element

Header file

```
#include "libov/ov_path.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_STRING ov_path_getcanonicalpath(
    OV_INSTPTR_ov_object   pobj,
    const OV_UINT          version
);
```

Parameters

`pobj`

`version`

Return value

Remarks

The memory for the path name is allocated on the memory stack, use `ov_memstack_lock()`/`unlock()`outside of this function.

18.4 Get the pointer to an object with given path name

Header file

```
#include "libov/ov_path.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_INSTPTR_ov_object ov_path_getobjectpointer(
    const OV_STRING   pathname,
    const OV_UINT     version
);
```

Parameters

`pathname`

`version`

Return value

Remarks

You need *not* call `ov_memstack_lock()`/`unlock()` outside of this function.

19 Functions associated with results of function calls

19.1 Return error string associated with an error code

Header file

```
#include "libov/ov_result.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_STRING ov_result_getresulttext(
    OV_RESULT   result
);
```

Parameters

`result`

Return value

Remarks

20 Datatypes and functions associated with the scheduler

Datatypes associated with the scheduler:
Functions associated with the scheduler:

20.1 Function prototype for methods used in active objects

Header file

```
#include "libov/ov_scheduler.h"
```

Declaration

```
typedef OV_DLLFNCEXPORT void OV_FNC_EXECUTE(
    OV_INSTPTR_ov_object   pobj
);
```

Parameters

`pobj`

Return value

Remarks

20.2 Event in a simple event queue, ordered by time

Header file

```
#include "libov/ov_scheduler.h"
```

Declaration

```
struct OV_SCHEDULER_EVENT {
    struct OV_SCHEDULER_EVENT   *pnext;            /* Pointer to next scheduled event */
    OV_INSTPTR_ov_object        pobj;              /* Pointer to scheduled active object *
    OV_FNC_EXECUTE              *executefnc;       /* method to call on active object */
    OV_TIME                     time;              /* time of this scheduled event */
};

typedef struct OV_SCHEDULER_EVENT   OV_SCHEDULER_EVENT;
```

Remarks

20.3 Register an active object with the scheduler

Header file

```
#include "libov/ov_scheduler.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_scheduler_register(
    OV_INSTPTR_ov_object   pobj,
    OV_FNC_EXECUTE         *executefnc
);
```

Parameters

`pobj`

`executefnc`

Return value

Remarks

20.4 Unregister an active object with the scheduler

Header file

```
#include "libov/ov_scheduler.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_scheduler_unregister(
    OV_INSTPTR_ov_object   pobj
);
```

Parameters

`pobj`

Return value

Remarks

20.5 Set absolute event time of a registered active object

Header file

```
#include "libov/ov_scheduler.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_scheduler_setabseventtime(
    OV_INSTPTR_ov_object   pobj,
    OV_TIME                *ptime
);
```

Parameters

`pobj`

`ptime`

Return value

Remarks

20.6 Set relative event time of a registered active object (time span from now on)

Header file

```
#include "libov/ov_scheduler.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_scheduler_setreleventtime(
    OV_INSTPTR_ov_object   pobj,
    OV_TIME_SPAN           *ptimespan
);
```

Parameters

`pobj`

`ptimespan`

Return value

Remarks

20.7 Schedule the next event of the event queue if the event is pending

Header file

```
#include "libov/ov_scheduler.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_TIME_SPAN *ov_scheduler_schedulenextevent(void);
```

Parameters

none

Return value

Remarks

21 Functions associated with string variables

21.1 Set value of a string in the database

Header file

```
#include "libov/ov_string.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_string_setvalue(
    OV_STRING         *pstring,
    const OV_STRING   value
);
```

Parameters

`pstring`

`value`

Return value

Remarks

21.2 Set value of a string vector in the database

Header file

```
#include "libov/ov_string.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_string_setvecvalue(
    OV_STRING         *pstringvec,
    const OV_STRING   *pvalue,
    OV_UINT           veclen
);
```

Parameters

`pstringvec`

`pvalue`

`veclen`

Return value

Remarks

21.3 Compare two strings, result is greater than, equal to or less than zero

Header file

```
#include "libov/ov_string.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_INT ov_string_compare(
    const OV_STRING   string1,
    const OV_STRING   string2
);
```

Parameters

`string1`

`string2`

Return value

Remarks

21.4 Get the length of a string

Header file

```
#include "libov/ov_string.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_UINT ov_string_getlength(
    const OV_STRING   string
);
```

Parameters

`string`

Return value

Remarks

21.5 Append a string to an existing one

Header file

```
#include "libov/ov_string.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_string_append(
    OV_STRING         *pstring,
    const OV_STRING   appstring
);
```

Parameters

`pstring`

`appstring`

Return value

Remarks

21.6 Formatted print to a string

Header file

```
#include "libov/ov_string.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_string_print(
    OV_STRING         *pstring,
    const OV_STRING   format,
    ...
);
```

Parameters

`pstring`

`format`

`...`

Return value

Remarks

21.7 Test if a string matches a regular expression

Header file

```
#include "libov/ov_string.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_string_match(
    const OV_STRING   string,
    const OV_STRING   mask
);
```

Parameters

`string`

`mask`

Return value

Remarks

21.8 Convert a string to lower case

Header file

```
#include "libov/ov_string.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_STRING ov_string_tolower(
    const OV_STRING   string
);
```

Parameters

`string`

Return value

Remarks

You must call `ov_memstack_lock/unlock()` outside of this function!

21.9 Convert a string to upper case

Header file

```
#include "libov/ov_string.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_STRING ov_string_toupper(
   const OV_STRING   string
);
```

Parameters

`string`

Return value

Remarks

You must call `ov_memstack_lock/unlock()` outside of this function!

22 Functions associated with Structures

22.1 Search for a structure object with given identifier

Header file

```
#include "libov/ov_structure.h"
```

Declaration

```
OV_INSTPTR_ov_structure ov_structure_search(
    OV_STRING   identifier
);
```

Parameters

`identifier`

Return value

Remarks

22.2 Load a structure into the database

Header file

```
#include "libov/ov_structure.h"
```

Declaration

```
OV_RESULT ov_structure_load(
    OV_STRUCTURE_DEF       *pstructdef,
    OV_INSTPTR_ov_domain   pparent
);
```

Parameters

`pstructdef`

`pparent`

Return value

Remarks

22.3 Compare a structure with its definition

Header file

```
#include "libov/ov_structure.h"
```

Declaration

```
OV_RESULT ov_structure_compare(
    OV_INSTPTR_ov_structure   pstruct,
    OV_STRUCTURE_DEF          *pstructdef
);
```

Parameters

`pstruct`

`pstructdef`

Return value

Remarks

22.4 Test if we can unload a structure from the database

Header file

```
#include "libov/ov_structure.h"
```

Declaration

```
OV_BOOL ov_structure_canunload(
    OV_INSTPTR_ov_structure   pstruct
);
```

Parameters

`pstruct`

Return value

Remarks

23 Execution time supervision functions

The purpose of these functions is to supervise the duration of execution of methods implemented by the user. Using this mechanism, the duration of execution of a certain function is limited to a certain value, which may be freely chosen. Before the function is actually called, a timer is started with the duration limit as timeout. Then the function is called. If the timer does not time out during the function call, the timer will be stopped and nothing else happens. But if there is a timeout, the function will be aborted by brute force and the function will return immediately. Note that this may have side effects!

Attention:

- The supervision code is *not reentrant*. If you call `ov_supervise_start()` while the supervision mechanism is already running, you will get an error.
- Under certain circumstances variables of the function calling the supervision functions have to be declared as volatile, because they may be optimized away when the `longjmp()` function is called.
- Under Windows NT each call of the supervise routines has to come from the *same thread*.

Resources used:

- Unix: `ITIMER_REAL` and `SIGALRM`
- NT: an invisible timer window

Side effects: Currently unknown – it seems to work, but you should *not rely on it*.

Datatypes:

Functions:

23.1 Example code for the execution time supervision functions

```
#include "libov/ov_supervise.h"

OV_TIME_SPAN    timeout;
OV_JUMPBUFFER   jumpbuffer;
timeout.secs = 1;
timeout.usecs = 500000;
if(ov_supervise_setjmp(jumpbuffer) == 0) {
    if(!ov_supervise_start(&timeout, &jumpbuffer)) {
        /* error in ov_supervise_start(), e.g. could not start timer */
        printf("error\n");
    }
    userfunction(); /* call of the user's function */
    if(!ov_supervise_end()) {
        /* error in ov_supervise_end() */
        printf("error\n");
    }
} else {
    /* timer timed out, here's code handling this error */
    printf("userfunction() aborted\n");
}
```

23.2 Datatype describing the stack frame before calling the user function

This datatype describes the stack frame before calling the user function and allows to abort it.

Header file

```
#include "libov/ov_supervise.h"
```

Declaration

```
#if OV_SYSTEM_UNIX
typedef sigjmp_buf   OV_JUMPBUFFER;
#else
typedef jmp_buf      OV_JUMPBUFFER;
#endif
```

Remarks

23.3 Wrapper macro for the setjmp function/macro

Header file

```
#include "libov/ov_supervise.h"
```

Macro usage

```
ov_supervise_setjmp(jumpbuffer);
```

Parameters

`jumpbuffer`

Return value

Remarks

23.4 Start the supervision of a user function

Header file

```
#include "libov/ov_supervise.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_supervise_start(
    OV_TIME_SPAN    *ptimeout,
    OV_JUMPBUFFER   *pjumpbuffer
);
```

Parameters

`ptimeout`

`pjumpbuffer`

Return value

Remarks

23.5 Finish supervising a user function

Header file

```
#include "libov/ov_supervise.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_BOOL ov_supervise_end(void);
```

Parameters

none

Return value

Remarks

24 Functions associated with time variables

24.1 Get the current system time

Header file

```
#include "libov/ov_time.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_time_gettime(
    OV_TIME   *ptime
);
```

Parameters

ptime

Return value

Remarks

24.2 Calculate the sum of a time and a time span

Header file

```
#include "libov/ov_time.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_time_add(
    OV_TIME              *psum,
    const OV_TIME        *padd1,
    const OV_TIME_SPAN   *padd2
);
```

Parameters

psum

padd1

padd2

Return value

Remarks

24.3 Calculate the difference of two times

Header file

```
#include "libov/ov_time.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_time_diff(
    OV_TIME_SPAN    *pdiff,
    const OV_TIME   *psub1,
    const OV_TIME   *psub2
);
```

Parameters

pdiff

psub1

psub2

Return value

Remarks

24.4 Compare two times, result is -1, 0 or 1

Header file

```
#include "libov/ov_time.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_INT ov_time_compare(
    const OV_TIME   *ptime1,
    const OV_TIME   *ptime2
);
```

Parameters

ptime1

ptime2

Return value

Remarks

24.5 Convert a time into an ASCII string

Header file

```
#include "libov/ov_time.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_STRING ov_time_timetoascii(
    const OV_TIME   *ptime
);
```

Parameters

ptime

Return value

Remarks

24.6 Convert an ASCII string into a time

Header file

```
#include "libov/ov_time.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_time_asciitotime(
    OV_TIME           *ptime,
    const OV_STRING   timestring
);
```

Parameters

ptime

timestring

Return value

Remarks

25 Functions associated with variables

25.1 Load a variable into the database

Header file

```
#include "libov/ov_variable.h"
```

Declaration

```
OV_RESULT ov_variable_load(
    OV_VARIABLE_DEF        *pvardef,
    OV_INSTPTR_ov_domain   pparent
);
```

Parameters

`pvardef`

`pparent`

Return value

Remarks

25.2 Compare a variable with its definition

Header file

```
#include "libov/ov_variable.h"
```

Declaration

```
OV_RESULT ov_variable_compare(
    OV_INSTPTR_ov_variable   pvar,
    OV_VARIABLE_DEF          *pvardef
);
```

Parameters

`pvar`

`pvardef`

Return value

Remarks

25.3 Test if we can unload a variable from the database

Header file

```
#include "libov/ov_variable.h"
```

Declaration

```
OV_BOOL ov_variable_canunload(
    OV_INSTPTR_ov_variable   pvar
);
```

Parameters

`pvar`

Return value

Remarks

26 Functions associated with vector variables

26.1 Set the value of a static vector variable

Header file

```
#include "libov/ov_vector.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vector_setstaticvalue(
    OV_POINTER          pvector,
    const OV_POINTER    pvalue,
    const OV_UINT       veclen,
    const OV_UINT       size,
    const OV_VAR_TYPE   vartype
);
```

Parameters

`pvector`

`pvalue`

`veclen`

`size`

`vartype`

Return value

Remarks

26.2 Set the value of a dynamic vector variable

Header file

```
#include "libov/ov_vector.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vector_setdynamicvalue(
    OV_GENERIC_VEC      *pvector,
    const OV_POINTER    pvalue,
    const OV_UINT       veclen,
    const OV_UINT       size,
    const OV_VAR_TYPE   vartype
);
```

Parameters

`pvector`

`pvalue`

`veclen`

`size`

`vartype`

Return value

Remarks

26.3 Set the vector length of a dynamic vector variable value

Header file

```
#include "libov/ov_vector.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vector_setdynamicveclen(
    OV_GENERIC_VEC      *pvector,
    const OV_UINT       veclen,
    const OV_UINT       size,
    const OV_VAR_TYPE   vartype
);
```

Parameters

`pvector`

`veclen`

`size`

`vartype`

Return value

Remarks

26.4 Compare two vector variable values, result is greater than, equal to or less than zero

Header file

```
#include "libov/ov_vector.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_INT ov_vector_compare(
    const OV_POINTER    pvalue1,
    const OV_POINTER    pvalue2,
    const OV_UINT       veclen,
    const OV_UINT       size,
    const OV_VAR_TYPE   vartype
);
```

Parameters

`pvalue1`

`pvalue2`

`veclen`

`size`

`vartype`

Return value

Remarks

27 Datatypes and functions associated with the vendor tree

Datatypes associated with the vendor tree:
Functions associated with the vendor tree:

27.1 Function prototype for getting vendor variables

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
typedef OV_DLLFNCEXPORT OV_RESULT OV_FNC_GETVENDORVAR(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.2 Information of a vendor tree object

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
typedef struct {
    OV_STRING              identifier;
    OV_STRING              unit;
    OV_FNC_GETVENDORVAR    *getvarfnc;
}   OV_VENDORTREE_INFO;
```

Remarks

27.3 Initialize the vendor tree

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
void ov_vendortree_init(void);
```

Parameters

none

Return value

Remarks

27.4 Get unit of a vendor object

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_STRING ov_vendortree_getunit(
    OV_INSTPTR_ov_object   pobj
);
```

Parameters

pobj

Return value

Remarks

27.5 Get variable of a vendor object

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getvar(
    OV_INSTPTR_ov_object   pobj,
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

pobj

pvarcurrprops

pticket

Return value

Remarks

27.6 Set database name

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_vendortree_setdatabasename(
    OV_STRING   name
);
```

Parameters

name

Return value

Remarks

27.7 Set vendor name

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_vendortree_setname(
    OV_STRING   name
);
```

Parameters

`name`

Return value

Remarks

27.8 Set semantic flag

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_vendortree_setsemanticflag(
    OV_UINT     flagnum,
    OV_STRING   flagvalue
);
```

Parameters

`flagnum`

`flagvalue`

Return value

Remarks

27.9 Set server name

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_vendortree_setservername(
    OV_STRING   name
);
```

Parameters

`name`

Return value

Remarks

27.10 Set server description

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_vendortree_setserverdescription(
    OV_STRING   name
);
```

Parameters

name

Return value

Remarks

27.11 Set server version

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_vendortree_setserverversion(
    OV_STRING   name
);
```

Parameters

name

Return value

Remarks

27.12 Set startup time

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT void ov_vendortree_setstartuptime(
    OV_TIME   *ptime
);
```

Parameters

ptime

Return value

Remarks

27.13 Get list of associations in the database

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getassociations(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.14 Get list of classes in the database

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getclasses(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.15 Get fragmentation of the database

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getdatabasefrag(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.16 Get free storage of the database

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getdatabasefree(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.17 Get database name

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getdatabasename(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.18 Get size of the database

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getdatabasesize(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.19 Get whether the database is started or not

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getdatabasestarted(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.20 Get used storage of the database

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getdatabaseused(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.21 Get vendor name

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getname(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.22 Get LibKS version

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getlibksversion(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.23 Get LibOV version

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getlibovversion(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.24 Get LibOVKS version

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getlibovksversion(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.25 Get list of libraries in the database

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getlibraries(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.26 Get list of semantic flags in the database

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getsemanticflags(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.27 Get server description

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getserverdescription(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.28 Get server name

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getservername(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.29 Get server time

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getservertime(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.30 Get server version

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getserverversion(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.31 Get startup time

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getstartuptime(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

27.32 Get list of structures in the database

Header file

```
#include "libov/ov_vendortree.h"
```

Declaration

```
OV_DLLFNCEXPORT OV_RESULT ov_vendortree_getstructures(
    OV_VAR_CURRENT_PROPS   *pvarcurrprops,
    const OV_TICKET        *pticket
);
```

Parameters

`pvarcurrprops`

`pticket`

Return value

Remarks

Table of Contents