ObjectDataBase

Release 0.1.17

Apr 07, 2018

Contents:

CHAPTER 1

COdb

The OjectDatabase is a Graph-Database.

It aims to build a natural image of the relevant links between CThings (=nodes) amongst each other as well as things and atoms (=data fields).

CThings are objects or nodes, like cars in cars or trees or people.

CThing's may be linked amongst each other for specified CReasons CReason's are unidirectional linking CThing's CProperty is a primitive property of a CThing CAtom's are attributes of CThings containing descriptive data

- A thing may be a tree
- Attributes of a tree may be, position, health status, animals living in it
- A tree may relate to other trees by taking the light, building a forrest
- ...

Sample code

```
#include <iostream>

#include "odb.h"

int main()
   {
   auto oOdb = odb::COdb();
   oOdb.print_json( std::cout );
   std::cout << '\n';
   }
```

class odb::**COdb**

The database.

A Object Database itself is an Identifiable object. Enabeling multiple database instances in one application.

Author Manfred Morgner

Since 0.1.17

Inherits from Identifiable< COdb >

Public Functions

COdb()
Creates a *COdb*, a database.

We only default construct the database, no copy construction no copy of the database at all. It is not known, what copying the database means.

COdb(*COdb* **const** &*src*)
deleted: copy constructor

Copying a databse is not supported yet. We will support it as soon as we find out what it means.

COdb(*COdb* **const** &&*src*)
deleted: move constructor

Moving a databse is not supported yet.

COdb &**operator=**(*COdb* **const** &*src*)
deleted: Assignment operator

Here we are in the same situation as with copy construction

virtual ~COdb()
Some cleanup.

If we wish to convience valgrind we are doing it right, we have to free all links to other objects before leaving the show.

void **clear**()
Frees all objects.

Frees all known objects at last so far that freeing the object collections releases all memory, bound to them. So valgrind will find nothing left on the table.

PThing **MakeThing**(std::string **const** &*crsName* = "")
Creates a PThing.

Creates a shared_ptr with a new *CThing* named as given in the call. If no name is given, the name will be the class default

Parameters

- `crsName`: The name for the *CThing*

PThing **LoadThing**(size_t *nId*, std::string **const** &*crsName* = "")
Creates a PThing with predefined ID.

Creates a shared_ptr with a new *CThing* named as given in the call. If no name is given, the name will be the class default

Parameters

- `nId`: The predefined ID if loading given sets into odb
- `crsName`: The name for the *CThing*

PProperty **MakeProperty**(std::string **const** &*crsName* = "")
Creates a PProperty.

Creates a shared_ptr with a new *CProperty* named as given in the call. If no name is given, the name will be the class default

Parameters

- `crsName`: The name for the *CProperty*

PProperty **LoadProperty**(size_t *nId*, std::string **const** &*crsName* = "")
Creates a PProperty with predefined ID.

Creates a shared_ptr with a new *CProperty* named as given in the call. If no name is given, the name will be the class default

Parameters

- `nId`: The predefined ID if loading given sets into odb
- `crsName`: The name for the *CProperty*

PAtom **const MakeAtom**(int *data*, std::string **const** &*crsName* = "", std::string **const** &*crsPrefix* = "", std::string **const** &*crsSuffix* = "", std::string **const** &*crsFormat* = "")
Creates a PAtom.

Creates a shared_ptr with a new *CAtom* named as given in the call. If no name is given, the name will be the class default

The data type of the data element in *CAtom* follows the input data type. It can be if any primitive type or most of the simple containers, like string or vector.

CAtom further on manages the life time of the data element. It's a unique_ptr

Parameters

- `data`: The data for the *CAtom*
- `crsName`: The name for the *CAtom*
- `crsPrefix`: The prefix for the *CAtom* in a GUI
- `crsSuffix`: The suffix for the *CAtom* in a GUI
- `crsFormat`: The format for the *CAtom* in a GUI

PAtom **const LoadAtom**(size_t *nId*, int *data*, std::string **const** &*crsName* = "", std::string **const** &*crsPrefix* = "", std::string **const** &*crsSuffix* = "", std::string **const** &*crsFormat* = "")
Creates a PAtom with predefined ID.

Creates a shared_ptr with a new *CAtom* named as given in the call. If no name is given, the name will be the class default

The data type of the data element in *CAtom* follows the input data type. It can be if any primitive type or most of the simple containers, like string or vector.

CAtom further on manages the life time of the data element. It's a unique_ptr

Parameters

- `nId`: The predefined ID if loading given sets into odb

- `data`: The data for the *CAtom*
- `crsName`: The name for the *CAtom*
- `crsPrefix`: The prefix for the *CAtom* in a GUI
- `crsSuffix`: The suffix for the *CAtom* in a GUI
- `crsFormat`: The format for the *CAtom* in a GUI

PReason **MakeReason**(std::string **const** &*crsName* = "")
Creates a PReason.

Creates a shared_ptr with a new *CReason* named as given in the call. If no name is given, the name will be the class default

Parameters

- `crsName`: The name for the *CReason*

PReason **LoadReason**(size_t *nId*, std::string **const** &*crsName* = "")
Creates a PReason with predefined ID.

Creates a shared_ptr with a new *CReason* named as given in the call. If no name is given, the name will be the class default

Parameters

- `nId`: The predefined ID if loading given sets into odb
- `crsName`: The name for the *CReason*

PStrand **MakeStrand**(std::string **const** &*crsName* = "")
Creates a PStrand.

Creates a shared_ptr with a new *CStrand* named as given in the call. If no name is given, the name will be the class default

Parameters

- `crsName`: The name for the *CStrand*

void **print**() **const**
Print out the database (Informative format)

void **print**(**const** CAtoms &*crContainer*) **const**
Print out container of *CAtom* objects.

Parameters

- `crContainer`: The forward iterable container, containing PAtom's

template <**typename** T>
void **print**(**const** CT<T> &*crContainer*) **const**
Print out container of *CThing*'s.

Parameters

- `crContainer`: The forward iterable container, containing all *CThing* instances

auto **Escape**(std::string **const** &*crsInput*)
Replaces 2 with " and \ with .

Parameters

- `crsInput`: The string to be escaped

void **print_json**(CThings **const** &*crContainer*, std::ostream &*ros*)
Dump all CThings in Sub-JSON format.

Parameters

- `crContainer`: The forward iterable container, containing all *CThing* instances
- `ros`: The output destination

void **print_json_stream**(CThings **const** &*crContainer*, std::ostream &*ros*)
Dump all CThings in Sub-JSON format.

Parameters

- `crContainer`: The forward iterable container, containing all *CThing* instances
- `ros`: The output destination

void **print_json**(CProperties **const** &*crContainer*, std::ostream &*ros*)
Dump all *CProperty*'s in Sub-JSON format.

Parameters

- `crContainer`: The forward iterable container, containing all *CProperty* instances
- `ros`: The output destination

void **print_json_stream**(CProperties **const** &*crContainer*, std::ostream &*ros*)
Dump all *CProperty*'s in Sub-JSON format.

Parameters

- `crContainer`: The forward iterable container, containing all *CProperty* instances
- `ros`: The output destination

void **print_json**(CAtoms **const** &*crContainer*, std::ostream &*ros*)
Dump all CAtoms in Sub-JSON format.

Parameters

- `crContainer`: The forward iterable container, containing all *CAtom* instances
- `ros`: The output destination

void **print_json_stream**(CAtoms **const** &*crContainer*, std::ostream &*ros*)
Dump all CAtoms in Sub-JSON format.

Parameters

- `crContainer`: The forward iterable container, containing all *CAtom* instances

- `ros`: The output destination

void **`print_json`**(CReasons **`const`** &*crContainer*, std::ostream &*ros*)
Dump all CReasons in Sub-JSON format.

Parameters

- `crContainer`: The forward iterable container, containing all *CReason* instances
- `ros`: The output destination

void **`print_json_stream`**(CReasons **`const`** &*crContainer*, std::ostream &*ros*)
Dump all CReasons in Sub-JSON format.

Parameters

- `crContainer`: The forward iterable container, containing all *CReason* instances
- `ros`: The output destination

void **`print_json`**(std::ostream &*ros*)
Dump the hole database in JSON format.

see also: JSON validator+converter: https://jsonformatter.org/

Sample: Link Atoms to Things and Thing to Thing and dump it as JSON

```
#include <iostream>

#include "odb.h"
#include "atom.h"
#include "thing.h"

int main()
{
auto oOdb    = odb::COdb();
auto pThing1 = oOdb.MakeThing("Ulrich");
auto pThing2 = oOdb.MakeThing("Fred");
auto pAtom1  = oOdb.MakeAtom("Leader", "Role");
auto pAtom2  = oOdb.MakeAtom("Member", "Role");
auto pReason = oOdb.MakeReason("pays");
pThing1->Append(pAtom1);
pThing2->Append(pAtom2);
pThing1->Link(pThing2, pReason);
oOdb.print_json(std::cout);
}
```

Output

```
{
"Object Database Dump":
    {
    "Sizes": [ {"P": 0},{"A": 2},{"R": 1},{"T": 2} ],
    "Properties":
        [
        ],
    "Atoms":
        [
```

```
            { "id": 0, "data": "Leader" },
            { "id": 1, "data": "Member" }
        ],
    "Reasons":
        [
            { "id": 0, "name": "pays" }
        ],
    "Things":
        [
            { "id": 0, "name": "Ulrich",
                "properties": [  ],
                "atoms": [ {"id": 0} ],
                "links": [ {"thing-id": 1, "reason-id": 0} ] },
            { "id": 1, "name": "Fred",
                "properties": [  ],
                "atoms": [ {"id": 1} ],
                "links": [  ] }
        ]
    }
}
```

Parameters

- `ros`: The output destination

void **`print_json_stream`**(std::ostream &*ros*)
Dump the hole database in JSON format.

Parameters

- `ros`: The output destination

void **`SaveDB`**(std::string **`const`** &*crsFilename*)
Saves an odb json file.

Parameters

- `crsFilename`:

bool **`LoadDB`**(std::string **`const`** &*crsFilename*)
Loads an odb json file.

Parameters

- `crsFilename`:

PThing **`FindOrLoadThingById`**(size_t **`const`** *nId*, std::string **`const`** &*crsName* = "")
Has to return a thing with specified ID, if it does not exists, it is to make.

Parameters

- `nId`: The id of the thing
- `crsName`: The name of the thing if it has to be created

OThing **`FindThingByProperty`**(std::string **`const`** &*crsProperty*)
Finds PThing with a named Property only if it's unique.

Parameters

- `crsProperty`: The name for the *CProperty*

CThings **FindThingsByProperty**(std::string **const** &*crsProperty*)
Finds PThing with a named Property only if it's one.

Parameters

- `crsProperty`: The name for the *CProperty*

CThings **FindThingsByProperty**(std::regex **const** &*crsRegex*)
Finds PThing with a named Property only if it's one.

Parameters

- `crsRegex`: The name for the *CProperty*

PThing **FindOrMakeThingByProperty**(std::string **const** &*crsThing*, std::string **const** &*crsProperty*)
Finds or creates a PThing with a named Property, which also may be created and assigned.

Parameters

- `crsThing`: The name for the *CThing*
- `crsProperty`: The name for the *CProperty*

PProperty **FindOrMakeProperty**(std::string **const** &*crsProperty*)
Has to return a property, if it does not exists, it is to make.

Parameters

- `crsProperty`: The name of the Property

PReason **FindOrMakeReason**(std::string **const** &*crsReason*)
Has to return a Reason, if it does not exists, it is to make.

Parameters

- `crsReason`: The name of the Reason

bool **AppendProperty2Thing**(size_t *nProperty*, size_t *nThing*)
todo: optimize / Appends a Property to a Thing by given index value

bool **AppendProperty2Thing**(std::string **const** &*crsProperty*, bool *bForce*, std::string **const** &*crsThing*)
todo: optimize / Appends a Property to a Thing by given names

bool **AppendAtom2Thing**(size_t *nThing*, size_t *nAtom*)
todo: optimize / Appends an Atom to a Thing by given index value

bool **LinkThing2Thing**(size_t *nThingFrom*, size_t *nThingTo*, size_t *nReason*)
todo: optimize / Links a Thing to a Thing for a Reason by given index value

template <typename T>
std::optional<PT<T>> **Find**(CT<T> **const** &*croContainer*, size_t *nId*)
Finds the T with ID nId.

Template Parameters

- `T`: the type of the filtered objects

Parameters

- `croContainer`: The container to be filtered
- `nId`: The ID of the T

template <typename T>
CT<T> **Find**(CT<T> **const** &*croContainer*, std::string **const** &*crsName*)
Finds all Ts with the given NAME.

Template Parameters

- `T`: the type of the filtered objects

Parameters

- `croContainer`: The container to be filtered
- `crsName`: The NAME of the Reasons

template <typename T>
CT<T> **Find**(CT<T> **const** &*croContainer*, std::regex **const** &*crsRegex*)
Finds all Ts with the given NAME as RegEx.

Template Parameters

- `T`: the type of the filtered objects

Parameters

- `croContainer`: The container to be filtered
- `crsRegex`: The NAME of the Ts in Regex

auto **FindThing**(size_t *nId*)
API Adapter.

auto **FindThings**(std::string **const** &*crsName*)
API Adapter.

auto **FindThings**(std::regex **const** &*crsRegex*)
API Adapter.

auto **FindProperty**(size_t *nId*)
API Adapter.

auto **FindProperties**(std::string **const** &*crsName*)
API Adapter.

auto **FindProperties**(std::regex **const** &*crsRegex*)
API Adapter.

auto **FindReason**(size_t *nId*)
API Adapter.

auto **FindReasons**(std::string **const** &*crsName*)
API Adapter.

auto **FindReasons** (std::regex **const** &*crsRegex*)
API Adapter.

auto **FindAtoms** (size_t *nId*)
API Adapter.

auto **FindAtoms** (std::string **const** &*crsName*)
API Adapter.

auto **FindAtoms** (std::regex **const** &*crsRegex*)
API Adapter.

CThings **const** &**Things** () **const**
Access function to call then container of *CThing*'s.

CProperties **const** &**Properties** () **const**
Access function to call then container of CProperties.

CAtoms **const** &**Atoms** () **const**
Access function to call then container of *CAtom*'s.

CReasons **const** &**Reasons** () **const**
Access function to call then container of *CReason*'s.

CStrands **const** &**Strands** () **const**
Access function to call then container of *CStrand*'s.

Protected Attributes

CThings **m_oThings**
A container instance of *CThing*'s.

CProperties **m_oProperties**
A container instance of CProperties.

CAtoms **m_oAtoms**
A container instance of *CAtom*'s.

CReasons **m_oReasons**
A container instance of *CReason*'s.

CStrands **m_oStrands**
A container instance of *CStrand*'s.

CHAPTER 2

CThing

The Thing, formerly known as Object in the Object Database, renamed for the practicle reason of having a unique starting letter amongst the other code units.

It may contain an arbitrary amount of arbitrary Atoms (equivalents to Data Fields), Links to other CThing's for specified CReason's as well as Backlinks to Reason'ed Link sources.

The linking CThing is responsible for ressource management. It manages connections

- from thing to thing
- the backlink for links from itself to another thing
- from thing to atoms
- the backlink for links from itself to atoms

Sample code

```
/**
 *  @file main.cpp
 *
 *  @author Manfred Morgner
 *  @date 21.01.2018
 */

#include <iostream>

#include "odb.h"
#include "thing.h"

/// @brief Demo main program "linking things together"
int main()
    {
    // creating things
    auto oOdb    = odb::COdb();
    auto pThing1 = oOdb.MakeThing("Ulrich");
    auto pThing2 = oOdb.MakeThing("Fred");
```

```
    auto pReason = oOdb.MakeReason("pays");

    // linking them together
    pThing1->Link(pThing2, pReason);

    // 1) print the thing's view
    std::cout << "thing: " << *pThing1 << '\n';
    std::cout << "thing: " << *pThing2 << '\n';

    // 2) print complete database in simple form
    oOdb.print();

    // 3) print complete database in json format
    oOdb.print_json(std::cout);
    }
```

Output 1: The CThing's explaining there content

```
thing: Ulrich
  Role: Leader
   => linked to: "Fred" for reason: "pays" = Ulrich pays Fred
thing: Fred
  Role: Member
   <= linked from: Ulrich
```

Output 2: The whole database in list format

```
#    TYPE      ID       NAME          RefCnt    DATA
#=========================================================
odb::CThing   id: 0   name: Ulrich   (5)
odb::CThing   id: 1   name: Fred     (5)
odb::CAtom    id: 0   name: Role     (3)       data: Leader
odb::CAtom    id: 1   name: Role     (3)       data: Member
odb::CReason  id: 0   name: pays     (3)
```

Output 3: The whole database in JSON format

```
{
"Object Database Dump":
    {
    "Things":
        [
            { "id": "0", "name": "Ulrich",
              "atoms": [ {"id": "0"} ],
              "links": [ {"thing-id": "1", "reason-id": "0"} ] },
            { "id": "1", "name": "Fred",
              "atoms": [ {"id": "1"} ],
              "links": [  ] }
        ],
    "Atoms":
        [
            { "id": "0", "name": "Role", "data": "Leader" },
            { "id": "1", "name": "Role", "data": "Member" }
        ],
    "Reasons":
        [
            { "id": "0", "name": "pays" }
        ]
```

```
    }
}
```

class odb::**CThing**

A Thing as there are many.

Author Manfred Morgner

Since 0.1.17

Inherits from enable_shared_from_this< CThing >, Identifiable< CThing >

Public Functions

CThing()

We never construct without a name for the thing.

CThing(*CThing* **const**&)

and we don't make copies

CThing(*CThing*&&)

make_shared<T> moveconstructs

CThing(std::string **const** &*crsName*)

Normal constructor, receiving the name of the reason.

CThing(size_t *nId*, std::string **const** &*crsName*)

Load constructor, receiving the ID and name of the reason.

virtual **~CThing**()

Nothings special here.

void **clear**()

We need to unbind all relations in the odb before destructing.

PProperty **Append**(PProperty *poProperty*)

Appends an *CProperty* to its property list.

Appending an *CProperty* to this *CThing* requires the thing to inform the appended Property about this *CThing* is linking to it

Parameters

- `poProperty`: A Property to bind with the Thing

PAtom **Append**(PAtom *poAtom*)

Appends an *CAtom* to its atom list.

Appending an *CAtom* to this *CThing* requires the thing to inform the appended Atom about this *CThing* is linking to it

Parameters

- `poAtom`: An Atom to bind into the Thing

PThing **Link** (PThing *po2Thing*, PReason *po4Reason*)
Links this *CThing* to another *CThing* for a *CReason*.

Parameters

- `po2Thing`: A Thing to Link to
- `po4Reason`: The Reason we link for

PThing **Unlink** (PThing *po2Thing*, PReason *po4Reason*)
Removes a link to a specific *CThing* with a specific *CReason*.

Parameters

- `po2Thing`: A Thing to Linked to
- `po4Reason`: The Reason we linked for

PThing **RelatingThingAdd** (PThing *poThing*)
adds a *CThing* as referencing to this

Parameters

- `poThing`: A *CThing* that links to us notifies us, we register it

PThing **RelatingThingSub** (PThing *poThing*)
subtract a *CThing* as referencing to this

Parameters

- `poThing`: A *CThing* that linked to us notifies us, we deregister it

Public Static Attributes

constexpr auto **s_csNameUnnamedThing** = {"unnamedThing"}
The name of the thing.

constexpr bool **s_bDebug** = {false}
Do we generate debug output?

Protected Attributes

std::multimap<PThing, PReason, lessIdentifiableId<PThing>> **m_mLink**
Holds the links to another *CThing* for *CReason*.

Parameters

- `PThing`: The PThing we link to
- `PReason`: The PReason we link for
- `Compare`: Function to compare two CThings

std::set<PThing, lessIdentifiableId<PThing>> **m_spoThingsRelating**
Registers PThings relating to itself.

Parameters

- `PThing`: The PThing we are linked from
- `Compare`: Function to compare two *CThing*'s

std::set<PProperty, lessIdentifiableName<PProperty>> **m_spoProperties**
Registers PProperties of this *CThing*.

Parameters

- `PProperty`: PProperties we have
- `Compare`: Function to compare two PAtom's

std::set<PAtom, lessIdentifiableId<PAtom>> **m_spoAtoms**
Registers PAtoms of this *CThing*.

Parameters

- `PAtom`: PAtom's we own
- `Compare`: Function to compare two PAtom's

Friends

bool **operator==** (PThing **const** &*croThing*, std::string **const** &*crsInput*)
Compares the name with an input string.

bool **operator<** (PThing **const** &*croThing*, std::string **const** &*crsInput*)
Compares the name with an input string.

std::ostream &**operator<<** (std::ostream &*ros*, *CThing* **const** &*crThing*)
The free output operator for *CThing*.

Parameters

- `ros`: The output stream to send the Thing to
- `crThing`: The Thing to output

CHAPTER 3

CProperty

Sample code

```
/**
 *  @file main.cpp
 *
 *  @author Manfred Morgner
 *  @date 10.02.2018
 */

#include <iostream>

#include "odb.h"
#include "thing.h"
#include "property.h"

/// @brief Demo main program for "property in thing"
int main()
   {
   auto oOdb = odb::COdb();
   auto thing = oOdb.MakeThing( "Tree" );
   auto property = oOdb.MakeProperty( "Acorn" );
   thing->Append(property);
   std::cout << "thing: " << *thing;
   std::cout << '\n';
   }
```

Output

```
thing: Tree
  Property: Acorn
```

class odb::**CProperty**

A Property for a *CThing*.

Inherits from Identifiable< CProperty >

Public Functions

CProperty()
deleted: default constructor

CProperty(*CProperty* **const**&)
There is no reason to copy a *CProperty*.

CProperty(*CProperty*&&)
There is no reason to moveconstruct a *CProperty*.

CProperty(std::string **const** &*crsName*)
Normal constructor, receiving the name of the property.

CProperty(size_t *nId*, std::string **const** &*crsName*)
Load constructor, receiving the id and the name of the property.

operator std::string const&()
Conversion operator will return the name of the instance.

void **RelationAdd**(PThing *poThing*)
Add the information about a link from a *CThing*.

void **RelationSub**(PThing *poThing*)
Removes a link to a *CThing*.

Parameters

- `poThing`: The thing the link is pointing to

void **print**()
Prints an informational output to std::cout.

SThings **const** &**Relations**() **const**
Access function to call then container of PThings's.

Public Static Attributes

constexpr auto **s_csNameUnnamedProperty** = {"unnamedProperty"}
The name of an unnamed property.

Protected Attributes

std::set<PThing> **m_oRelations**
A set containing backlinks from things.

Friends

std::ostream &**operator<<**(std::ostream &*ros*, *CProperty* **const** &*croProperty*)
Output operator to do an output of the instance.

bool **operator==**(PProperty **const** &*croProperty*, std::string **const** &*crsInput*)
Compares the name with an input string.

CHAPTER 4

CReason

This class is necessary to link two CThing instances to give the link an explaination. This enables us to link the same CThing's multiple time.

- thing1 - reason - thing2

For example:

- Heinz - wrote - 'Trees of Estonia'
- Heinz - signed - 'Trees of Estonia'

Links are unidirectional. Meaning if it's true that

- Heinz - wrote - 'Trees of Estonia'

it may not be true that

- 'Trees of Estonia' - wrote - Heinz

But to ensure thing2 feels the link, it will be informed that a link to it became established. The linked thing registers which thing is linking to it only ones. In our example 'Trees of Estonia' registers, that Heinz links to it.

If some process/entity needs to know how often and for which reasons, it has to go to Heinz and ask. The linking thing is resonsible for correct management of links, reasons and backlinks.

CReason registers each link it is used for

Demostration

```
#include <iostream>

#include "odb.h"
#include "thing.h"

int main()
    {
    // generating the objects
    auto oOdb    = odb::COdb();
    auto pThing1 = oOdb.MakeThing("Ulrich");
```

```
    auto pThing2 = oOdb.MakeThing("Fred");
    auto pReason = oOdb.MakeReason("pays");
    // create a connection
    pThing1->Link(pThing2, pReason);
    // let them explain the situation
    std::cout << "thing: " << *pThing1 << '\n';
    std::cout << "thing: " << *pThing2 << '\n';
    }
```

Output:

```
thing: Ulrich
   => linked to: "Fred" for reason: "pays" = Ulrich pays Fred
thing: Fred
   <= linked from: Ulrich
```

class odb::**CReason**

A Reason to link two Things (Unidirectional)

Inherits from Identifiable< CReason >

Public Functions

CReason()

Forbidden.

CReason(*CReason* **const**&)

There is no reason to copy a *CReason*.

CReason(*CReason*&&)

make_shared<T> moveconstruct

CReason(std::string **const** &*crsName*)

Normal constructor, receiving the name of the reason.

CReason(size_t *nId*, std::string **const** &*crsName*)

Load constructor, receiving the ID and name of the reason.

operator std::string const&()

Conversion operator will return the name of the instance.

void **RelationAdd**(PThing &*poThingFrom*, PThing &*poThingTo*)

Add the information about a link from one *CThing* to another regarding 'this' reason.

void **RelationSub**(PThing &*poThingFrom*, PThing &*poThingTo*)

Removes a link between two *CThing*'s.

Removes the information about a particular link from one *CThing* to another regarding 'this' reason

Parameters

- `poThingFrom`: The thing the link is claimed to be made from
- `poThingTo`: The thing the link is claimed to be made to

void **print**()

Prints an informational output to std::cout.

Public Static Attributes

constexpr auto **s_csNameUnnamedReason** = {"unnamedReason"}
The name of an unnamed reason.

Protected Attributes

std::multimap<PThing, PThing> **m_mRelations**
A map containing links from one thing to another.

Friends

std::ostream &**operator<<** (std::ostream &*ros*, *CReason* **const** &*croReason*)
Output operator to do an output of the instance.

CHAPTER 5

CAtom

An CAtom is a container for a single data element, let's say a number or a text. It stores additional information to use in a GUI as there are

- Name
- Prefix
- Suffix
- Format template

One can compare an atom with single data field like in a conventional database. Unlike conventional databases fields/atoms do not have a fixed structure, they even do not have to exist.

CAtom may act as a template for other atoms. In such case the atom, using the other as template, does not need to fill elements which are given by the template. It will appear as if the elements of the template are elements of the using atom, as long as they are not overwritten.

Sample code

```
#include <iostream>

#include "odb.h"
#include "atom.h"

int main()
   {
   auto oOdb = odb::COdb();
   auto atom = oOdb.MakeAtom(2.5, "gain", "is", "%");
   std::cout << "atom data: " << *atom << '\n';
   std::cout << "atom frmt: " << atom->m_sName << ' ';
   atom->print_atom_data_formated(std::cout);
   std::cout << '\n';
   }
```

Output

```
atom data: 2.5
atom frmt: gain is 2.5 %
```

class odb::**CAtom**

An Atom is a data field for a *CThing*.

Sample Code goes here

```
#include <odb>
```

Inherits from enable_shared_from_this< CAtom >, Identifiable< CAtom >

Public Functions

template <**typename** T>
CAtom(T *tAtomData*, std::string **const** &*crsName* = "", std::string **const** &*crsPrefix* = "", std::string **const** &*crsSuffix* = "", std::string **const** &*crsFormat* = "")

Constructor able to receive data of maany types.

Parameters

- tAtomData: The data unit to encapsulate
- crsName: The name for the atom
- crsPrefix: The prefix for user output
- crsSuffix: The suffix for user output
- crsFormat: The format for user output

template <**typename** T>
CAtom(size_t *nId*, T *tAtomData*, std::string **const** &*crsName* = "", std::string **const** &*crsPrefix* = "", std::string **const** &*crsSuffix* = "", std::string **const** &*crsFormat* = "")

Constructor able to receive data of maany types.

Parameters

- nId: The predefiined ID if loading a dataset
- tAtomData: The data unit to encapsulate
- crsName: The name for the atom
- crsPrefix: The prefix for user output
- crsSuffix: The suffix for user output
- crsFormat: The format for user output

virtual **~CAtom**()

Destruction as usual (=default)

void **clear**()

Remove all links between all objects.

This is necessary to enable freeing of all memory ressources. So we become able to put valgrind to use

void **print_xml**(std::ostream &*out*, size_t **const** *cnDepth*, bool *bFormated* = false) **const**

todo: output the instance xml formated

void **print_atom_data_formated**(std::ostream &*out*) **const**
Prints the content of the instance for UI use (well formated)

auto **RelatingThingAdd**(PThing *poThing*)
Adds the backlink from the atom to a thing

Parameters

- `poThing`: Inform a thing about being linked from another thing

auto **RelatingThingSub**(PThing *poThing*)
Removes the backlink from the atom to a thing

Parameters

- `poThing`: Inform a thing about no more being linked from another thing

Public Static Attributes

constexpr auto **s_csNameUnnamedAtom** = {"unnamedAtom"}
The name of an unnamed atom.

constexpr bool **s_bDebug** = {false}
Switch to enable/disable debug information output, regarding *CAtom*.

Protected Attributes

std::string **m_sFormat** = {""s}
The UI output format for the atom.

std::string **m_sPrefix** = {""s}
The UI prefix (if any) for the atom.

std::string **m_sSuffix** = {""s}
The UI suffix (if any) for the atom.

std::set<PThing, lessIdentifiableId<PThing>> **m_spoThingsRelating**
CThing's Relating to this *CAtom*.

std::unique_ptr<**const** SAtomDataConcept> **m_pAtomData**
The pointer and holder of the data element of type T.

Friends

void **print**(CAtoms **const** &*crContainer*)
friend function to print the atom instance in an inforamtional manner

std::ostream &**operator<<**(std::ostream &*ros*, *CAtom* **const** &*croAtom*)
sends the data of the atom to the given ostream

template <**typename** T, **typename** U>
struct decay_equiv
Compares the type of a variable with a chosen type for similarity, e.g:

- if (decay_equiv<T, int>::value) . . .

Inherits from std::is_same::type< std::decay< T >::type, U >

template <**typename** T>

struct SAtomData
The templated data structure to hold an arbitrary data element.

Inherits from *CAtom::SAtomDataConcept*

Public Functions

SAtomData(T *tData*)
The function to deal with the arbitrary data element.

Parameters
- `tData`: The data element to hold

void **ToStream**(std::ostream &*ros*) **const**
Send the data element to std::ostream.

Helper function to break the boundary between non uniform data content of the atom instance and the uniform output expectation

Public Members

T **m_tData**
The decalartion of the data element of type T.

struct SAtomDataConcept
start of data implementation

Subclassed by *CAtom::SAtomData< T >*

Public Functions

virtual void **ToStream**(std::ostream&) **const** = 0
Will send the data of the atom to the given stream.

CHAPTER 6

CStrand

class odb::**CStrand**

A Strand to link two Things (Unidirectional)

Inherits from Identifiable< CStrand >

Public Functions

CStrand()

forbidden

CStrand(std::string **const** &*crsName*)

Normal constructor requesting a name for the strand.

CStrand(*CStrand* **const**&)

and we don't make copies

Nothings special

CStrand(*CStrand*&&)

make_shared<T> moveconstructs

PAtom **Append**(PAtom *poAtom*)

Appending another atom to the strand.

operator std::string const&()

Function to receive the name of the strand.

void **print**()

Print the strand in an informational manner.

Public Static Attributes

constexpr auto **s_csNameUnnamedStrand** = {"unnamedStrand"}
The name of an unnamed strand.

Protected Attributes

std::string **m_sName** = { *s_csNameUnnamedStrand* }
The name of an unnamed strand.

CAtoms **m_poAtoms**
The atoms of the strand.

Friends

std::ostream &**operator<<**(std::ostream &*ros*, *CStrand* **const** &*croStrand*)
friend function to send the atoms of the strand to the given stream

CHAPTER 7

Indices and tables

- genindex
- search

Index

Symbols

A

C

E

F

L

M

O

P

R

S

T

U