self_architecture

COLLABORATORS			
	TITLE : self_architecture		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 29, 2021	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

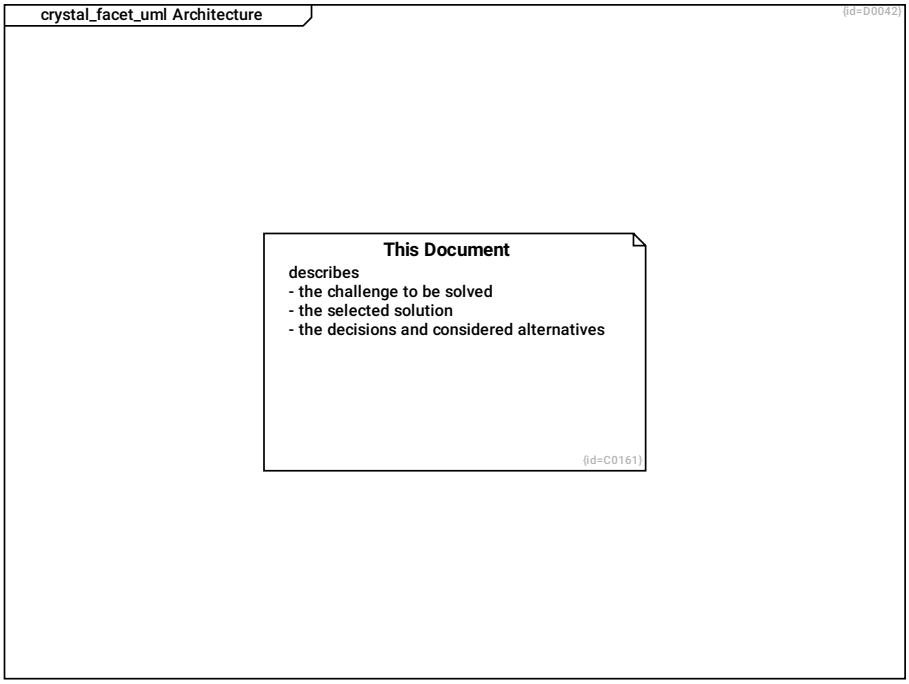
Contents

Chapter 1

crystal_facet_uml Architecture

About this Document:

(C) We acknowledge that this document uses material from the arc 42 architecture template, http://www.arc42.de. Created by Dr. Peter Hruschka & Dr. Gernot Starke.



This Document C0161

describes

- the challenge to be solved

- the selected solution

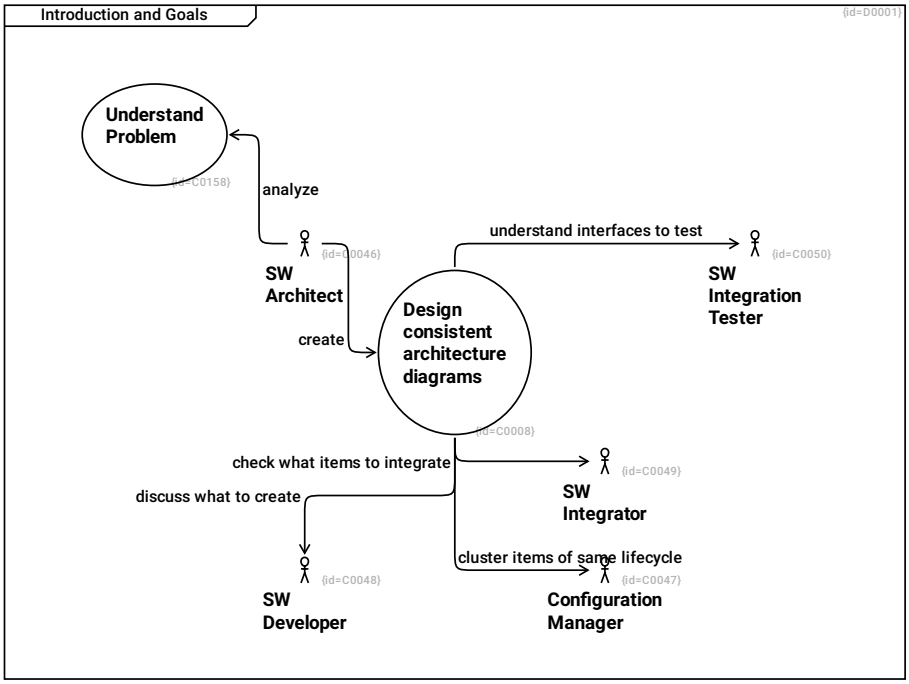
- the decisions and considered alternatives

1.1 Introduction and Goals

crystal_facet_uml creates sysml/uml diagrams to document system and software architecture.

As software architect, you create a set of diagrams describing use-cases, requirements, structural views, behavioral and deployment views.

This diagram shows the use-cases that crystal facet uml addresses.



SW Integrator C0049

Understand Problem C0158

Understand

- Use Cases
- Requirements
- Quality Goals

SW Architect C0046

create --> Design consistent architecture diagrams R0035

analyze --> Understand Problem R0224

Design consistent architecture diagrams C0008

The main goal of crystal facet uml is to create a set of consistent uml diagrams. Diagrams help in desinging a system that satisfies a set of requirements and decomposes the system into smaller entities.

Consistency encompasses:

- Consistent relations between objects
- Consistent naming of objects
- Consistent locations of objects in diagrams

check what items to integrate --> SW Integrator R0036

understand interfaces to test --> SW Integration Tester R0037

cluster items of same lifecycle --> Configuration Manager R0038

discuss what to create --> SW Developer R0039

Configuration Manager C0047

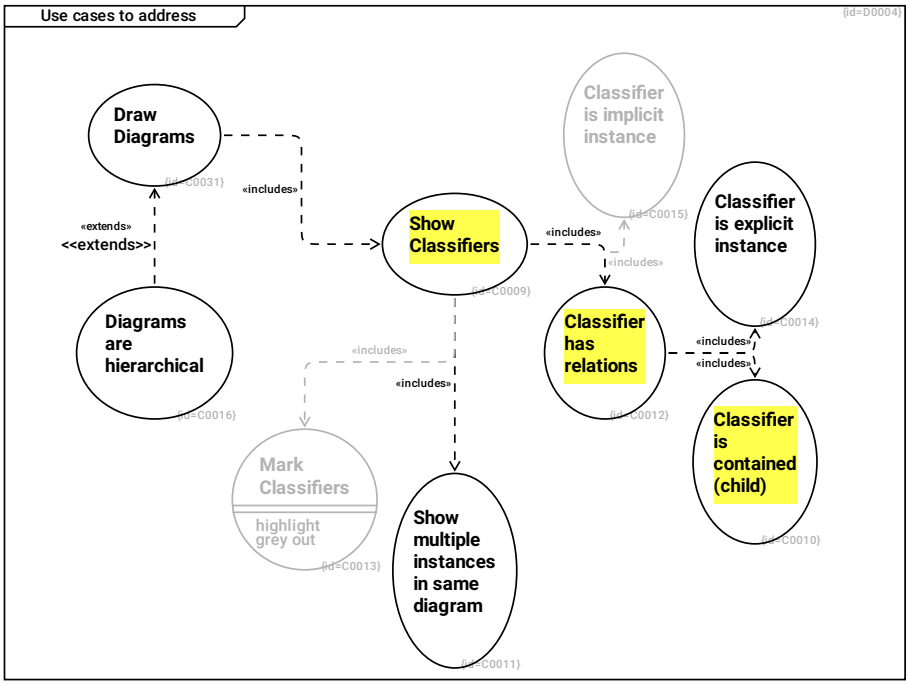
SW Developer C0048

A software developer understands the requirements and architecture; creates the software parts and performs reviews and unit-tests.

SW Integration Tester C0050

1.1.1 Use cases to address

This diagram lists the main features of crystal facet uml.



Show Classifiers C0009

Classifiers are drawn to diagrams one classifier is visible in one or multiple diagrams

--> Show multiple instances in same diagram R0058

--> Mark Classifiers R0061

--> Classifier has relations R0062

Classifier is contained (child) C0010

A Classifier can either exist stand-alone or be a sub-element of another classifier.

Show multiple instances in same diagram C0011

One and the same classifier can be shown in a diagram multiple times (either as class/type or as instance)

Classifier is explicit instance C0014

A classifier can be an explicitly modelled instance of another classifier

Classifier is implicit instance C0015

A classifier can represent an anonymous (implicit) instance without defining an explicit instance relation

Diagrams are hierarchical C0016

There is one root diagram. All other diagrams shall have a parent diagram.

<<extends>> --> Draw Diagrams R0014

Draw Diagrams C0031

--> Show Classifiers R0057

Classifier has relations C0012

A classifier can have zero, one or many relations; these may be of one or of differnet types. Circular dependencies may be forbidden if semantics undefined.

--> Classifier is implicit instance R0059

--> Classifier is explicit instance R0060

--> Classifier is contained (child) R0063

Mark Classifiers C0013

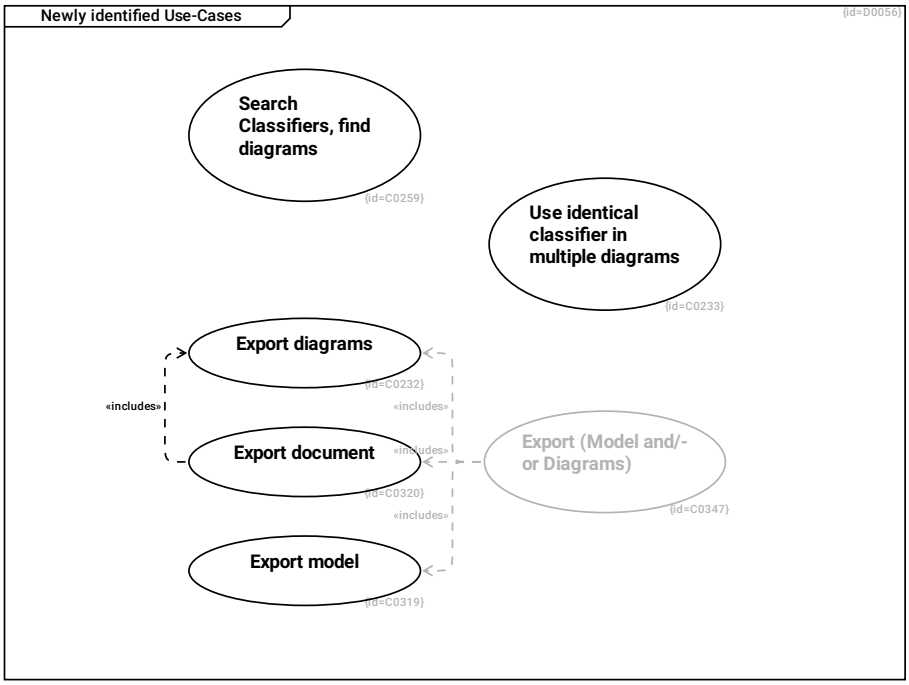
A classifier can be highlighted in one or several diagrams. The highlighting may be different in different diagrams where the classifier is shown.

highlight F0007

grey out F0008

1.1.2 Newly identified Use-Cases

During development, new use cases are identified.



Export (Model and/or Diagrams) C0347

--> Export diagrams R0495

--> Export document R0496

--> Export model R0497

Search Classifiers, find diagrams C0259

Use identical classifier in multiple diagrams C0233

Provide means to use the identical classifier in multiple diagrams

Export diagrams C0232

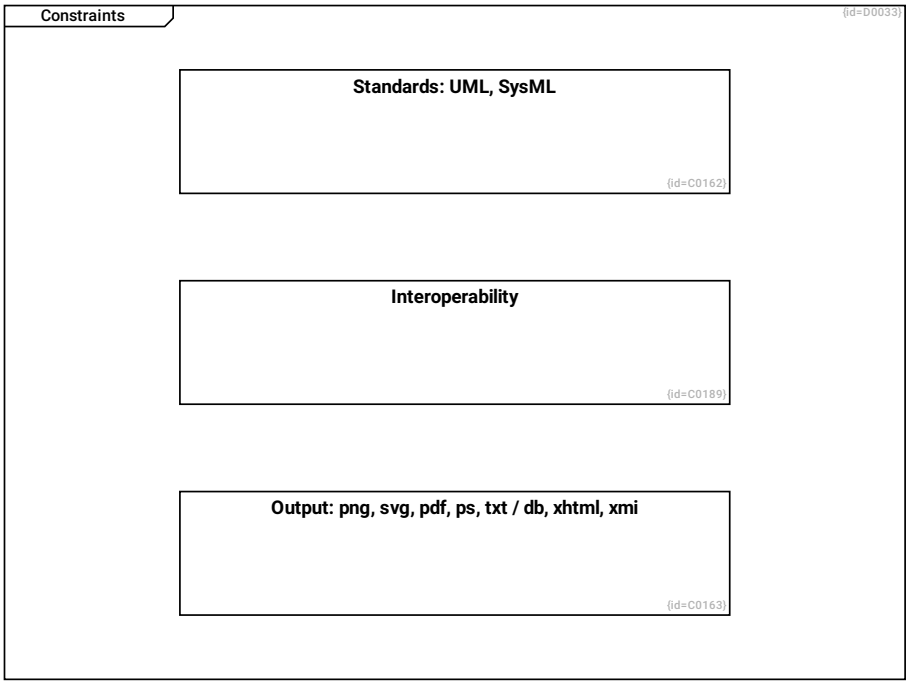
Export diagrams for use in architecture Documents.

Export model C0319

Export document C0320

--> Export diagrams R0456

1.2 Constraints



Standards: UML, SysML C0162

crystal_facet_uml shall draw diagrams that are compliant (subset) to UML or SysML.

Interoperability C0189

crystal_facet_uml has its strenghts. Providing a full-featured text layout engine is not one of them.

Therefore it is important to be interoperable with text layout engines like

- latex
- docbook
- markdown, asciidoc, doxygen

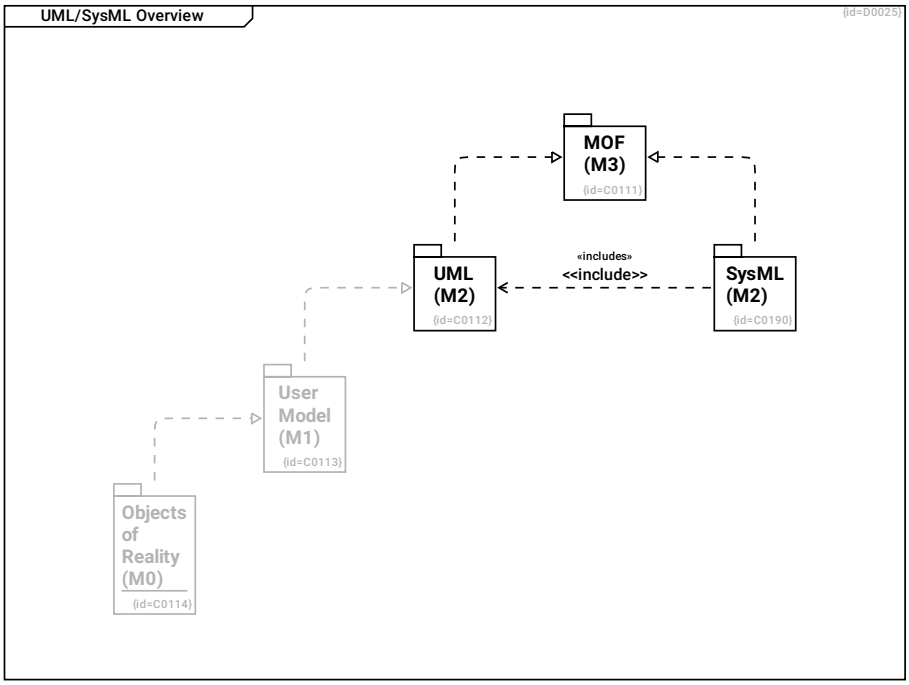
Output: png, svg, pdf, ps, txt / db, xhtml, xmi C0163

crystal_facet_uml shall produce output suitable to be processed by text layout engines:

- latex
- docbook
- doxygen
- html

1.2.1 UML/SysML Overview

This diagram shows an overview on UML and aims to explain what parts are available in crystal_facet_uml and how this is mapped to the crystal_facet_uml database.



MOF (M3) C0111

User Model (M1) C0113

--> UML (M2) R0151

Objects of Reality (M0) C0114

--> User Model (M1) R0150

SysML (M2) C0190

--> MOF (M3) R0248

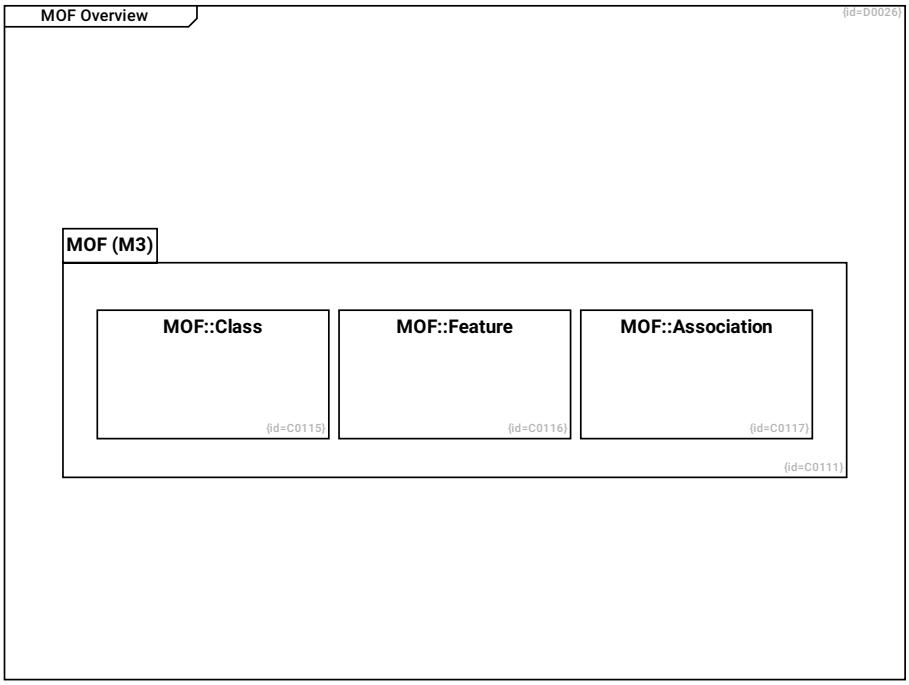
<<include>> --> UML (M2) R0249

UML (M2) C0112

--> MOF (M3) R0152

1.2.1.1 MOF Overview

This diagram shows a selection of MOF meta classes.



MOF::Class C0115

MOF::Feature C0116

MOF::Association C0117

This diagram shows some example classes of the MOF Meta Model.

MOF (M3) C0111

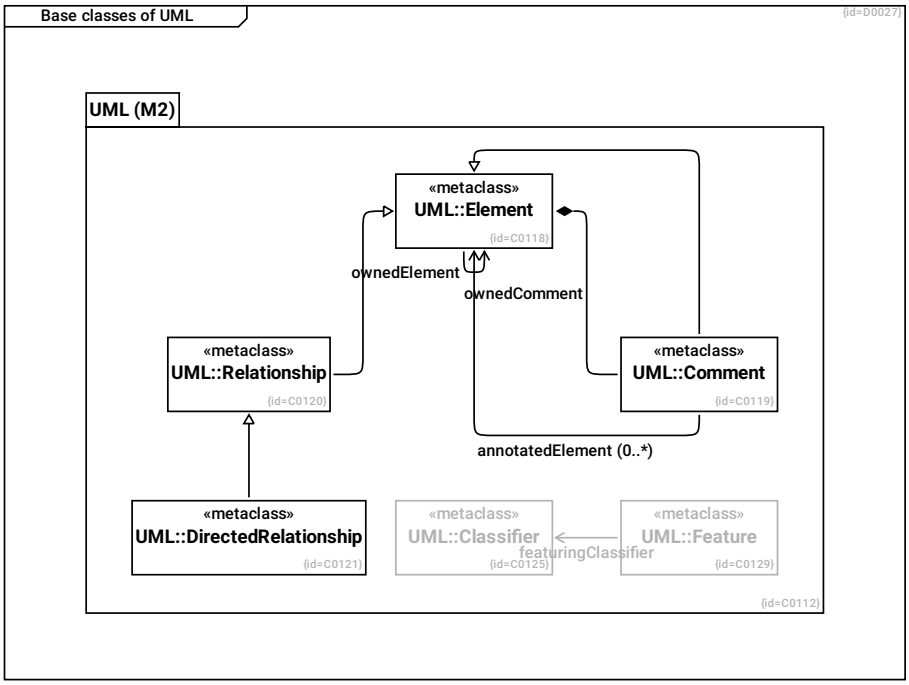
--> MOF::Class R0153

--> MOF::Feature R0154

--> MOF::Association R0155

1.2.1.2 Base classes of UML

This diagram shows the base classes of the UML



UML::Element C0118

ownedComment --> UML::Comment R0160

ownedElement --> UML::Element R0161

UML::Comment C0119

--> UML::Element R0162

annotatedElement (0..*) --> UML::Element R0439

UML::Relationship C0120

--> UML::Element R0163

UML::DirectedRelationship C0121

--> UML::Relationship R0164

UML::Classifier C0125

UML::Feature C0129

featuringClassifier --> UML::Classifier R0176

UML (M2) C0112

--> UML::Comment R0157

--> UML::Relationship R0158

--> UML::DirectedRelationship R0159

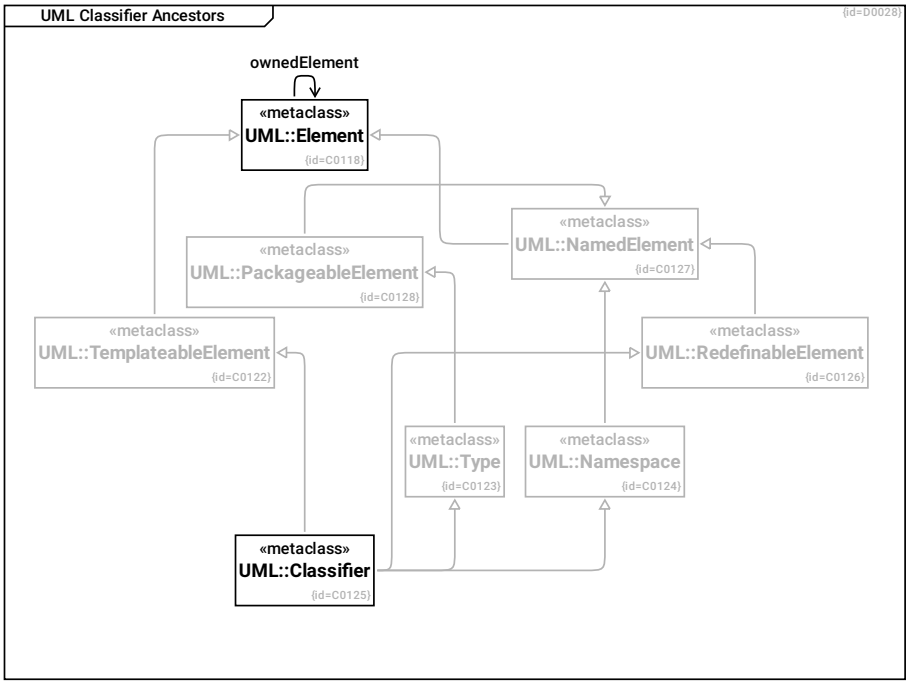
--> UML::Element R0156

--> UML::Classifier R0177

--> UML::Feature R0178

1.2.1.2.1 UML Classifier Ancestors

description: see http://www.omg.org/spec/UML/2.5



UML::TemplateableElement C0122

description: see http://www.omg.org/spec/UML/2.5

--> UML::Element R0170

UML::Type C0123

description: see http://www.omg.org/spec/UML/2.5

--> UML::PackageableElement R0173

UML::Namespace C0124

description: see http://www.omg.org/spec/UML/2.5

--> UML::NamedElement R0172

UML::Classifier C0125

--> UML::Namespace R0165

--> UML::TemplateableElement R0166

--> UML::Type R0167

--> UML::RedefinableElement R0168

UML::RedefinableElement C0126

description: see http://www.omg.org/spec/UML/2.5

--> UML::NamedElement R0169

UML::NamedElement C0127

description: see http://www.omg.org/spec/UML/2.5

--> UML::Element R0171

UML::Element C0118

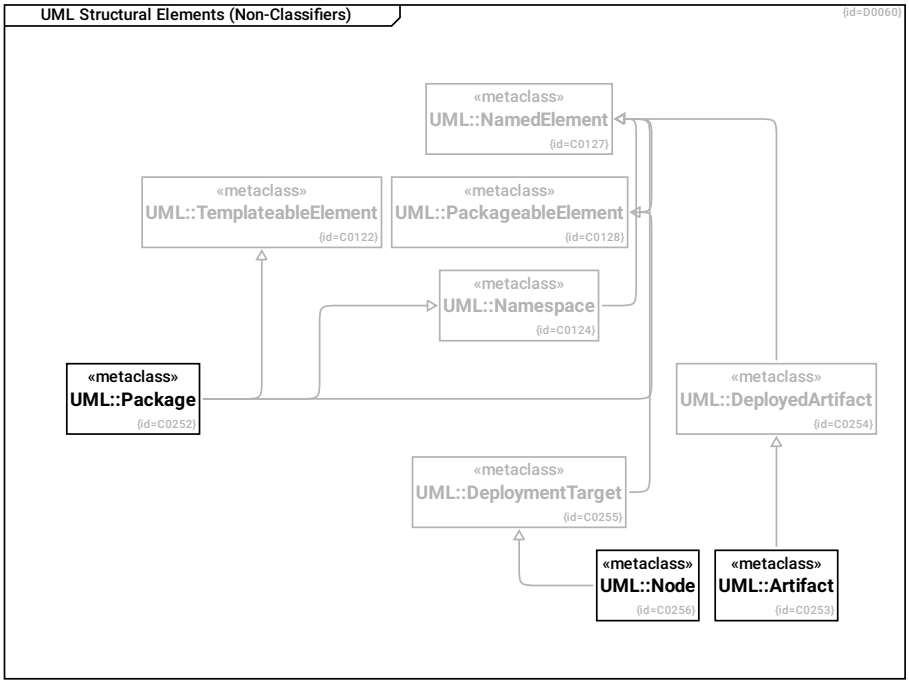
ownedElement --> UML::Element R0161

UML::PackageableElement C0128

description: see http://www.omg.org/spec/UML/2.5

--> UML::NamedElement R0174

1.2.1.2.1.1 UML Structural Elements (Non-Classifiers)



UML::PackageableElement C0128

description: see http://www.omg.org/spec/UML/2.5

--> UML::NamedElement R0174

UML::Namespace C0124

description: see http://www.omg.org/spec/UML/2.5

--> UML::NamedElement R0172

UML::TemplateableElement C0122

description: see http://www.omg.org/spec/UML/2.5

UML::NamedElement C0127

description: see http://www.omg.org/spec/UML/2.5

UML::Package C0252

--> UML::PackageableElement R0343

--> UML::Namespace R0342

--> UML::TemplateableElement R0344

UML::Artifact C0253

--> UML::DeployedArtifact R0345

UML::DeployedArtifact C0254

--> UML::NamedElement R0346

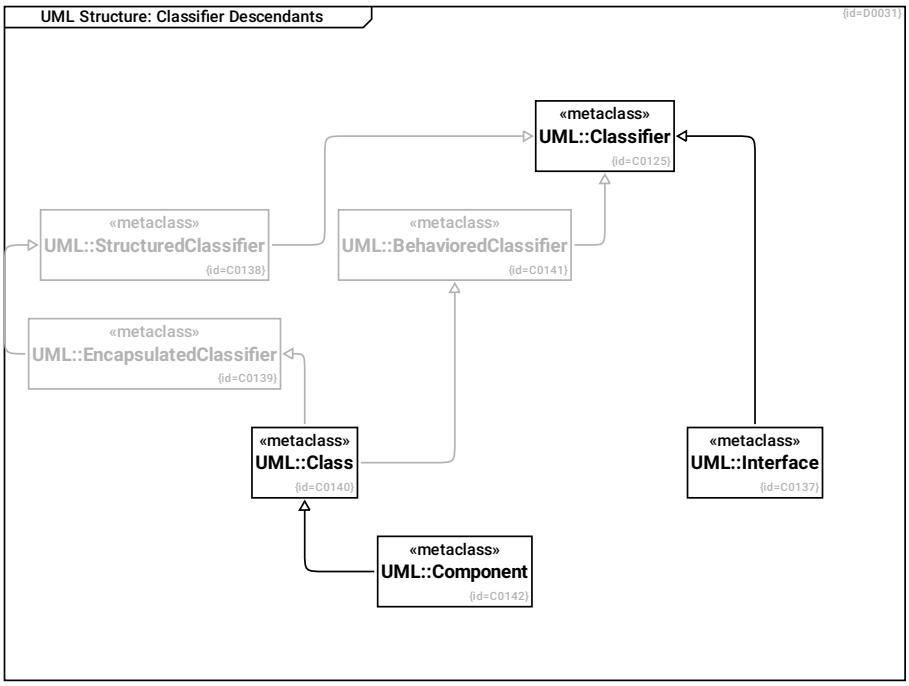
UML::DeploymentTarget C0255

--> UML::NamedElement R0347

UML::Node C0256

--> UML::DeploymentTarget R0348

1.2.1.2.1.2 UML Structure: Classifier Descendants



UML::Classifier C0125

UML::Interface C0137

--> UML::Classifier R0188

UML::StructuredClassifier C0138

--> UML::Classifier R0189

UML::EncapsulatedClassifier C0139

--> UML::StructuredClassifier R0190

UML::Class C0140

--> UML::EncapsulatedClassifier R0191

--> UML::BehavioredClassifier R0192

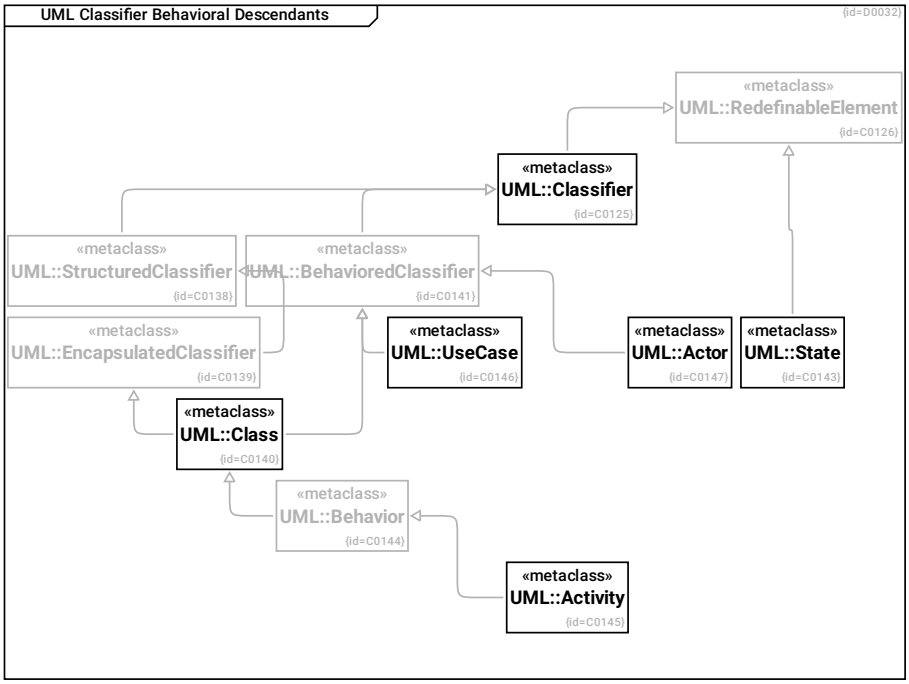
UML::BehavioredClassifier C0141

--> UML::Classifier R0193

UML::Component C0142

--> UML::Class R0194

1.2.1.2.1.3 UML Classifier Behavioral Descendants



UML::Activity C0145

--> UML::Behavior R0196

UML::Behavior C0144

--> UML::Class R0197

UML::Class C0140

--> UML::EncapsulatedClassifier R0191

--> UML::BehavioredClassifier R0192

UML::BehavioredClassifier C0141

--> UML::Classifier R0193

UML::StructuredClassifier C0138

--> UML::Classifier R0189

UML::EncapsulatedClassifier C0139

--> UML::StructuredClassifier R0190

UML::Classifier C0125

--> UML::RedefinableElement R0168

UML::Actor C0147

--> UML::BehavioredClassifier R0199

UML::State C0143

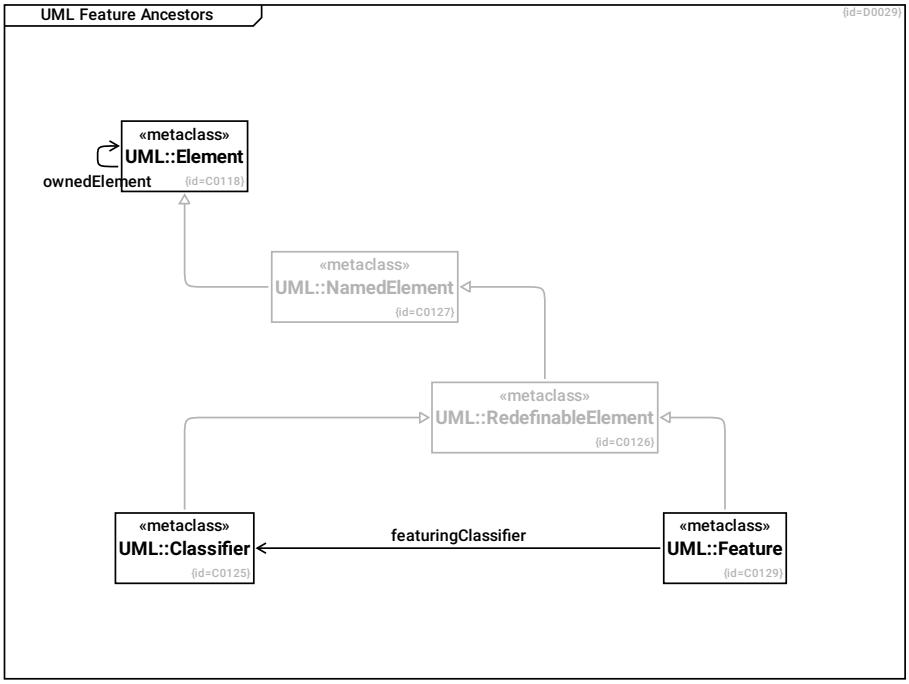
--> UML::RedefinableElement R0195

UML::RedefinableElement C0126
description: see http://www.omg.org/spec/UML/2.5

UML::UseCase C0146

--> UML::BehavioredClassifier R0198

1.2.1.2.2 UML Feature Ancestors



UML::Element C0118

ownedElement --> UML::Element R0161

UML::RedefinableElement C0126

description: see http://www.omg.org/spec/UML/2.5

--> UML::NamedElement R0169

UML::NamedElement C0127

description: see http://www.omg.org/spec/UML/2.5

--> UML::Element R0171

UML::Classifier C0125

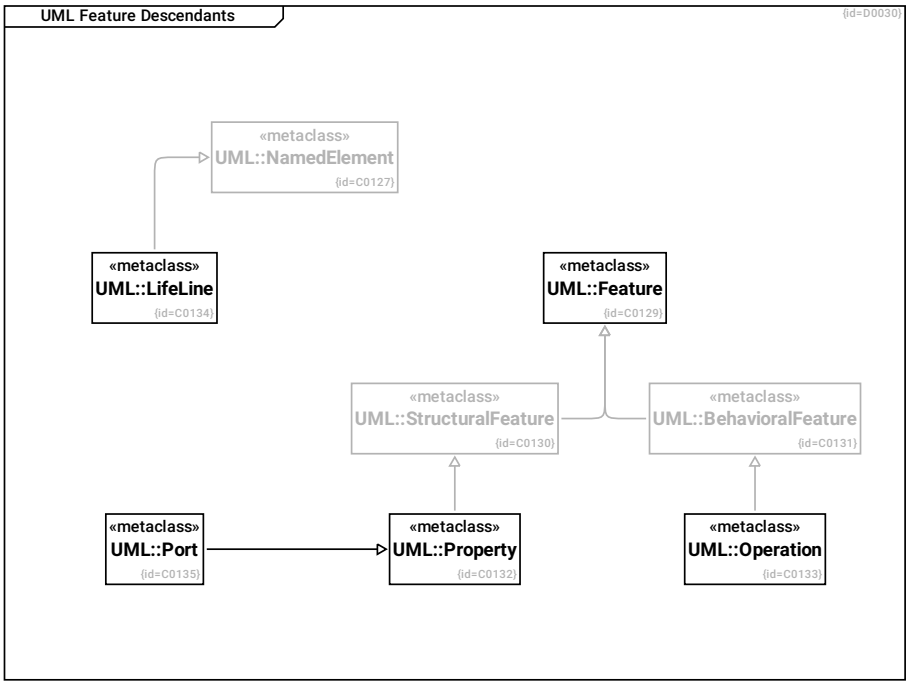
--> UML::RedefinableElement R0168

UML::Feature C0129

--> UML::RedefinableElement R0175

featuringClassifier --> UML::Classifier R0176

1.2.1.2.2.1 UML Feature Descendants



UML::Feature C0129

UML::StructuralFeature C0130

--> UML::Feature R0179

UML::BehavioralFeature C0131

--> UML::Feature R0180

UML::Property C0132

--> UML::StructuralFeature R0182

UML::Operation C0133

--> UML::BehavioralFeature R0181

UML::NamedElement C0127

description: see http://www.omg.org/spec/UML/2.5

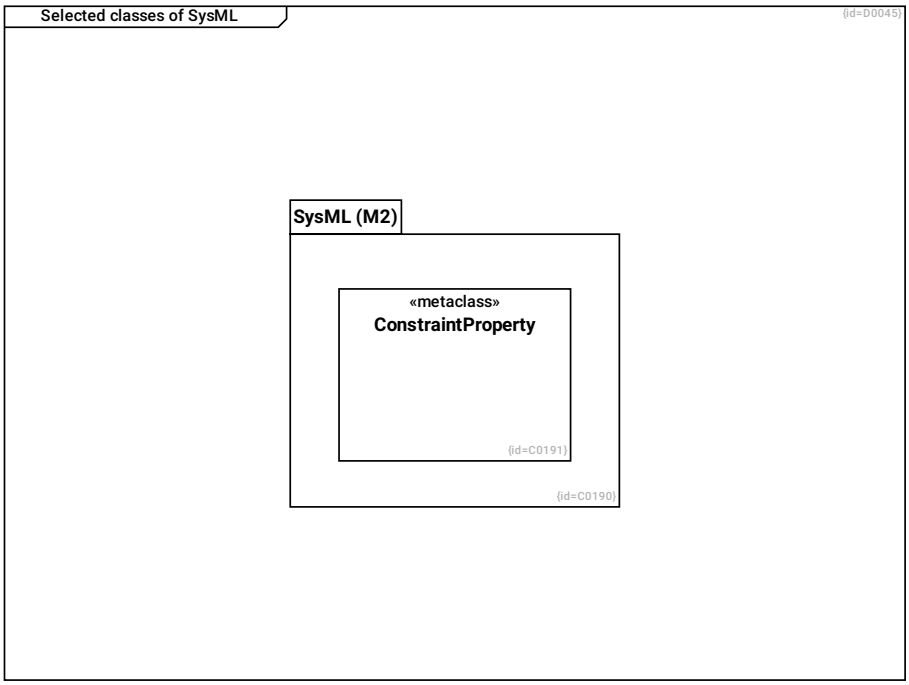
UML::LifeLine C0134

--> UML::NamedElement R0183

UML::Port C0135

--> UML::Property R0184

1.2.1.3 Selected classes of SysML

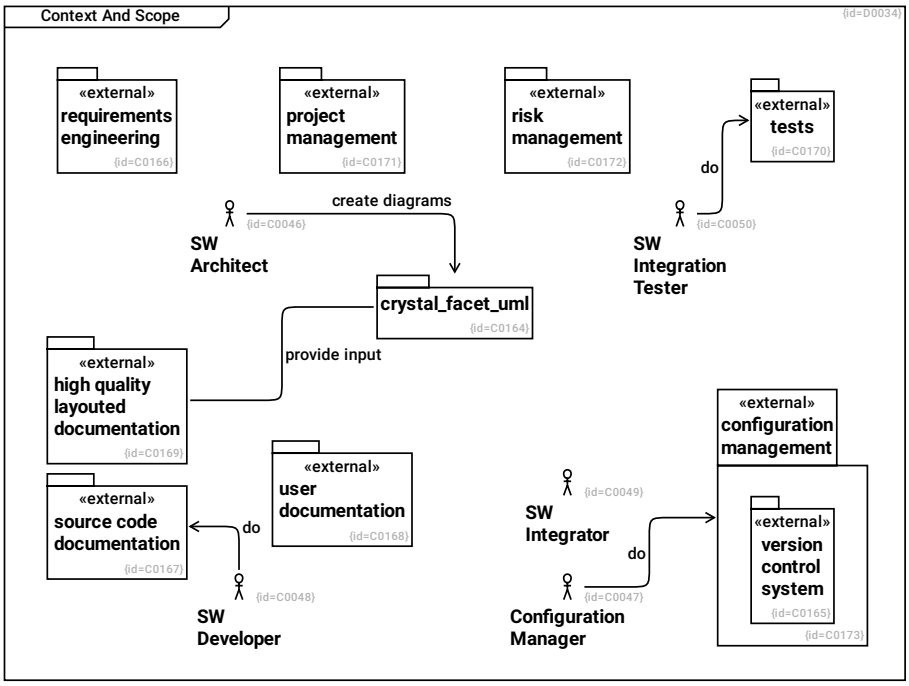


SysML (M2) C0190

--> ConstraintProperty R0250

ConstraintProperty C0191

1.3 Context And Scope



SW Integrator C0049

version control system C0165

SW Architect C0046

create diagrams --> crystal_facet_uml R0232

requirements engineering C0166

crystal_facet_uml C0164

provide input --> high quality layouted documentation R0231

source code documentation C0167

user documentation C0168

configuration management C0173

--> version control system R0229

risk management C0172

project management C0171

tests C0170

unit-tests, integration tests, qualification tests, regression tests

high quality layouted documentation C0169

There are good layout engines to create high quality pdf pages, like:

- latex
- docbook

Configuration Manager C0047

do --> configuration management R0230

SW Developer C0048

A software developer understands the requirements and architecture; creates the software parts and performs reviews and unit-tests.

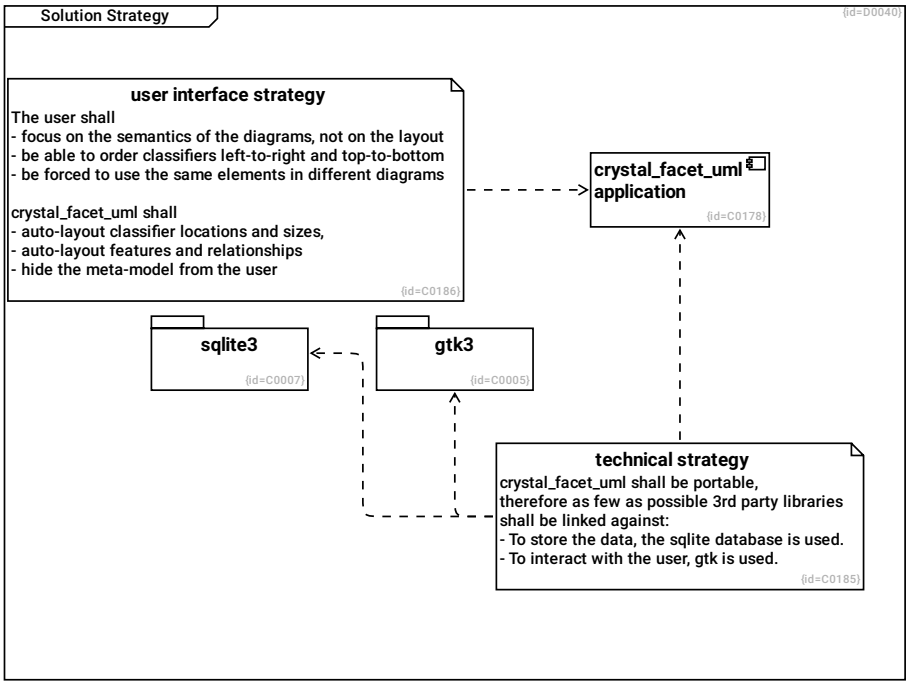
do --> source code documentation R0233

SW Integration Tester C0050

do --> tests R0234

1.4 Solution Strategy

During operation, crystal_facet_uml runs as application based on the libraries sqlite and gtk.



gtk3 C0005

Toolkit which provides

- Windows and Widgets
- User Input handling

user interface strategy C0186

The user shall

- focus on the semantics of the diagrams, not on the layout
- be able to order classifiers left-to-right and top-to-bottom
- be forced to use the same elements in different diagrams

crystal_facet_uml shall

- auto-layout classifier locations and sizes,
- auto-layout features and relationships
- hide the meta-model from the user

--> crystal_facet_uml application R0247

crystal_facet_uml application C0178

technical strategy C0185

crystal_facet_uml shall be portable, therefore as few as possible 3rd party libraries shall be linked against:

- To store the data, the sqlite database is used.
- To interact with the user, gtk is used.

--> crystal_facet_uml application R0244

--> gtk3 R0245

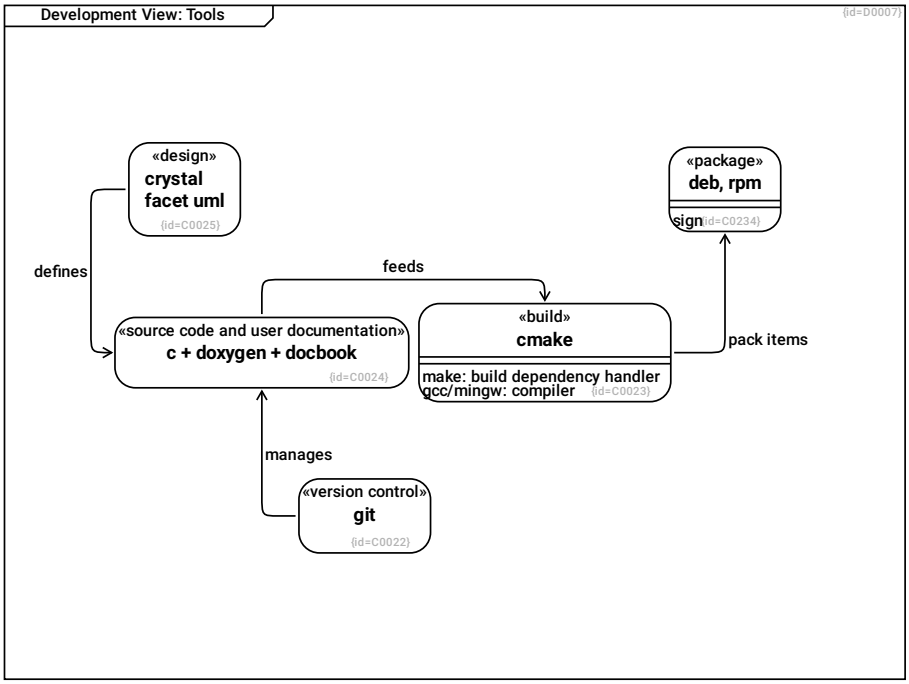
--> sqlite3 R0246

sqlite3 C0007

SQL database

1.4.1 Development View: Tools

This diagram shows the tools for designing, implementing, documenting, configuring and building the software.



git C0022

version control system

manages --> c + doxygen + docbook R0054

cmake C0023

cmake and cpack are meta-build files which generate e.g. make files, rpm and deb package files.

make: build dependency handler F0001

gcc/mingw: compiler F0003

pack items --> deb, rpm R0312

c + doxygen + docbook C0024

c code and doxygen are the source code and documentation formats

feeds --> cmake R0007

crystal facet uml C0025

- drafts the software design (using this crystal facet uml tool which is under construction)

defines --> c + doxygen + docbook R0006

deb, rpm C0234

sign F0057

1.5 Building Block View

This diagram shows the layering of the main software modules.

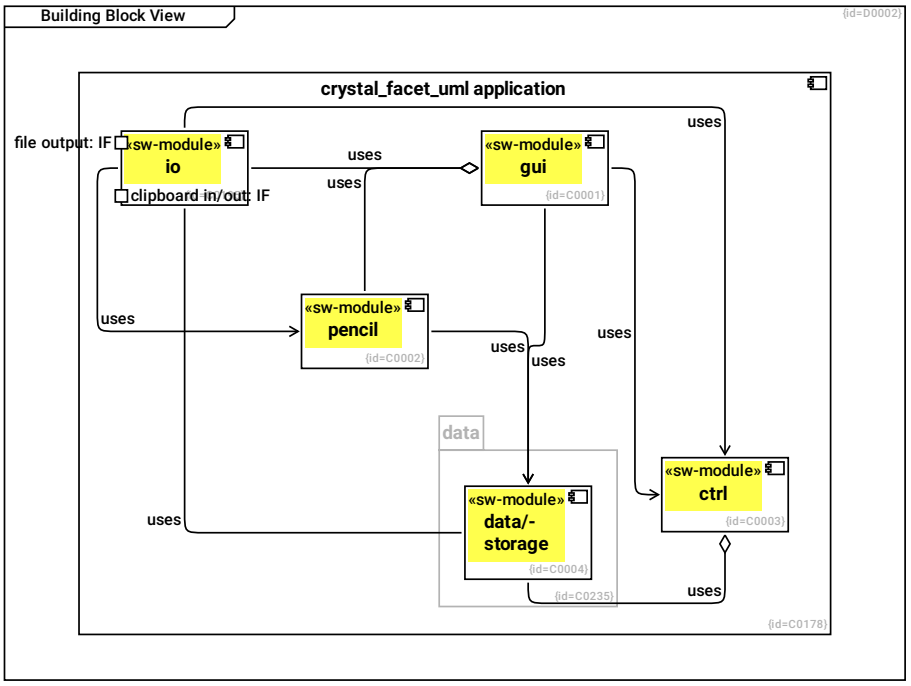
Concept for Multithreading: In general, all modules shall work in a multithreaded environment.

- The gui module requires own data structures in order to be thread-safe and relies on the base libraries gtk, gdk which need to support multi-threading,

- The pencil module requires own data structures in order to be thread safe and relies on the base libraries pango and cairo which need to support multi-threading.

- The data module does not need own data structures, it comes with a module-wide lock in the database_t structure.

- The ctrl module requires own data structures in order to be thread safe. This limitation would need to be changed if a multi-threaded gui should access a single ctrl and data instance. (Currently, the software is single-threaded)



gui C0001

- allows a user - to select the database - to export diagrams - to modify the uml-model

uses --> pencil R0008

uses --> ctrl R0009

uses --> data/storage R0010

uses --> io R0256

ctrl C0003

- provides write access to the database
- ensures consistency of the uml-model, e.g. no circular contains-relations, no lifelines if no corresponding diagramelement exists

uses --> data/storage R0012

data/storage C0004

- provides read access to ctrl, pencil and gui components
- provides write acccess to only the ctrl component
- ensures low-level consistency of database: all referenced objects exist.
- sends notifications on changes

crystal_facet_uml application C0178

--> gui R0239

--> pencil R0240

--> ctrl R0241

--> data/storage R0242

--> io R0255

--> data R0314

io C0195

file output: IF F0075

clipboard in/out: IF F0074

uses --> pencil R0257

uses --> data/storage R0258

uses --> ctrl R0392

pencil C0002

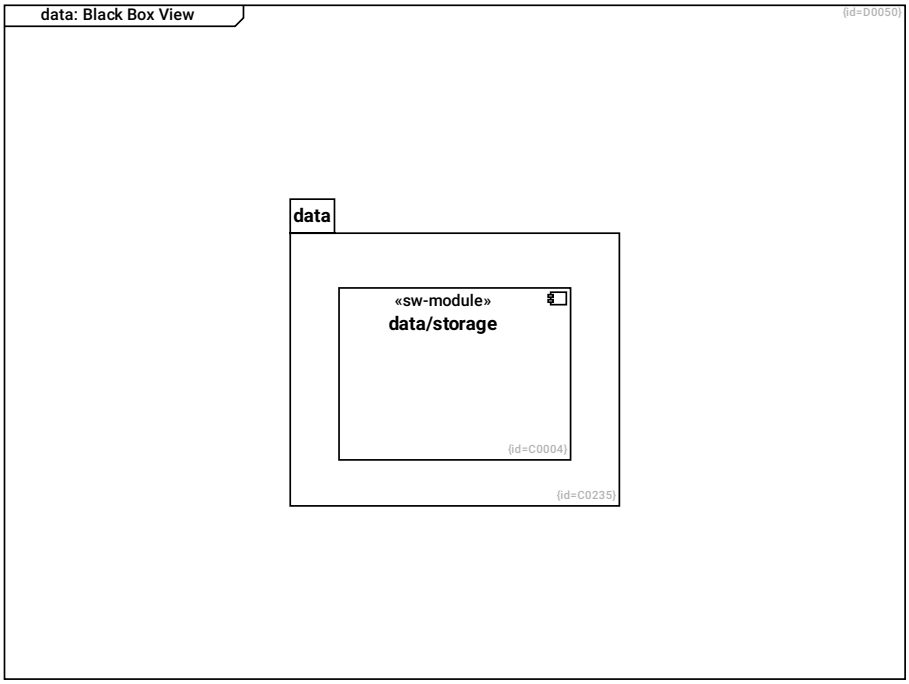
- renders diagrams to cairo drawing contexts
- highlights uml-elements as requested by the gui module

uses --> data/storage R0011

data C0235

--> data/storage R0313

1.5.1 data: Black Box View



data/storage C0004

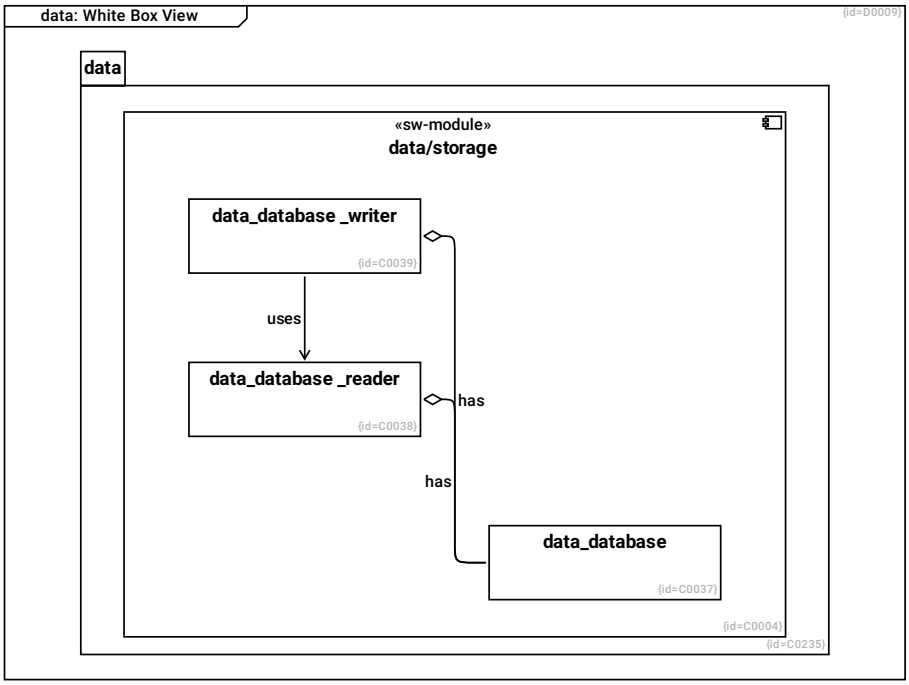
- provides read access to ctrl, pencil and gui components
- provides write acccess to only the ctrl component
- ensures low-level consistency of database: all referenced objects exist.
- sends notifications on changes

data C0235

--> data/storage R0313

1.5.1.1 data: White Box View

This diagram shows the main classes in the data package.



data/storage C0004

- provides read access to ctrl, pencil and gui components
- provides write acccess to only the ctrl component
- ensures low-level consistency of database: all referenced objects exist.
- sends notifications on changes

--> data_database _writer R0040

--> data_database _reader R0041

--> data_database R0042

data_database C0037

- wraps the sqlite database
- knows the current status and filename
- opens and closes the database
- manages the list of listeners and notifies changes

data_database _writer C0039

- writes changes to the database
- ensures low-level consistency of the database, e.g. all foreign keys point to existing rows (referential integrity)
- retrieves the old, changed values for the undo/redo list

uses --> data_database _reader R0027

has --> data_database R0029

data_database _reader C0038

- reads data from the database

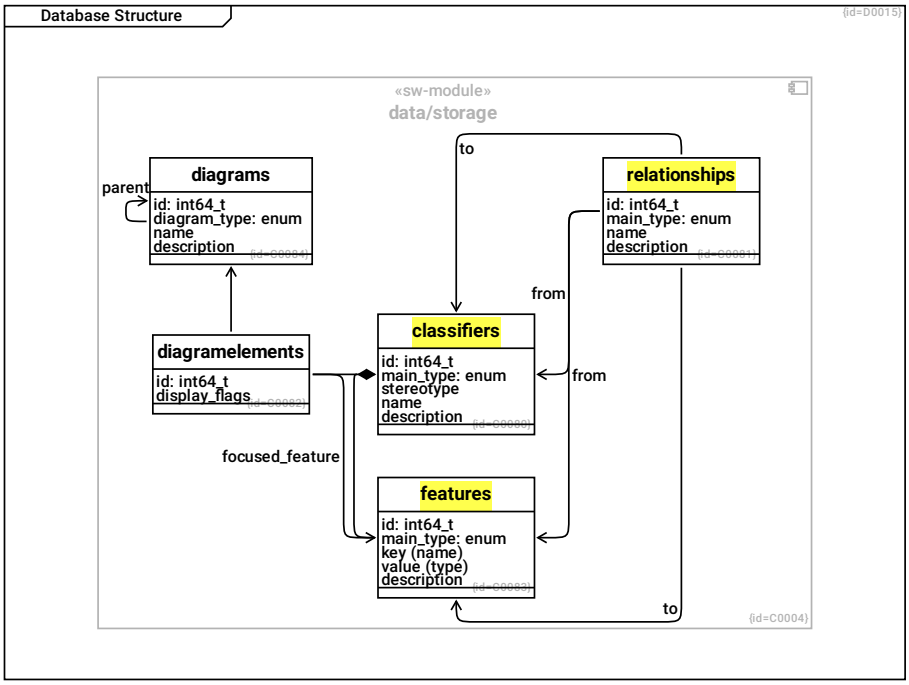
has --> data_database R0028

data C0235

--> data/storage R0313

1.5.1.1.1 Database Structure

This diagram shows the tables and relationships of the database.



classifiers C0080

id: int64_t F0012

main_type: enum F0017

stereotype F0018

name F0019

description F0020

--> features R0210

features C0083

id: int64_t F0013

main_type: enum F0024

key (name) F0025

value (type) F0026

description F0027

diagrams C0084

id: int64_t F0015

diagram_type: enum F0028

name F0029

description F0030

parent --> diagrams R0099

data/storage C0004

- provides read access to ctrl, pencil and gui components
- provides write acccess to only the ctrl component
- ensures low-level consistency of database: all referenced objects exist.
- sends notifications on changes

--> diagrams R0266

--> diagramelements R0267

--> classifiers R0268

--> relationships R0269

--> features R0270

relationships C0081

id: int64_t F0014

main_type: enum F0021

name F0022

description F0023

from --> classifiers R0096

to --> classifiers R0097

from --> features R0205

to --> features R0206

diagramelements C0082

id: int64_t F0016

display_flags F0031

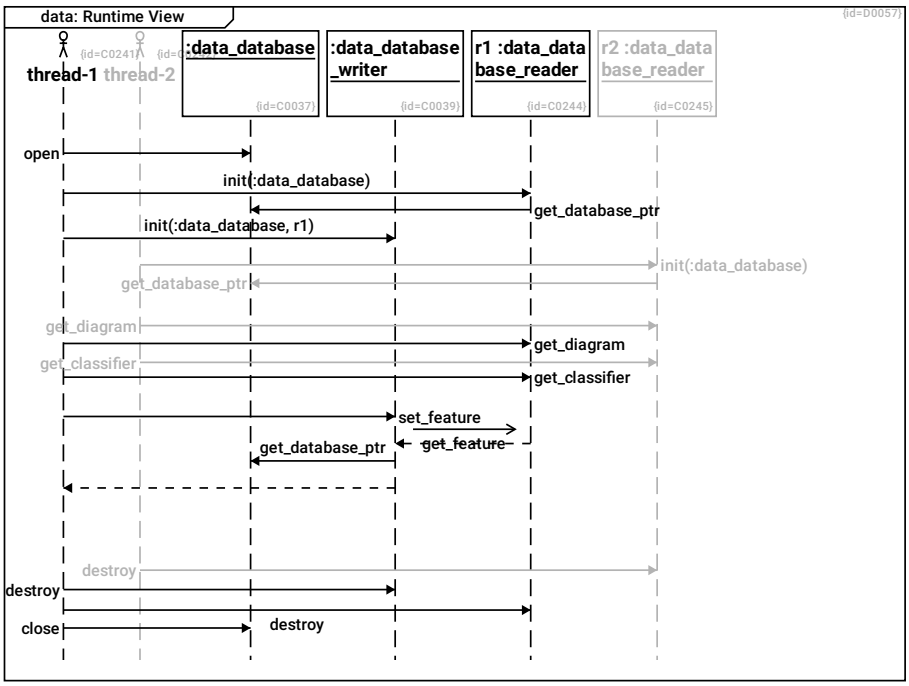
--> diagrams R0100

--> classifiers R0101

focused_feature --> features R0204

to select one of possibly several lifelines

1.5.1.2 data: Runtime View



thread-1 C0241

open --> data_database R0320

init(:data_database) --> r1 :data_data base_reader R0326

init(:data_database, r1) --> data_database _writer R0324

get_diagram --> r1 :data_data base_reader R0332

get_classifier --> r1 :data_data base_reader R0333

set_feature --> data_database _writer R0334

destroy --> data_database _writer R0325

destroy --> r1 :data_data base_reader R0329

close --> data_database R0321

thread-2 C0242

init(:data_database) --> r2 :data_data base_reader R0327

get_diagram --> r2 :data_data base_reader R0330

get_classifier --> r2 :data_data base_reader R0331

destroy --> r2 :data_data base_reader R0328

data_database C0037

- wraps the sqlite database
- knows the current status and filename
- opens and closes the database
- manages the list of listeners and notifies changes

data_database _writer C0039

- writes changes to the database
- ensures low-level consistency of the database, e.g. all foreign keys point to existing rows (referential integrity)
- retrieves the old, changed values for the undo/redo list

get_feature --> r1 :data_data base_reader R0335

get_database_ptr --> data_database R0340

--> thread-1 R0337

r1 :data_data base_reader C0244

get_database_ptr --> data_database R0338

--> data_database _writer R0336

r2 :data_data base_reader C0245

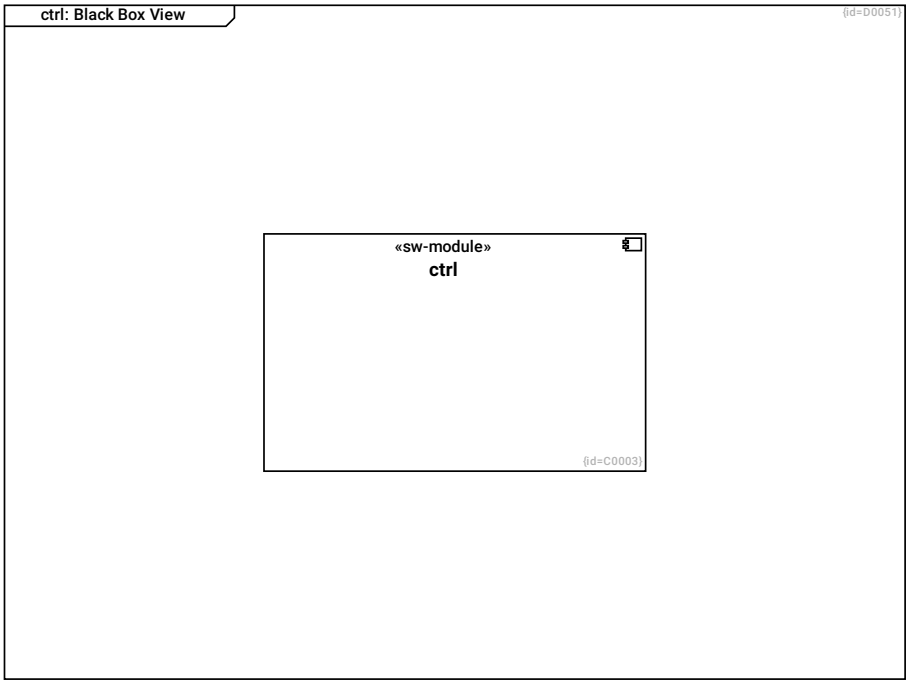
In the current implementation, the software runs single-threaded.

If running multiple threads, a possible implementation is that the data_database object exists just once and provides an sqlite3 connection object in SQLITE_CONFIG_SERIALIZED mode (see https://sqlite.org/threadsafe.html) Every thread gets its own readers and writers (to separate the statements-with-bound-variables and statement-string-buffers)

There is a chance that thread-2 performs multiple independant queries while thread-1 performs a write transaction. This is expected to not cause a problem, given the current sequences of statements.

get_database_ptr --> data_database R0339

1.5.2 ctrl: Black Box View



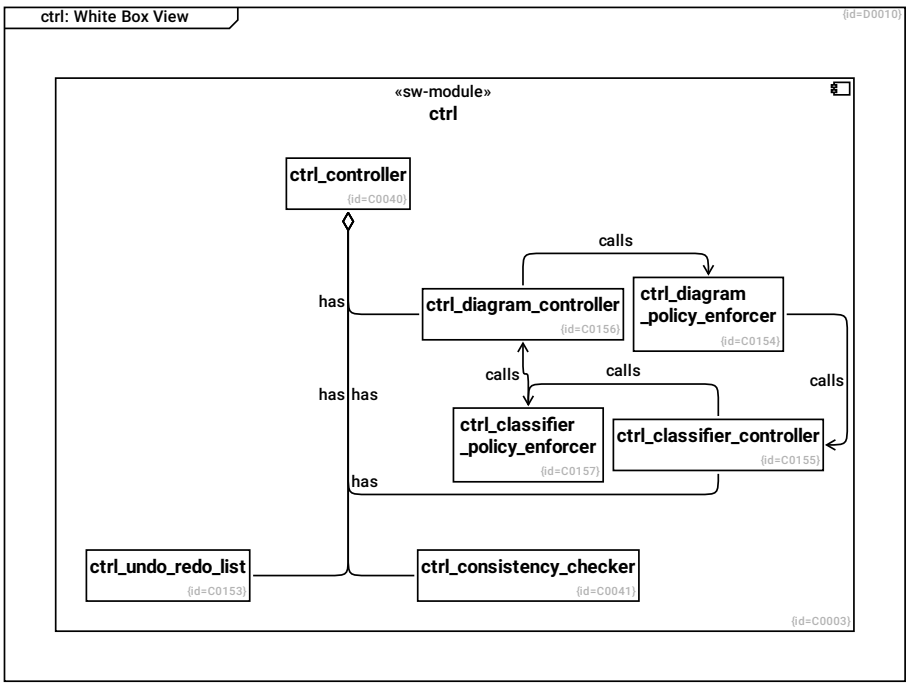
ctrl C0003

- provides write access to the database

- ensures consistency of the uml-model, e.g. no circular contains-relations, no lifelines if no corresponding diagramelement exists

1.5.2.1 ctrl: White Box View

This diagram shows the main classes in the ctrl package.



ctrl_controller C0040

- changes the database contents
- maintains the undo redo list

has --> ctrl_consistency_checker R0030

has --> ctrl_undo_redo_list R0213

has --> ctrl_classifier_controller R0216

has --> ctrl_diagram_controller R0218

ctrl_consistency_checker C0041

- checks and repairs a database

ctrl C0003

- provides write access to the database
- ensures consistency of the uml-model, e.g. no circular contains-relations, no lifelines if no corresponding diagramelement exists

--> ctrl_controller R0043

--> ctrl_consistency_checker R0044

--> ctrl_classifier _policy_enforcer R0221

--> ctrl_diagram_controller R0217

--> ctrl_classifier_controller R0215

--> ctrl_undo_redo_list R0211

--> ctrl_diagram _policy_enforcer R0212

ctrl_undo_redo_list C0153

manages a list of actions to be undone or/and redone.

ctrl_diagram _policy_enforcer C0154

ensures that application level policies on the model are adhered whenever diagrams are modified.

calls --> ctrl_classifier_controller R0220

ctrl_classifier_controller C0155

performs changes on classifier, feature and relationship tables

calls --> ctrl_classifier _policy_enforcer R0222

ctrl_diagram_controller C0156

performs changes on diagram and diagramelement tables

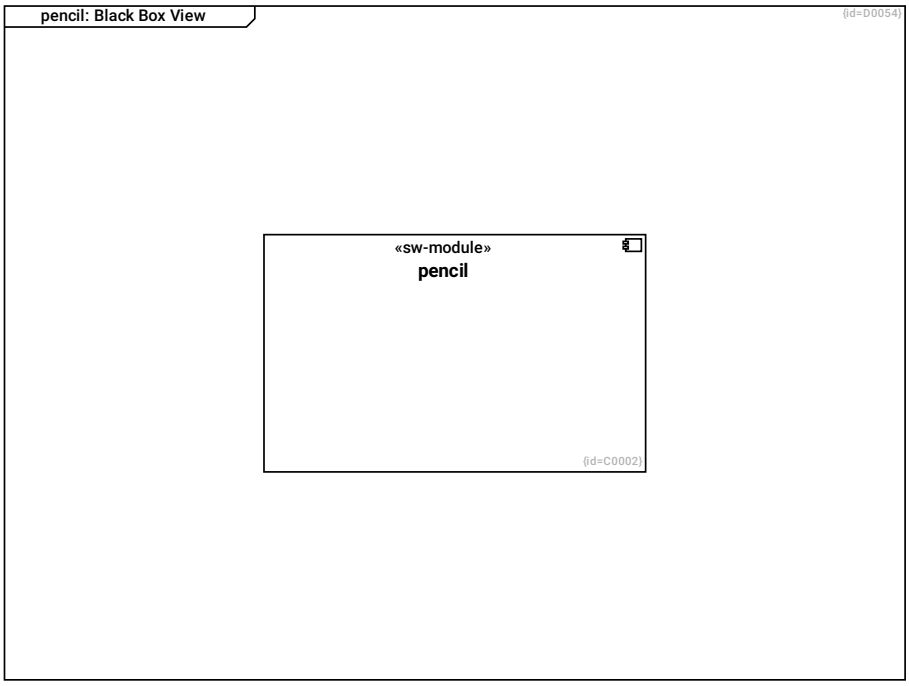
calls --> ctrl_diagram _policy_enforcer R0219

ctrl_classifier _policy_enforcer C0157

ensures that application level policies on the model are adhered whenever classifiers are modified.

calls --> ctrl_diagram_controller R0223

1.5.3 pencil: Black Box View

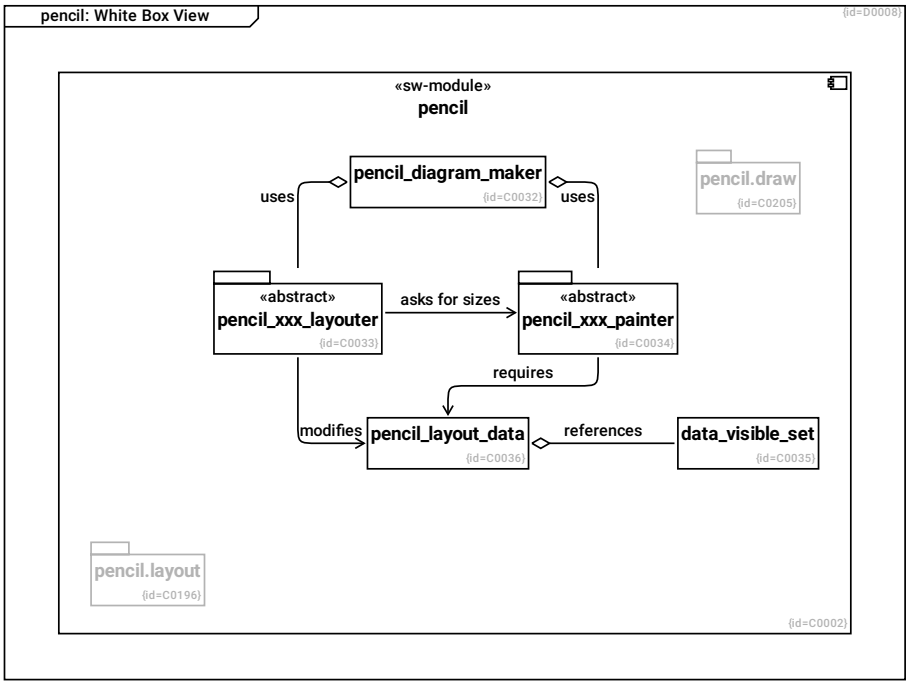


pencil C0002

- renders diagrams to cairo drawing contexts
- highlights uml-elements as requested by the gui module

1.5.3.1 pencil: White Box View

This diagram shows the main classes in the pencil package.



pencil_diagram_maker C0032

- coordinates layouting and drawing of a diagram
- implements kind of front controller pattern: all requests from gui to pencil are addressed to the pencil_diagram_maker

uses --> pencil_xxx_layouter R0023

uses --> pencil_xxx_painter R0024

pencil_xxx_layouter C0033

- layouts multiple elements within the bounds of a diagram
- can convert abstract order-values to x/y coordinates and vice versa

modifies --> pencil_layout_data R0022

asks for sizes --> pencil_xxx_painter R0034

data_visible_set C0035

- caches data from the database needed to draw a diagram

pencil_layout_data C0036

- stores locations of layouted elements (in the current diagram)

references --> data_visible_set R0021

pencil.layout C0196

pencil.draw C0205

Encapulates drawing routines for
- geometric objects
- labels
- symbols/icons

pencil_xxx_painter C0034

- knows the bounding boxes of single elements
- draws single graphical elements

requires --> pencil_layout_data R0025

pencil C0002

- renders diagrams to cairo drawing contexts
- highlights uml-elements as requested by the gui module

--> pencil.draw R0282

--> pencil_diagram_maker R0049

--> pencil_xxx_painter R0051

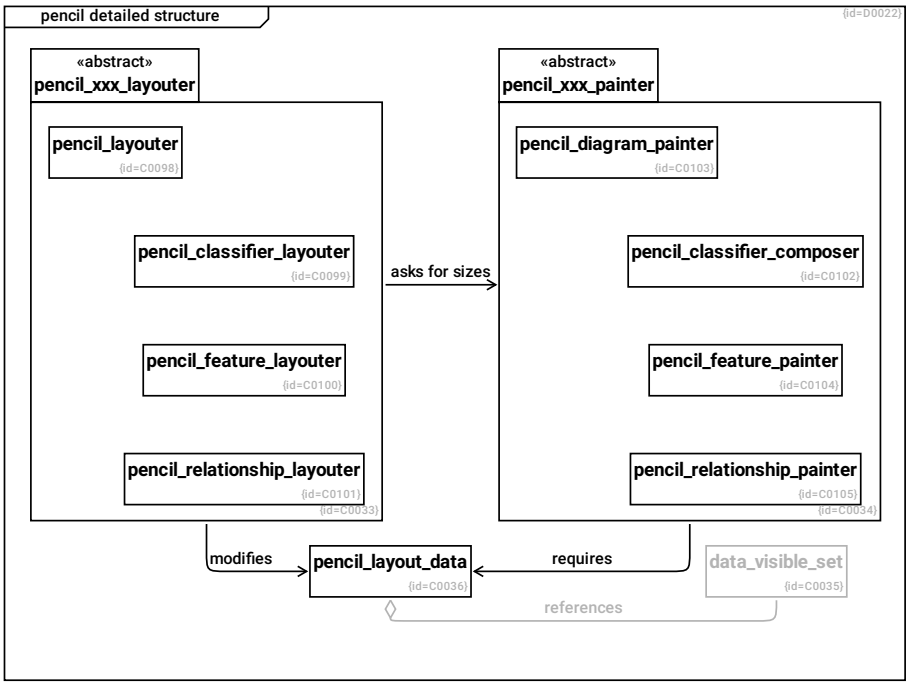
--> data_visible_set R0052

--> pencil_layout_data R0053

--> pencil_xxx_layouter R0056

--> pencil.layout R0265

1.5.3.1.1 pencil detailed structure



pencil_xxx_layouter C0033

- layouts multiple elements within the bounds of a diagram
- can convert abstract order-values to x/y coordinates and vice versa

--> pencil_classifier_layouter R0133

--> pencil_relationship_layouter R0135

--> pencil_layouter R0132

--> pencil_feature_layouter R0134

modifies --> pencil_layout_data R0022

asks for sizes --> pencil_xxx_painter R0034

pencil_layout_data C0036

- stores locations of layouted elements (in the current diagram)

references --> data_visible_set R0021

data_visible_set C0035

- caches data from the database needed to draw a diagram

pencil_layouter C0098

- layouts multiple elements within the bounds of a diagram
- layouting is implemented in two steps: 1) defining a grid which associates abstract ordering data to diagram coordinates and 2) assigning elements to coordinates

pencil_classifier_layouter C0099

pencil_feature_layouter C0100

pencil_relationship_layouter C0101

pencil_classifier_composer C0102

pencil_diagram_painter C0103

pencil_feature_painter C0104

pencil_relationship_painter C0105

pencil_xxx_painter C0034

- knows the bounding boxes of single elements
- draws single graphical elements

--> pencil_diagram_painter R0137

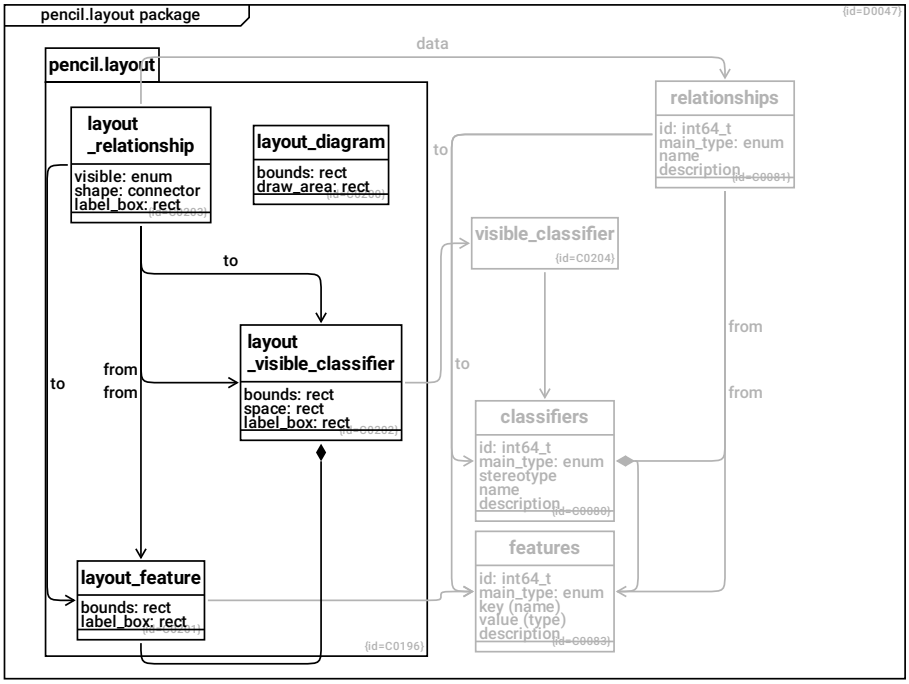
--> pencil_feature_painter R0138

--> pencil_relationship_painter R0139

--> pencil_classifier_composer R0136

requires --> pencil_layout_data R0025

1.5.3.1.2 pencil.layout package



classifiers C0080

id: int64_t F0012

main_type: enum F0017

stereotype F0018

name F0019

description F0020

--> features R0210

features C0083

id: int64_t F0013

main_type: enum F0024

key (name) F0025

value (type) F0026

description F0027

layout_feature C0201

bounds: rect F0049

label_box: rect F0053

--> features R0281

layout _visible_classifier C0202

bounds: rect F0048

space: rect F0050

label_box: rect F0052

--> visible_classifier R0280

--> layout_feature R0294

pencil.layout C0196

--> layout_feature R0262

--> layout _visible_classifier R0263

--> layout_diagram R0261

--> layout _relationship R0264

layout_diagram C0200

bounds: rect F0044

draw_area: rect F0045

layout _relationship C0203

visible: enum F0046

shape: connector F0047

label_box: rect F0051

data --> relationships R0272

from --> layout _visible_classifier R0273

to --> layout _visible_classifier R0274

from --> layout_feature R0275

to --> layout_feature R0293

relationships C0081

id: int64_t F0014

main_type: enum F0021

name F0022

description F0023

from --> classifiers R0096

to --> classifiers R0097

from --> features R0205

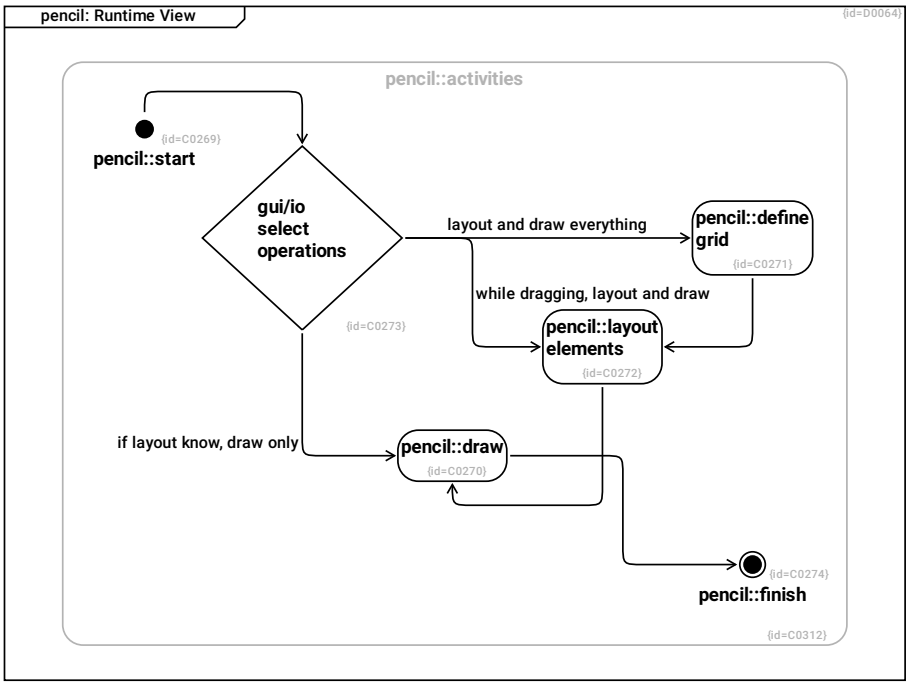
to --> features R0206

visible_classifier C0204

--> classifiers R0278

1.5.3.2 pencil: Runtime View

This diagram shows the different entry points for layouting and rendering a diagram.



pencil::start C0269

--> gui/io select operations R0371

pencil::activities C0312

--> gui/io select operations R0440

--> pencil::start R0441

--> pencil::draw R0442

--> pencil::layout elements R0443

--> pencil::define grid R0444

--> pencil::finish R0445

pencil::define grid C0271
implemented in pencil_diagram_maker

--> pencil::layout elements R0375

gui/io select operations C0273

layout and draw everything --> pencil::define grid R0372

while dragging, layout and draw --> pencil::layout elements R0373

if layout know, draw only --> pencil::draw R0374

pencil::layout elements C0272
implemented in pencil_diagram_maker

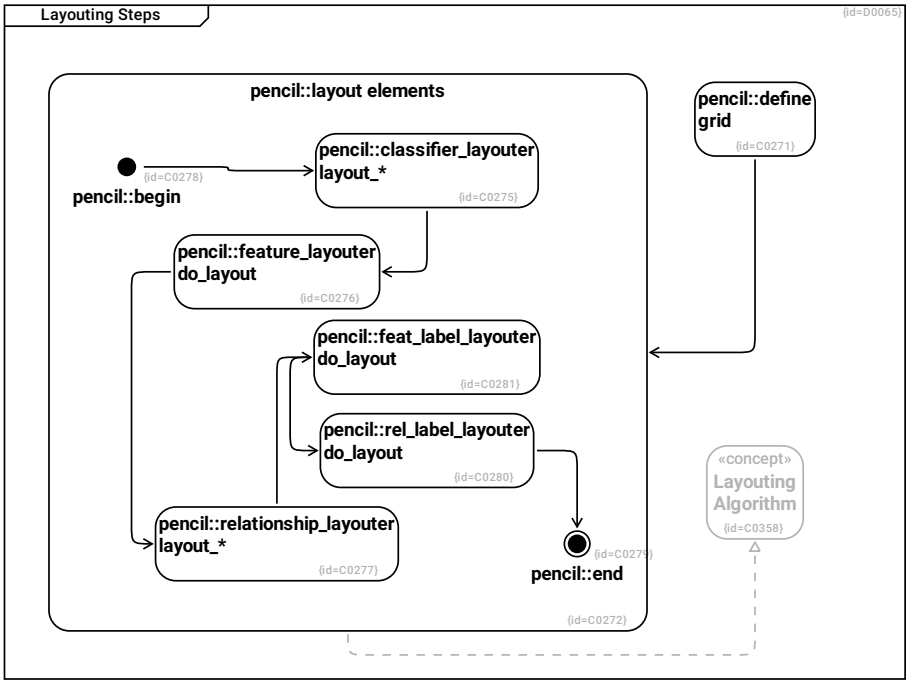
--> pencil::draw R0376

pencil::draw C0270

--> pencil::finish R0377

pencil::finish C0274

1.5.3.2.1 Layouting Steps



Layouting Algorithm C0358

pencil::define grid C0271

implemented in pencil_diagram_maker

--> pencil::layout elements R0375

pencil::begin C0278

--> pencil::classifier_layouter layout_* R0383

pencil::classifier_layouter layout_* C0275

--> pencil::feature_layouter do_layout R0384

pencil::feature_layouter do_layout C0276

--> pencil::relationship_layouter layout_* R0385

pencil::feat_label_layouter do_layout C0281

--> pencil::rel_label_layouter do_layout R0390

pencil::relationship_layouter layout_* C0277

--> pencil::feat_label_layouter do_layout R0389

pencil::rel_label_layouter do_layout C0280

--> pencil::end R0391

pencil::end C0279

pencil::layout elements C0272
implemented in pencil_diagram_maker

--> pencil::begin R0381

--> pencil::classifier_layouter layout_* R0378

--> pencil::feature_layouter do_layout R0379

--> pencil::feat_label_layouter do_layout R0388

--> pencil::relationship_layouter layout_* R0380

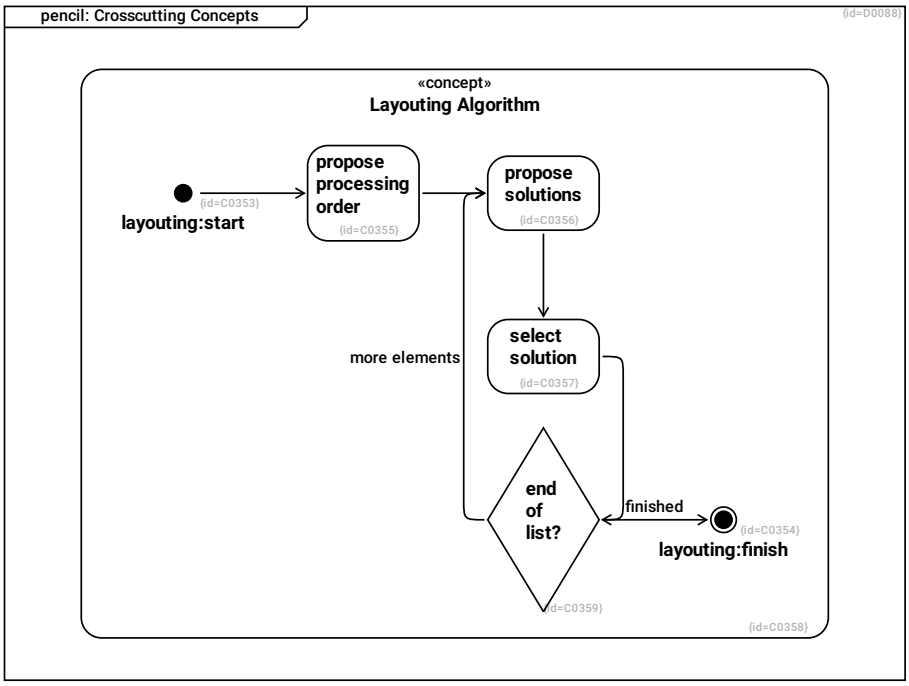
--> pencil::rel_label_layouter do_layout R0387

--> pencil::end R0382

--> Layouting Algorithm R0507

1.5.3.3 pencil: Crosscutting Concepts

Whenever one of several layouting solutions shall be selected, the following algorithm applies.



Layouting Algorithm C0358

--> end of list? R0508

--> propose processing order R0502

--> propose solutions R0503

--> select solution R0504

--> layouting:finish R0505

--> layouting:start R0506

propose processing order C0355

For the list of elements to be layouted, define the processing order.

--> propose solutions R0501

select solution C0357

This step selects the most promising of solutions for 1 element to be layouted.

--> end of list? R0509

propose solutions C0356

This step creates a set of possible solutions for 1 element to be layouted.

--> select solution R0499

end of list? C0359

finished --> layouting:finish R0510

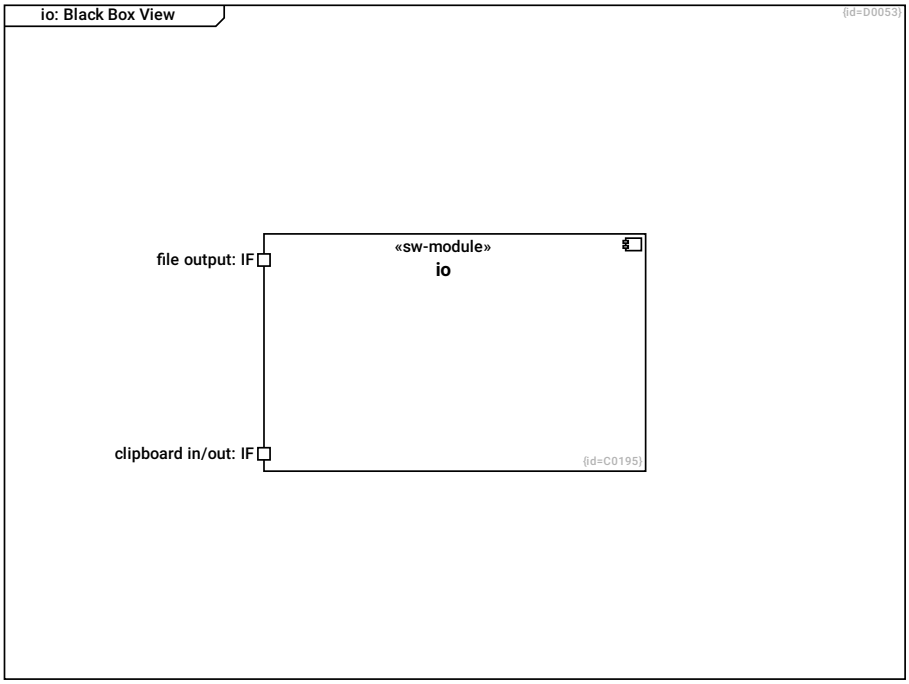
more elements --> propose solutions R0511

layouting:finish C0354

layouting:start C0353

--> propose processing order R0498

1.5.4 io: Black Box View



io C0195

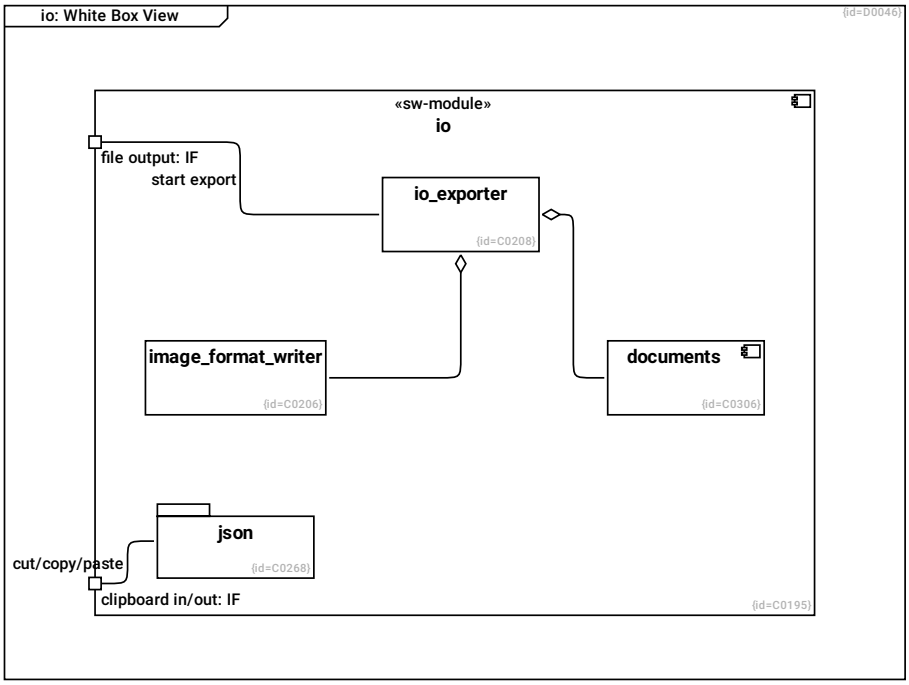
file output: IF F0075

clipboard in/out: IF F0074

1.5.4.1 io: White Box View

This component implements input output functionality, currently:

- Export of diagram images



image_format_writer C0206

documents C0306

json C0268

io_exporter C0208

cases about
- file names
- traversing the diagram tree

--> image_format_writer R0287

--> documents R0423

io C0195

file output: IF F0075

clipboard in/out: IF F0074

--> image_format_writer R0283

--> documents R0419

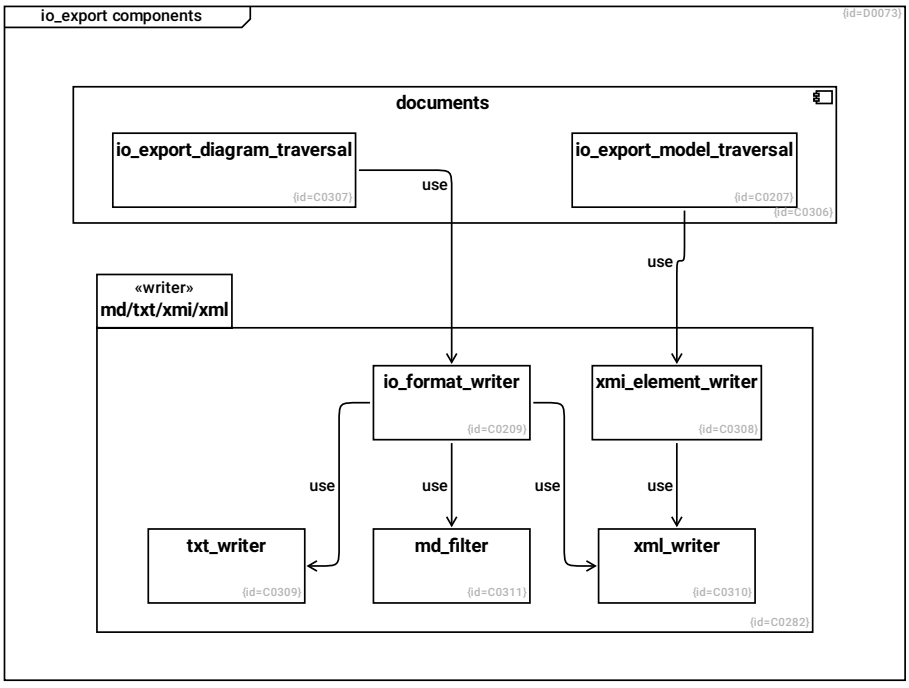
--> io_exporter R0285

--> json R0369

start export --> io_exporter R0368

cut/copy/paste --> json R0370

1.5.4.1.1 io_export components



io_export_model_traversal C0207
cares about
- selecting the model elements to export

use --> xmi_element_writer R0426

documents C0306

--> io_export_diagram_traversal R0420

--> io_export_diagram_traversal R0422

--> io_export_model_traversal R0421

xmi_element_writer C0308

use --> xml_writer R0431

md_filter C0311

xml_writer C0310

txt_writer C0309

io_export_diagram_traversal C0307

use --> io_format_writer R0427

io_format_writer C0209
cares about
- writing output-format specific meta data
- using right "writer" class to encode data

use --> txt_writer R0432

use --> md_filter R0433

use --> xml_writer R0434

md/txt/xmi/xml C0282

--> xmi_element_writer R0425

--> md_filter R0430

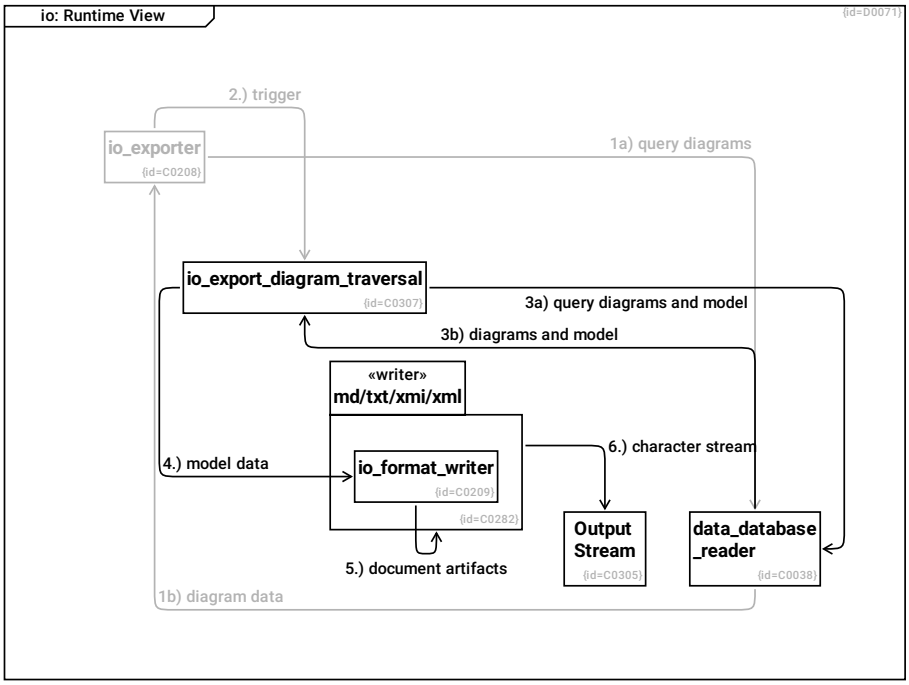
--> xml_writer R0429

--> txt_writer R0428

--> io_format_writer R0424

1.5.4.2 io: Runtime View

The io_exporter sets up the data pipe



io_exporter C0208

cases about

- file names
- traversing the diagram tree

1a) query diagrams --> data_database _reader R0416

2.) trigger --> io_export_diagram_traversal R0435

data_database _reader C0038

- reads data from the database

1b) diagram data --> io_exporter R0417

3b) diagrams and model --> io_export_diagram_traversal R0438

io_export_diagram_traversal C0307

3a) query diagrams and model --> data_database _reader R0436

4.) model data --> io_format_writer R0437

io_format_writer C0209

cares about
- writing output-format specific meta data
- using right "writer" class to encode data

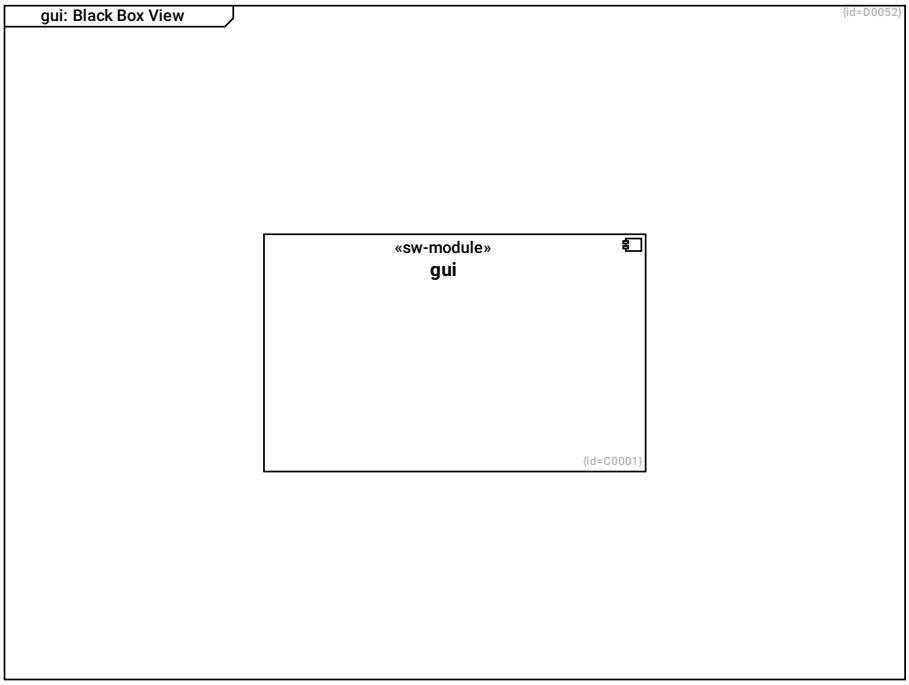
5.) document artifacts --> md/txt/xmi/xml R0413

md/txt/xmi/xml C0282

6.) character stream --> Output Stream R0414

Output Stream C0305

1.5.5 gui: Black Box View

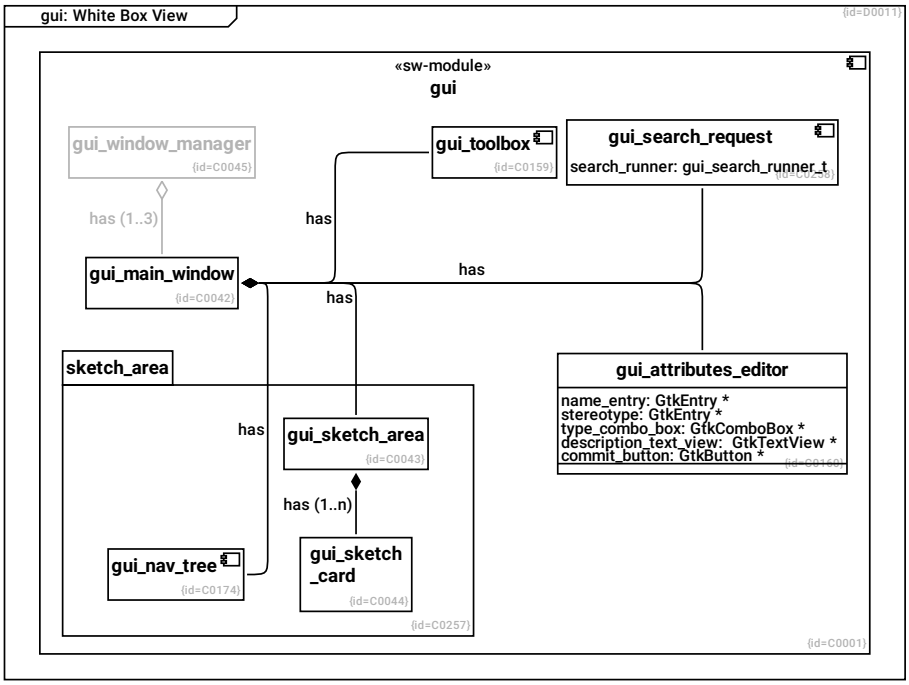


gui C0001

- allows a user - to select the database - to export diagrams - to modify the uml-model

1.5.5.1 gui: White Box View

This diagram shows the main classes in the gui package.



gui_main_window C0042

- represents an application window

has --> gui_sketch_area R0031

has --> gui_toolbox R0226

has --> gui_attributes_editor R0228

has --> gui_nav_tree R0236

--> gui_search_request R0354

gui_window_manager C0045

- starts and stops application windows

has (1..3) --> gui_main_window R0033

gui C0001

- allows a user - to select the database - to export diagrams - to modify the uml-model

--> gui_search_request R0353

--> sketch_area R0349

--> gui_toolbox R0225

--> gui_attributes_editor R0227

--> gui_window_manager R0045

--> gui_main_window R0046

--> gui_sketch _card R0047

--> gui_sketch_area R0048

--> gui_nav_tree R0235

gui_search_request C0258

This class manages

- a text entry widget so that the user can enter a search text

- a button that starts the search.

When a search is started, the gui_search_runner is triggered to execute it.

search_runner: gui_search_runner_t F0083

gui_toolbox C0159

controller for the buttons in the tool bar:

- performs the callbacks when the user presses buttons

- switches between search, nav, edit and create mode

gui_attributes_editor C0160

This class manages 5 widgets.

It reads the data of the currently focused object into cache, it writes back changes to the database (via the controller), it updates the visible widgets when the focused object or the data changes.

A challenge for implementation is, that on one hand, it implements the callback functions for 5 gtk widgets (which look independant if ignoring the internal implementation), on the other hand, it manages just one cached focused object (update on user-interaction, reload on change events).

name_entry: GtkEntry * F0039

stereotype: GtkEntry * F0042

type_combo_box: GtkComboBox * F0040

description_text_view: GtkTextView * F0041

commit_button: GtkButton * F0043

gui_sketch_area C0043

Manages the main drawing area:

- layouts its subwidgets,

- loads data-sets from the database,

- reacts on key+button events,

- handles data-change events,

- controls selection-set

- encapsulates quite complex logic for mouse events

has (1..n) --> gui_sketch _card R0032

gui_nav_tree C0174

sketch_area C0257

--> gui_sketch_area R0350

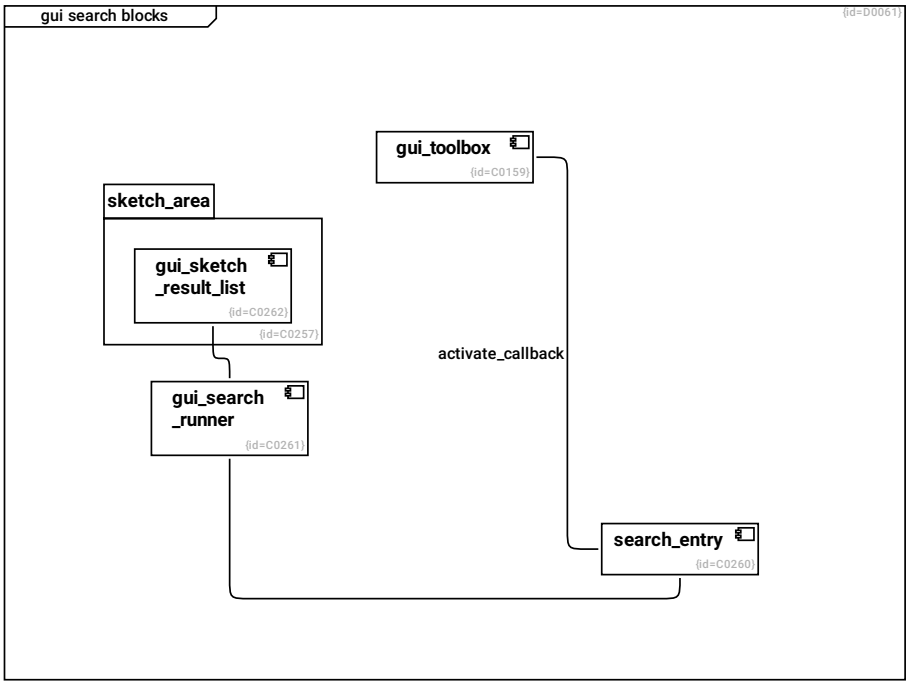
--> gui_sketch _card R0351

--> gui_nav_tree R0352

gui_sketch _card C0044

- maintains the data of one diagram
- displays one diagram

1.5.5.1.1 gui search blocks



search_entry C0260

--> gui_search _runner R0355

gui_toolbox C0159

controller for the buttons in the tool bar:

- performs the callbacks when the user presses buttons
- switches between search, nav, edit and create mode

activate_callback --> search_entry R0358

gui_search _runner C0261

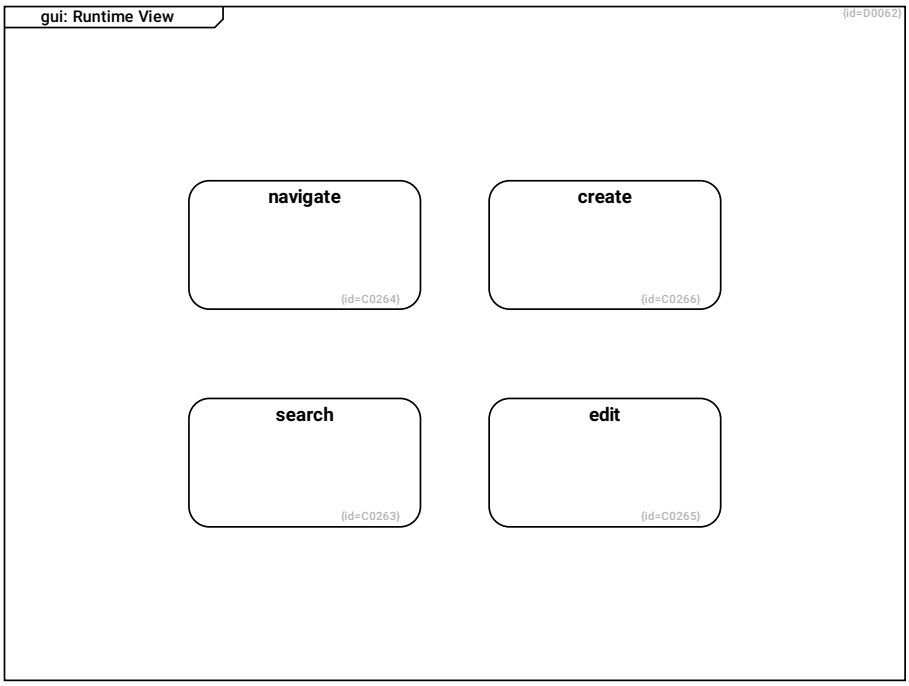
--> gui_sketch _result_list R0356

sketch_area C0257

--> gui_sketch _result_list R0357

gui_sketch _result_list C0262

1.5.5.2 gui: Runtime View



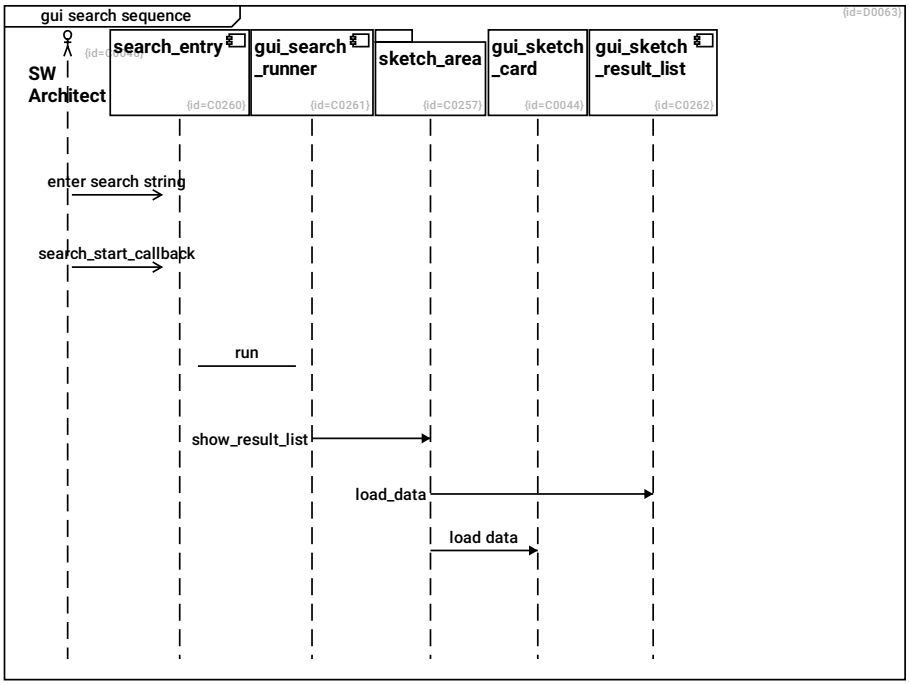
navigate C0264

edit C0265

create C0266

search C0263

1.5.5.2.1 gui search sequence



SW Architect C0046

enter search string --> search_entry R0362

search_start_callback --> search_entry R0361

search_entry C0260

run --> gui_search _runner R0363

gui_search _runner C0261

show_result_list --> sketch_area R0364

sketch_area C0257

load_data --> gui_sketch _result_list R0365

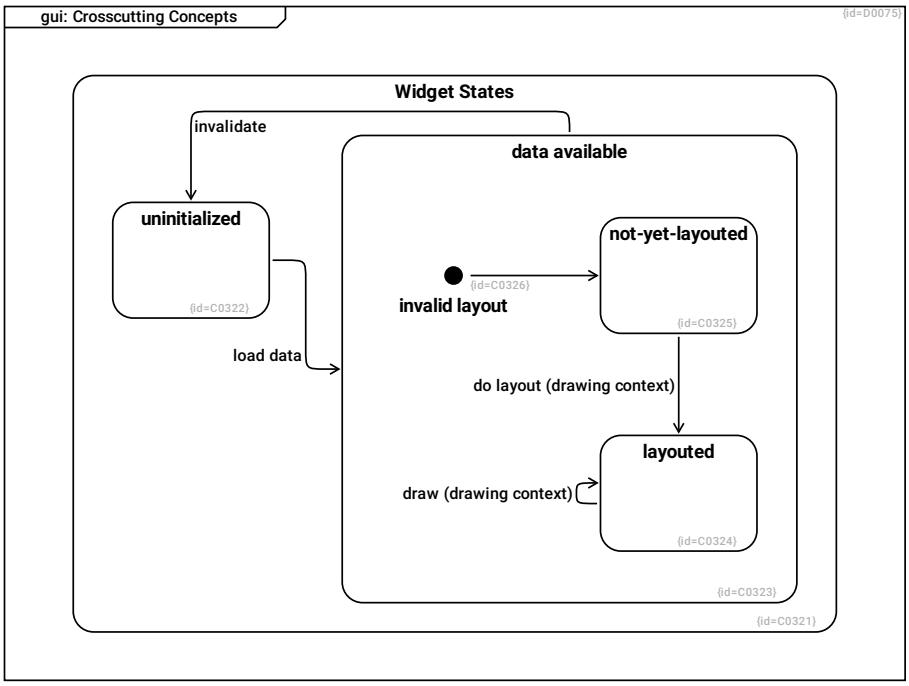
load data --> gui_sketch _card R0367

gui_sketch _card C0044

- maintains the data of one diagram
- displays one diagram

gui_sketch _result_list C0262

1.5.5.3 gui: Crosscutting Concepts



data available C0323

--> invalid layout R0466

--> layouted R0464

--> not-yet-layouted R0465

invalidate --> uninitialized R0469

not-yet-layouted C0325

do layout (drawing context) --> layouted R0468

invalid layout C0326

--> not-yet-layouted R0467

uninitialized C0322

load data --> data available R0460

Widget States C0321

--> data available R0458

--> not-yet-layouted R0463

--> uninitialized R0457

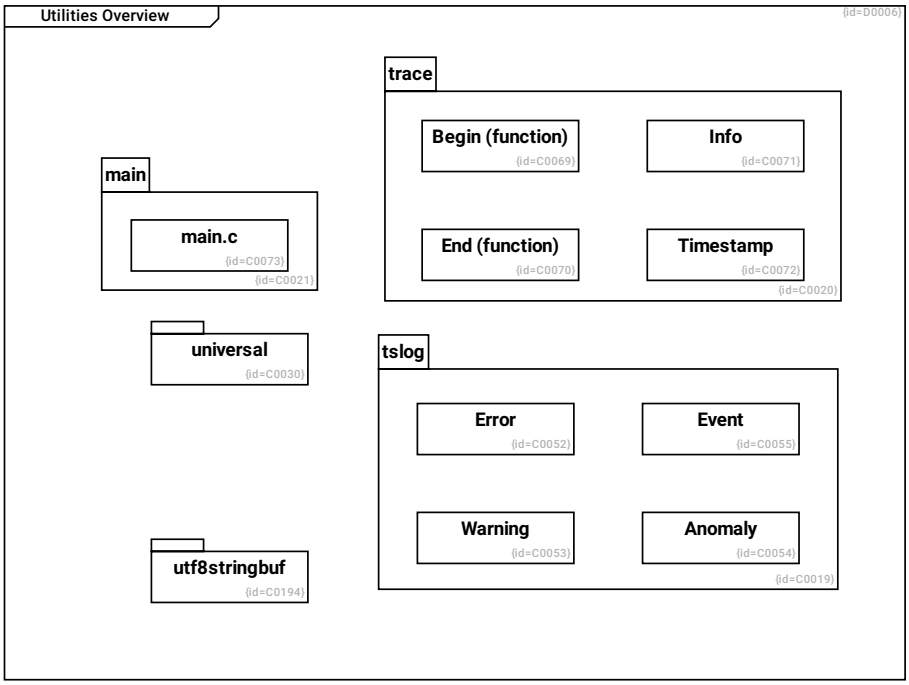
--> layouted R0459

layouted C0324

draw (drawing context) --> layouted R0462

1.5.6 Utilities Overview

This diagram shows small utility modules.



tslog C0019

- provides macros for event-logging and error-logging

- macros are typesafe (ts) so that e.g. integers are not acidentially passed to char* parameters

Logging functionality may be enabled even in RELEASE/NDEBUG code. Logs must not contain confidential data: These may be forwarded to syslog or the filesystem where access rights differ from the OS-process in which crystal_facet_uml runs.

--> Error R0064

--> Anomaly R0066

--> Event R0067

--> Warning R0065

trace C0020

- provides trace macros
- traces allow to reconstruct/follow the program flow

The macros shall be typesafe; The compiler shall complain if using wrong-typed parameters.

The trace functions are intended for debugging purposes only. They are disabled in RELEASE/NDEBUG code. Therefore it is valid to trace confidential information.

--> Info R0088

--> Timestamp R0089

--> End (function) R0087

--> Begin (function) R0086

main C0021

- starts the software
- evaluates command line parameters

--> main.c R0090

universal C0030

provides small, generic utility classes that are independant of the project

Error C0052

An error is a condition that leads to an observalbe malfunction.

Warning C0053

A warning is issued when a condition may possibly lead to a malfunction.

Anomaly C0054

An anomaly is a condition that is expected to not cause a malfunction but that should be logged to easier analyze issues.

Event C0055

An event is a signal that is send to or received from external software parts.

Begin (function) C0069

End (function) C0070

Info C0071

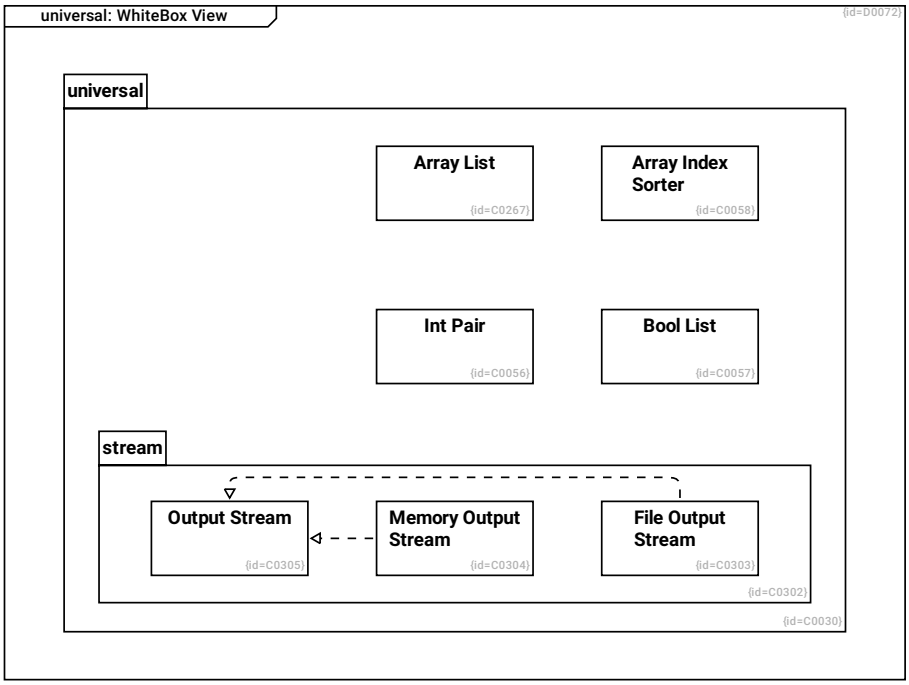
Timestamp C0072

main.c C0073

utf8stringbuf C0194

String library

1.5.6.1 universal: WhiteBox View



universal C0030

provides small, generic utility classes that are independant of the project

--> Array List R0366

--> Bool List R0069

--> Array Index Sorter R0070

--> Int Pair R0068

--> stream R0405

Int Pair C0056

Bool List C0057

Array Index Sorter C0058

Array List C0267

Memory Output Stream C0304

--> Output Stream R0409

File Output Stream C0303

--> Output Stream R0410

stream C0302

--> Output Stream R0408

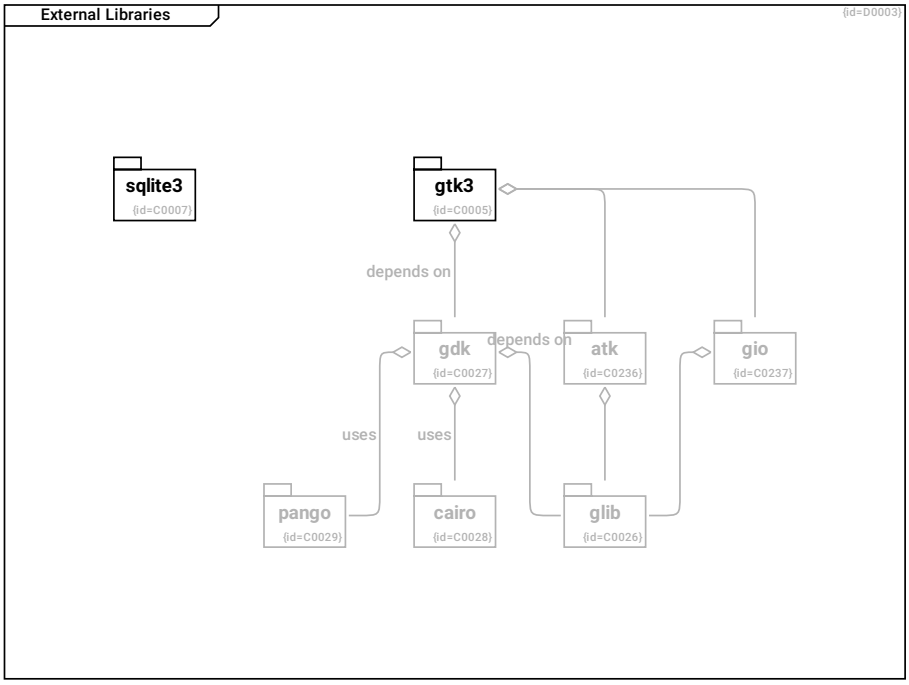
--> Memory Output Stream R0407

--> File Output Stream R0406

Output Stream C0305

1.5.7 External Libraries

This diagram shows the base libraries. crystal_facet_uml links to these dynamically.



gtk3 C0005

Toolkit which provides
- Windows and Widgets
- User Input handling

depends on --> gdk R0001

--> atk R0316

--> gio R0318

glib C0026
Platform abstraction layer

gdk C0027
Drawing library

uses --> cairo R0002

uses --> pango R0003

depends on --> glib R0004

cairo C0028
Library providing drawing primitives

pango C0029
Font rendering engine

sqlite3 C0007
SQL database

gio C0237
file system abstraction, distributed as part of glib

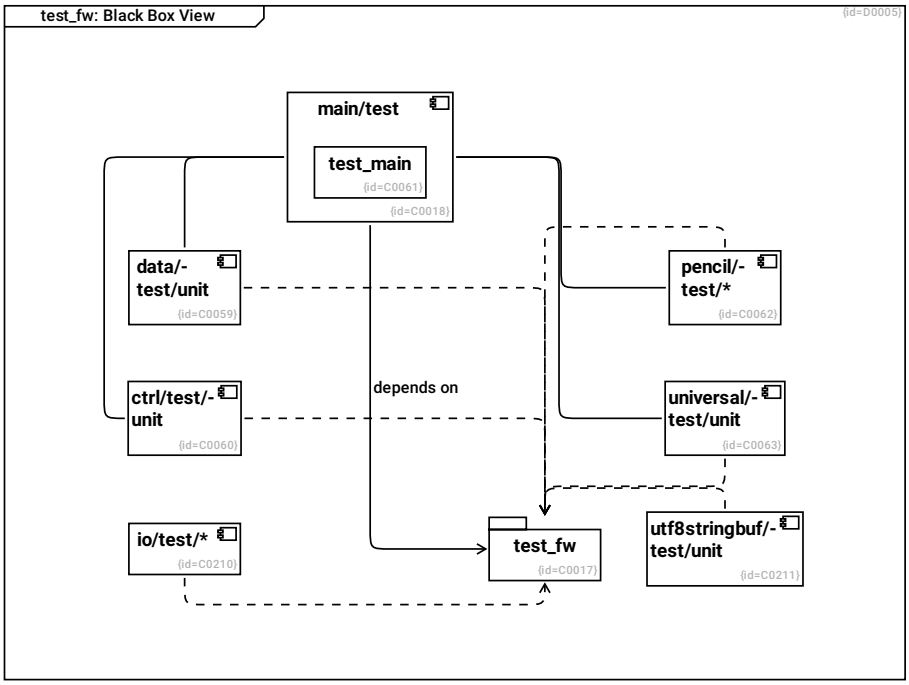
--> glib R0319

atk C0236
Accessibility Toolkit, kind of IPC to make widgets interfaces accessible.

--> glib R0317

1.5.8 test_fw: Black Box View

This diagram shows the sw-components involved in unit-testing.



test_fw C0017

Test framework

- runs test cases
- reports test results

main/test C0018

Each SW-moldule may have a subfolder named unittest which contains a set of tests; these

- test the SW-module

--> test_main R0071

depends on --> test_fw R0005

--> data/test/unit R0076

--> pencil/test/* R0077

--> universal/test/unit R0078

--> ctrl/test/unit R0079

data/test/unit C0059

--> test_fw R0075

test_main C0061

pencil/test/* C0062

--> test_fw R0074

universal/test/unit C0063

--> test_fw R0073

ctrl/test/unit C0060

--> test_fw R0072

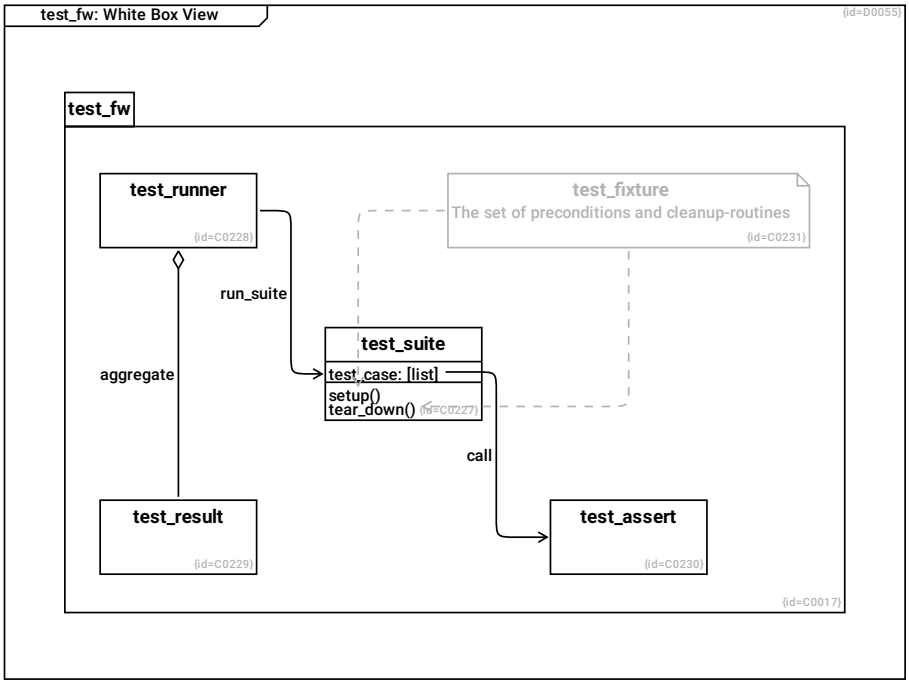
io/test/* C0210

--> test_fw R0291

utf8stringbuf/test/unit C0211

--> test_fw R0292

1.5.8.1 test_fw: White Box View



test_fw C0017

Test framework

- runs test cases
- reports test results

--> test_suite R0301

--> test_assert R0304

--> test_fixture R0305

--> test_runner R0302

--> test_result R0303

test_suite C0227

test_case: [list] F0054

setup() F0055

tear_down() F0056

call --> test_assert R0309

test_assert C0230

test_fixture C0231

The set of preconditions and cleanup-routines

--> test_suite R0306

--> test_suite R0307

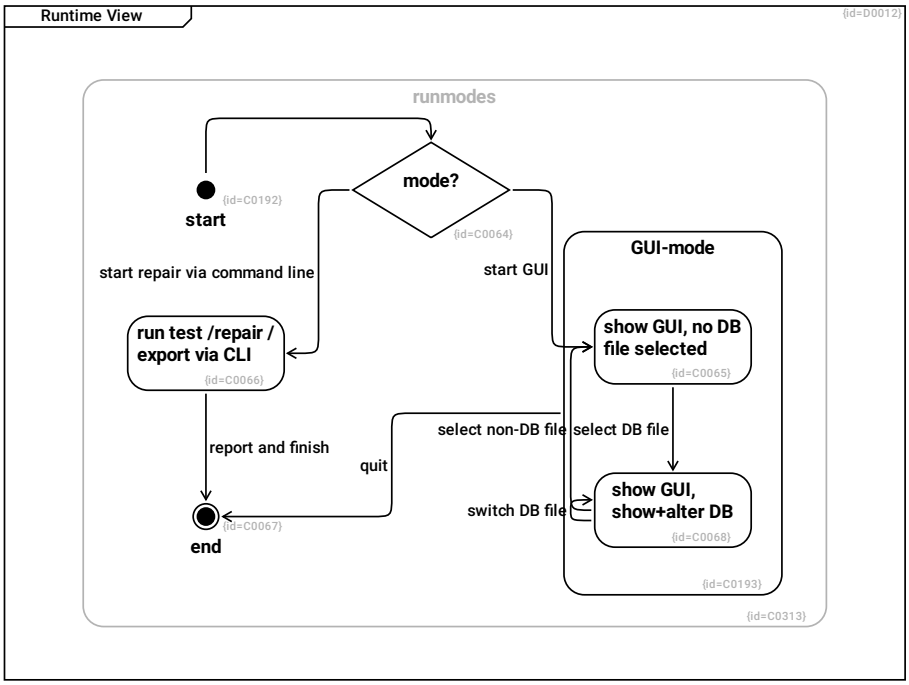
test_runner C0228

aggregate --> test_result R0308

run_suite --> test_suite R0311

test_result C0229

1.6 Runtime View



run test /repair /export via CLI C0066

report and finish --> end R0085

end C0067

start C0192

--> mode? R0251

GUI-mode C0193

--> show GUI, no DB file selected R0252

--> show GUI, show+alter DB R0253

quit --> end R0254

runmodes C0313

--> mode? R0446

--> start R0447

--> run test /repair /export via CLI R0448

--> end R0449

--> GUI-mode R0450

mode? C0064

start repair via command line --> run test /repair /export via CLI R0080

start GUI --> show GUI, no DB file selected R0081

show GUI, no DB file selected C0065

select DB file --> show GUI, show+alter DB R0082

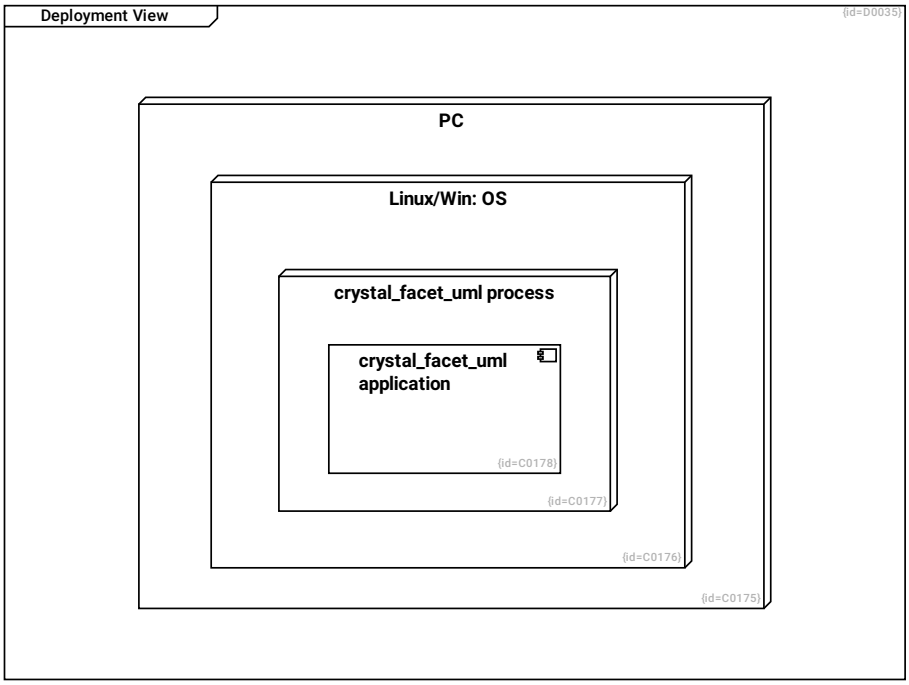
show GUI, show+alter DB C0068

switch DB file --> show GUI, show+alter DB R0083

select non-DB file --> show GUI, no DB file selected R0102

Try to open a non-sqlite 3 DB file.

1.7 Deployment View



crystal_facet_uml application C0178

Linux/Win: OS C0176

--> crystal_facet_uml process R0238

crystal_facet_uml process C0177

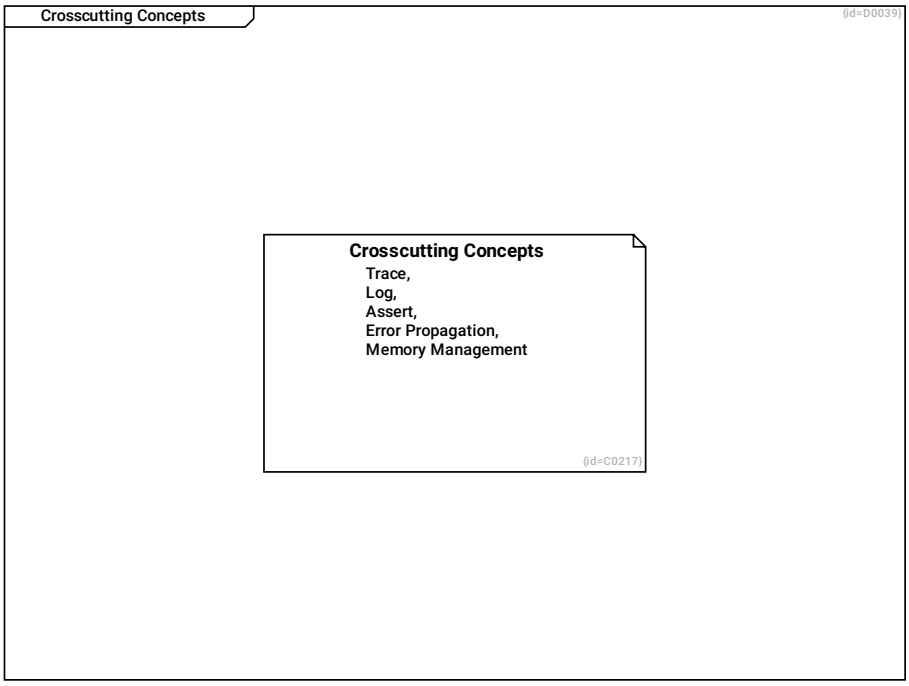
--> crystal_facet_uml application R0243

PC C0175

--> Linux/Win: OS R0237

1.8 Crosscutting Concepts

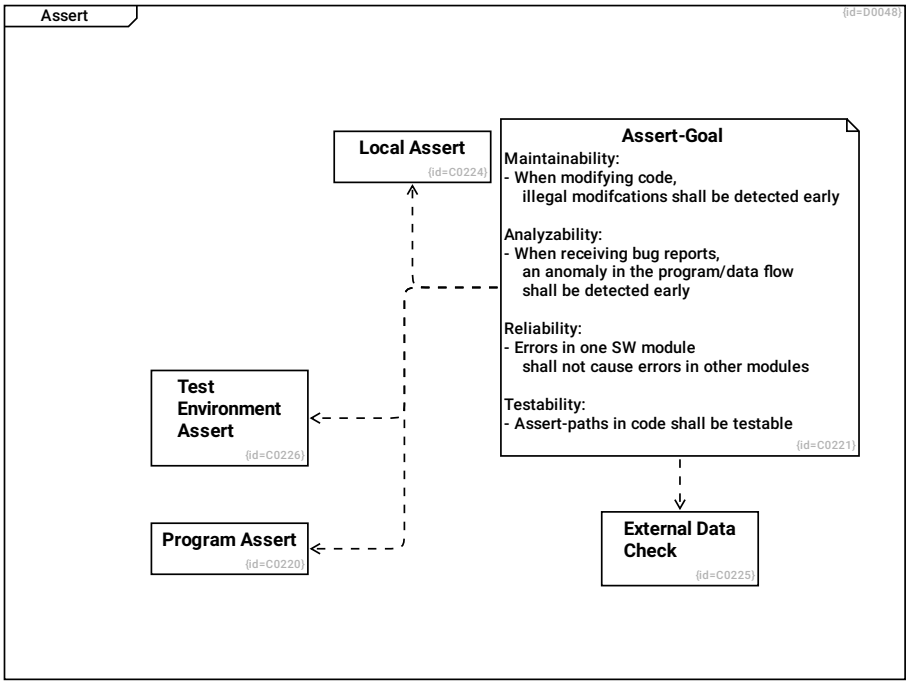
This chapter shows concepts that shall be applied through all parts of the software.



Crosscutting Concepts C0217

Trace, Log, Assert, Error Propagation, Memory Management

1.8.1 Assert



Local Assert C0224

Use "assert(COND);" statements to ensure code+data consistency within one software module, These are executed in DEBUG mode only.

External Data Check C0225

Use "if(COND) {...} else {LOG("error");}" statements to ensure code+data consistency towards external input, External input may possibly not follow any specification.

Assert-Goal C0221

Maintainability:

- When modifying code, illegal modifcations shall be detected early

Analyzability:

- When receiving bug reports, an anomaly in the program/data flow shall be detected early

Reliability:

- Errors in one SW module shall not cause errors in other modules

Testability:

- Assert-paths in code shall be testable

--> Program Assert R0295

--> External Data Check R0298

--> Local Assert R0299

--> Test Environment Assert R0300

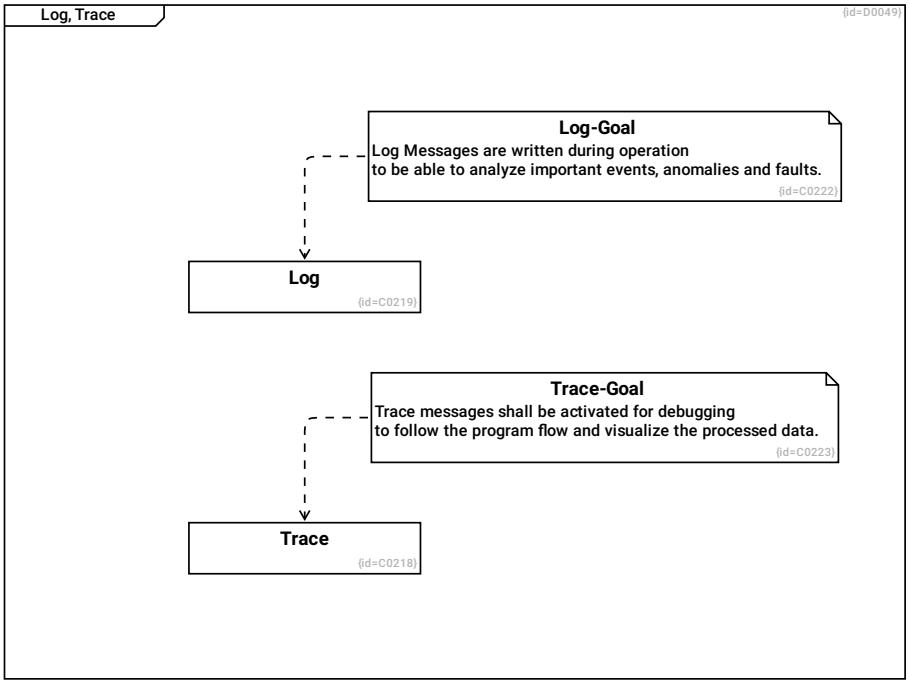
Test Environment Assert C0226

Use "TEST_ENVIRONMENT_ASSERT()" statement to ensure a valid test environment. Errors always terminate the test run.

Program Assert C0220

Use "if(COND) {...} else {LOG("error");assert(false);}" statements to ensure code+data consistency towards other software modules, This makes code more robust if unexpected behavior occurs during operation.

1.8.2 Log, Trace



Log-Goal C0222

Log Messages are written during operation to be able to analyze important events, anomalies and faults.

--> Log R0296

Log C0219

Security:

- Log messages must not contain confidential data

Trace-Goal C0223

Trace messages shall be activated for debugging to follow the program flow and visualize the processed data.

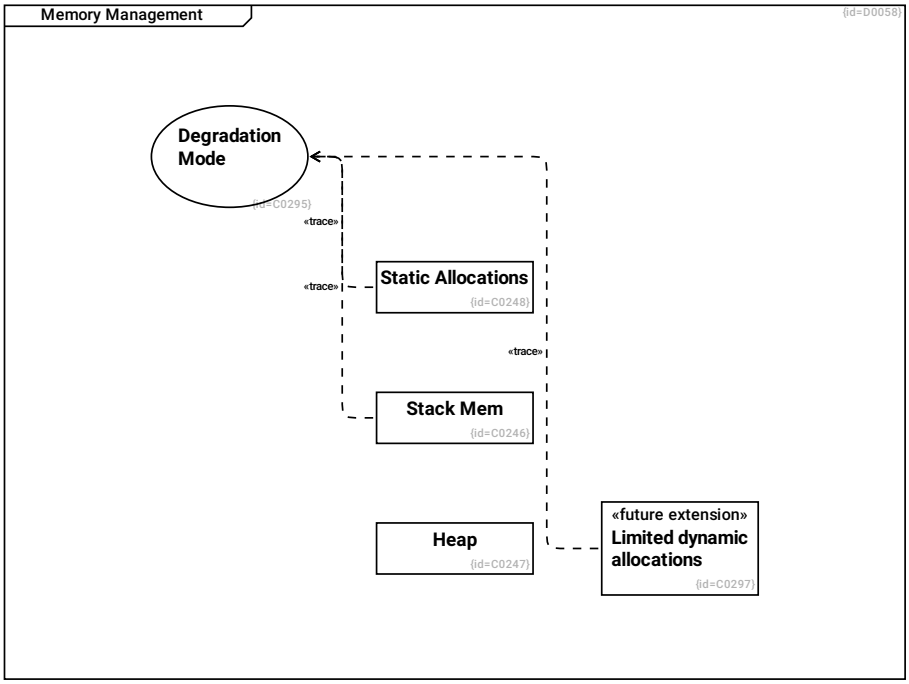
--> Trace R0297

Trace C0218

Performance:

- Deactivated trace messages shall not slow down the program by more than 1%.
- Activated trace messages shall not slow down the program by more than 10%.
- Operation performance (NDEBUG-mode) shall not be affected by trace messages.

1.8.3 Memory Management



Static Allocations C0248

All data structures of crystal_facet_uml shall be statically allocated.

--> Degradation Mode R0470

Heap C0247

Only the external libraries may use the heap.

Limited dynamic allocations C0297

Memory may be dynamically allocated from heap for non-base use cases. The program shall continue operation if no memorry is available anymore. See Section 1.10.1.1: Performance Efficiency (Time+Resource).

--> Degradation Mode R0472

Stack Mem C0246

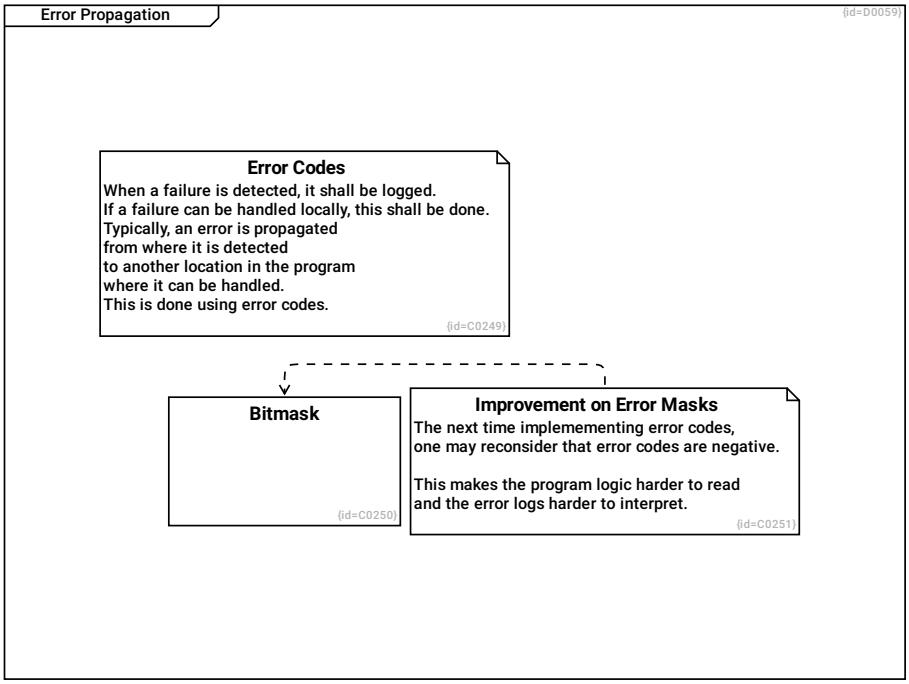
On stack, only small data structures (<10kB) shall be stored.

--> Degradation Mode R0471

Degradation Mode C0295

In case of low memory, the base functionality of crystal_facet_uml shall still work reliably.

1.8.4 Error Propagation



Improvement on Error Masks C0251

The next time implemementing error codes, one may reconsider that error codes are negative.

This makes the program logic harder to read and the error logs harder to interpret.

--> Bitmask R0341

Bitmask C0250

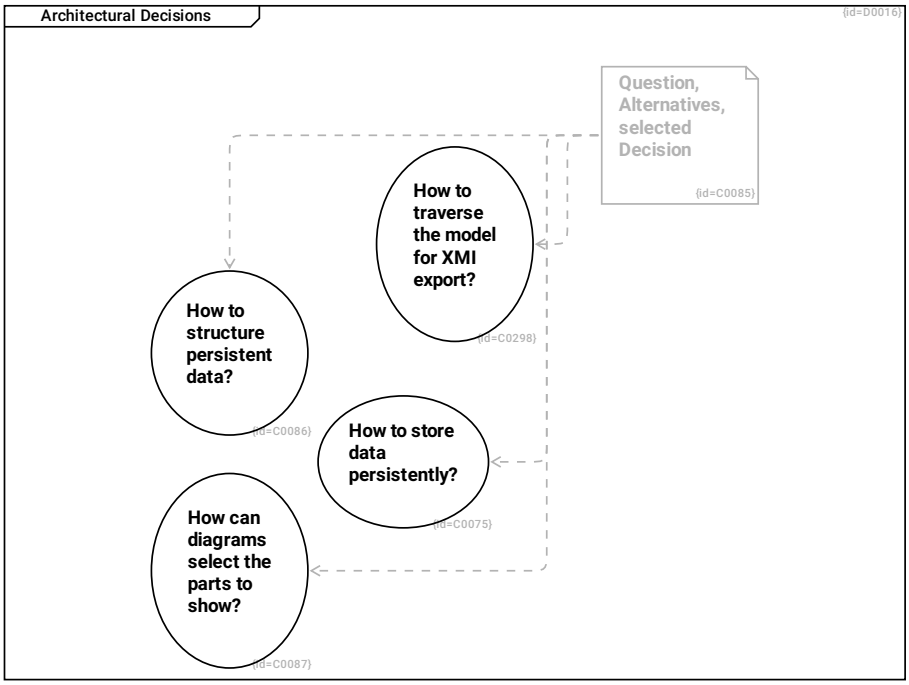
The error bitmask shall be designed in a way that a zero represents no error, that the bits define the error cause/type, that two errors can be merged into one error code using a bitwise or, that error codes are negative.

Error Codes C0249

When a failure is detected, it shall be logged. If a failure can be handled locally, this shall be done. Typically, an error is propagated from where it is detected to another location in the program where it can be handled. This is done using error codes.

1.9 Architectural Decisions

This diagram shows which major design decisions were taken.



How to structure persistent data? C0086

The database structure shall be simple but also close to UML in oder to store an UML-Model

How can diagrams select the parts to show? C0087

Each diagram shall show selected parts of the UML-Model, especially the scenario-based sequence, communication and timing diagrams.

How to traverse the model for XMI export? C0298

Question, Alternatives, selected Decision C0085

--> How to store data persistently? R0103

--> How to structure persistent data? R0104

--> How can diagrams select the parts to show? R0105

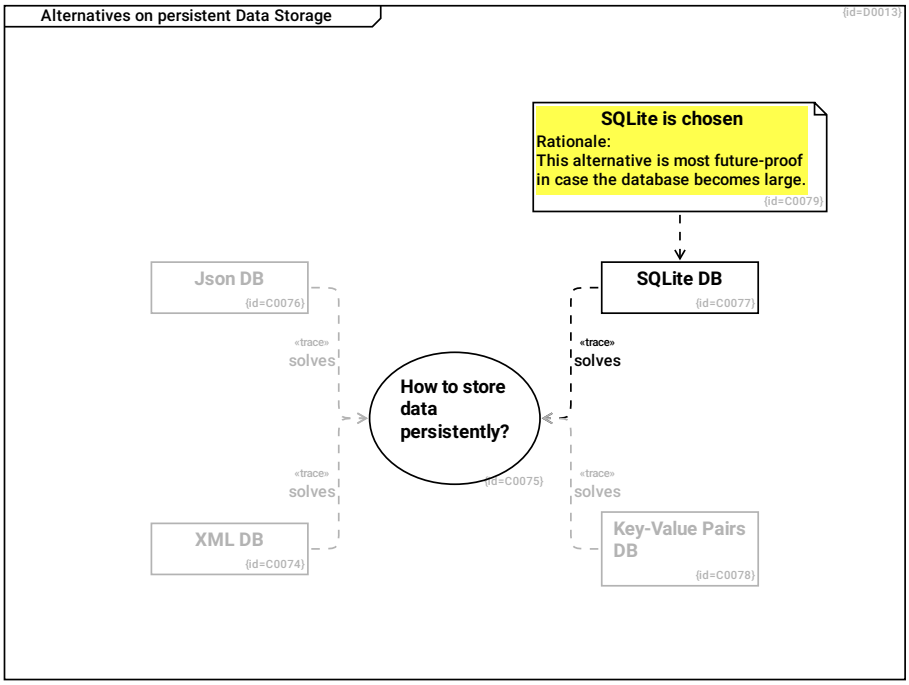
--> How to traverse the model for XMI export? R0401

How to store data persistently? C0075

The UML Model shall be stored persistently.

1.9.1 Alternatives on persistent Data Storage

This diagram shows fundamental design decisions of the crystal_facet_uml architecture.



XML DB C0074

The database is stored in XML format. pro: text based, fits to svn and git repositories con: all data needs to fit to RAM (or complex data handling)

solves --> How to store data persistently? R0094

Json DB C0076

The database is stored in Json format. pro: text based, fits to svn and git repositories con: all data needs to fit to RAM (or complex data handling)

solves --> How to store data persistently? R0093

SQLite DB C0077

The database is stored in sqlite3 format. pro: the database cares on handling big amounts of data the database cares on reliability and consistency of data con: binary files are not suitable for git repositories

solves --> How to store data persistently? R0091

Key-Value Pairs DB C0078

The database is stored as key-value pairs. pro: allows automatic merges by git and svn con: all data needs to fit to RAM (or complex data handling)

solves --> How to store data persistently? R0092

SQLite is chosen C0079

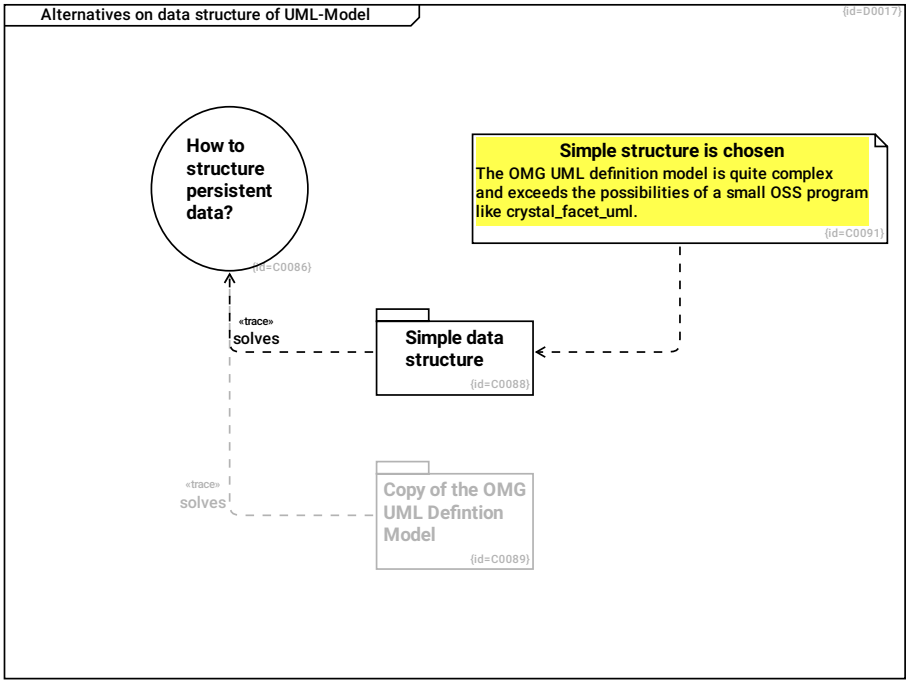
Rationale: This alternative is most future-proof in case the database becomes large.

--> SQLite DB R0095

How to store data persistently? C0075

The UML Model shall be stored persistently.

1.9.2 Alternatives on data structure of UML-Model



How to structure persistent data? C0086

The database structure shall be simple but also close to UML in oder to store an UML-Model

Simple data structure C0088

The data is structured with focus on simplicity. pro: Simple data structure. There are similarities to UML (M2 Meta-Model) and MOF (M3 Meta-Model). con: takes into account that some things of UML cannot be represented

solves --> How to structure persistent data? R0110

Copy of the OMG UML Defintion Model C0089

The data is structured according to the UML meta model defined by OMG. pro: allows to model all UML elements easy conversion to XMI and back con: the UML meta model is complex. Not suitable for a small diagram-drawing application like crystal_facet_uml.

solves --> How to structure persistent data? R0109

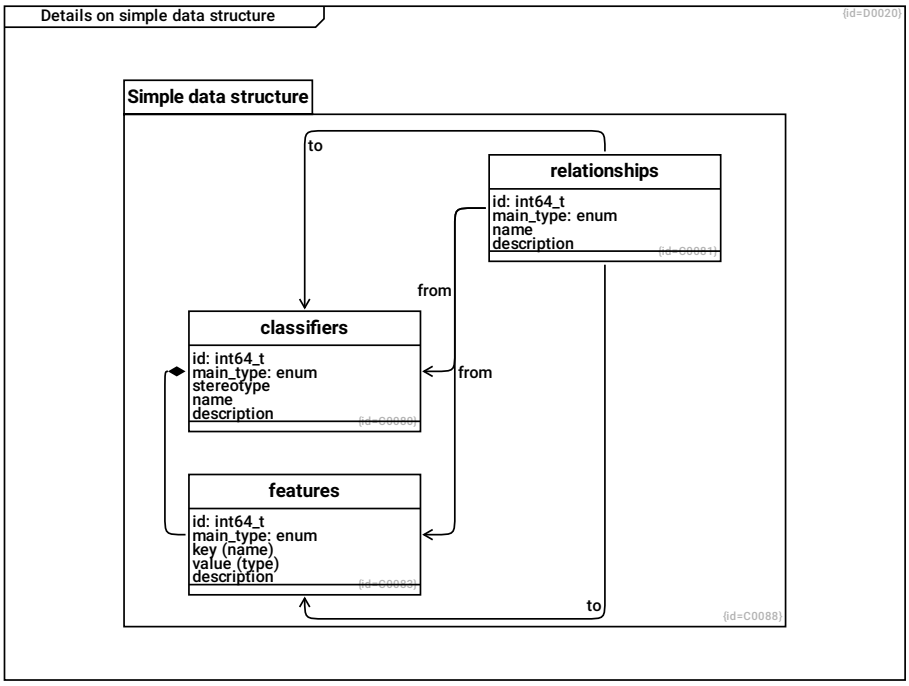
Simple structure is chosen C0091

The OMG UML definition model is quite complex and exceeds the possibilities of a small OSS program like crystal_facet_uml.

--> Simple data structure R0117

1.9.2.1 Details on simple data structure

This diagram shows the data structure of the "Simple data structure" alternative.



classifiers C0080

id: int64_t F0012

main_type: enum F0017

stereotype F0018

name F0019

description F0020

--> features R0210

features C0083

id: int64_t F0013

main_type: enum F0024

key (name) F0025

value (type) F0026

description F0027

Simple data structure C0088

> The data is structured with focus on simplicity. pro: Simple data structure. There are similarities to UML (M2 Meta-Model) and MOF (M3 Meta-Model). con: takes into account that some things of UML cannot be represented

--> relationships R0106

--> classifiers R0107

--> features R0108

relationships C0081

id: int64_t F0014

main_type: enum F0021

name F0022

description F0023

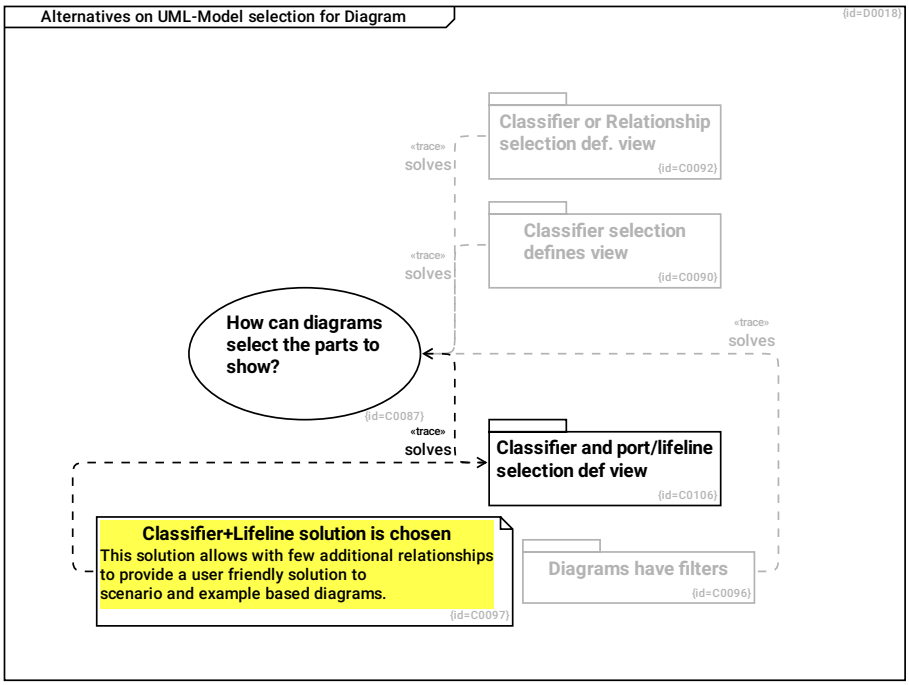
from --> classifiers R0096

to --> classifiers R0097

from --> features R0205

to --> features R0206

1.9.3 Alternatives on UML-Model selection for Diagram



How can diagrams select the parts to show? C0087

Each diagram shall show selected parts of the UML-Model, especially the scenario-based sequence, communication and timing diagrams.

Classifier or Relationship selection def. view C0092

Depending on the diagram type, either classifiers determine what is visble or relationships determine what is visible. pro: similar concept for diagrams that show options/alternatives/scenarios to diagrams that show invariants. con: Bigger extension to existing structure in version 1.0.0 Algorithm to distinguish used from unused elements gets complicated

solves --> How can diagrams select the parts to show? R0118

Diagrams have filters C0096

Either diagrams filter ranges of x-order/y-order/list-order or diagrams filter by use-case (which requires that all relations belong to a use-case) pro: small changes to existing database structure only con: Unclear gui concept: How to make this filtering transparent to the user? How to copy/paste between diagrams? How to make invisible relationships visible?

solves --> How can diagrams select the parts to show? R0127

Classifier+Lifeline solution is chosen C0097

This solution allows with few additional relationships to provide a user friendly solution to scenario and example based diagrams.

--> Classifier and port/lifeline selection def view R0141

Classifier and port/lifeline selection def view C0106

solves --> How can diagrams select the parts to show? R0140

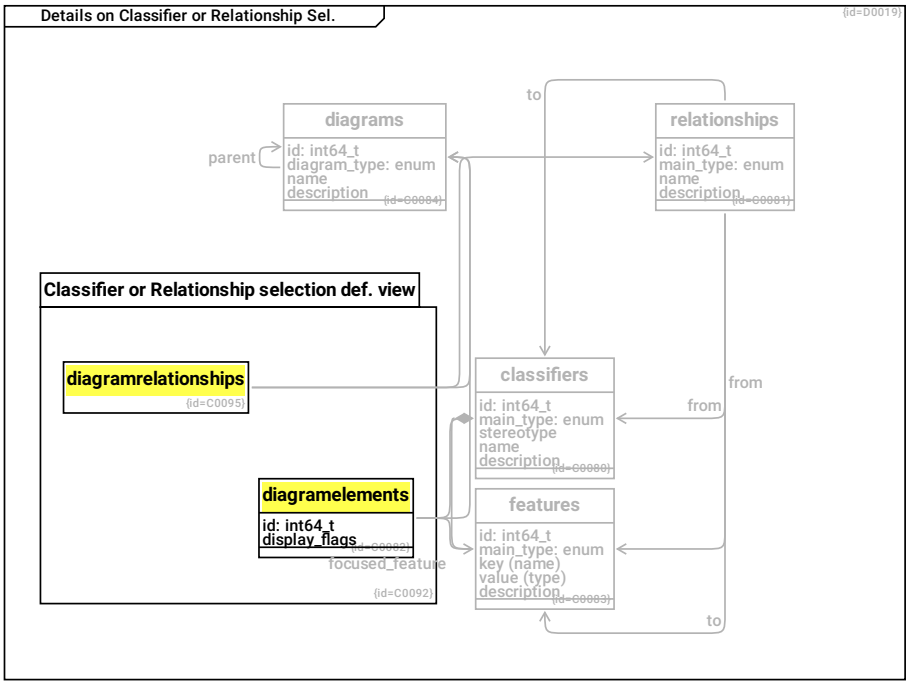
Classifier selection defines view C0090

Diagramelements define which classifiers are visible in which diagram. All features of a visible classifier are shown. All relationships are shown that have visible classifiers at both ends. pro: simple selection rule con: Problematic for diagrams that show possible scenarios and alternatives (e.g sequences). This solution assumes that the model is invariant for all diagrams, therefore all diagrams showing examples/scenarios have to show own instances of classes, not generic classes that are used in other diagrams.

solves --> How can diagrams select the parts to show? R0116

1.9.3.1 Details on Classifier or Relationship Sel.

Depending on the diagram type, selection of contents is done on classifiers or relationships.



classifiers C0080

id: int64_t F0012

main_type: enum F0017

stereotype F0018

name F0019

description F0020

--> features R0210

features C0083

id: int64_t F0013

main_type: enum F0024

key (name) F0025

value (type) F0026

description F0027

diagrams C0084

id: int64_t F0015

diagram_type: enum F0028

name F0029

description F0030

parent --> diagrams R0099

diagramrelationships C0095

--> relationships R0126

--> diagrams R0130

Classifier or Relationship selection def. view C0092

Depending on the diagram type, either classifiers determine what is visble or relationships determine what is visible. pro: similar concept for diagrams that show options/alternatives/scenarios to diagrams that show invariants. con: Bigger extension to existing structure in version 1.0.0 Algorithm to distinguish used from unused elements gets complicated

--> diagramrelationships R0121

--> diagramelements R0131

relationships C0081

id: int64_t F0014

main_type: enum F0021

name F0022

description F0023

from --> classifiers R0096

to --> classifiers R0097

from --> features R0205

to --> features R0206

diagramelements C0082

id: int64_t F0016

display_flags F0031

--> diagrams R0100

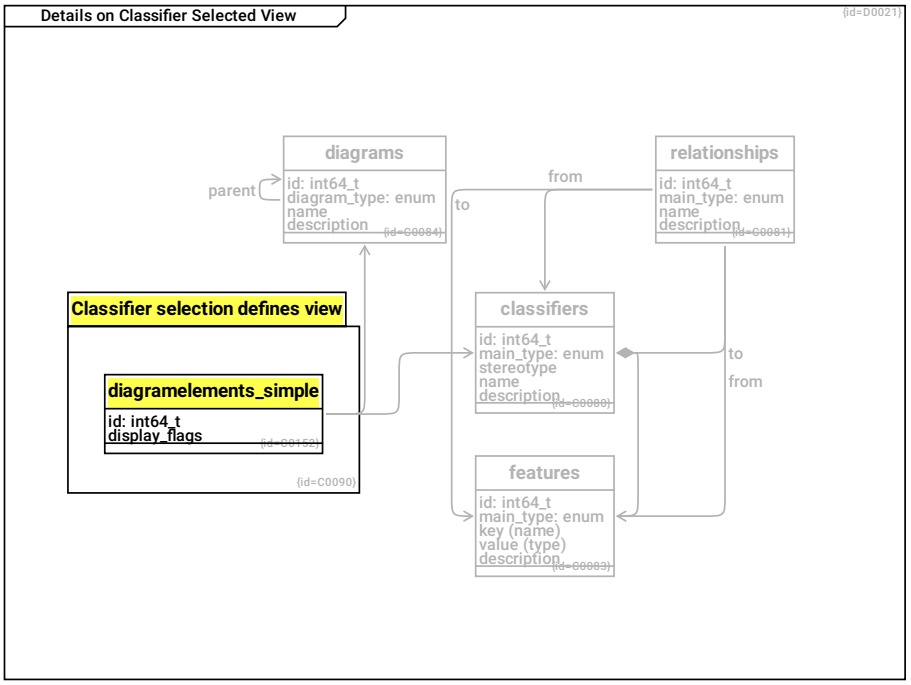
--> classifiers R0101

focused_feature --> features R0204

to select one of possibly several lifelines

1.9.3.2 Details on Classifier Selected View

The simple solution is to not implement special mechanisms for scenario-based diagrams but to require that all shown classifiers in a scenario based diagram are explicit instances of the model-class.



diagrams C0084

id: int64_t F0015

diagram_type: enum F0028

name F0029

description F0030

parent --> diagrams R0099

classifiers C0080

id: int64_t F0012

main_type: enum F0017

stereotype F0018

name F0019

description F0020

--> features R0210

features C0083

id: int64_t F0013

main_type: enum F0024

key (name) F0025

value (type) F0026

description F0027

diagramelements_simple C0152

id: int64_t F0034

display_flags F0035

--> diagrams R0207

--> classifiers R0208

Classifier selection defines view C0090

Diagramelements define which classifiers are visible in which diagram. All features of a visible classifier are shown. All relationships are shown that have visible classifiers at both ends. pro: simple selection rule con: Problematic for diagrams that show possible scenarios and alternatives (e.g sequences). This solution assumes that the model is invariant for all diagrams, therefore all diagrams showing examples/scenarios have to show own instances of classes, not generic classes that are used in other diagrams.

--> diagramelements_simple R0209

relationships C0081

id: int64_t F0014

main_type: enum F0021

name F0022

description F0023

from --> classifiers R0096

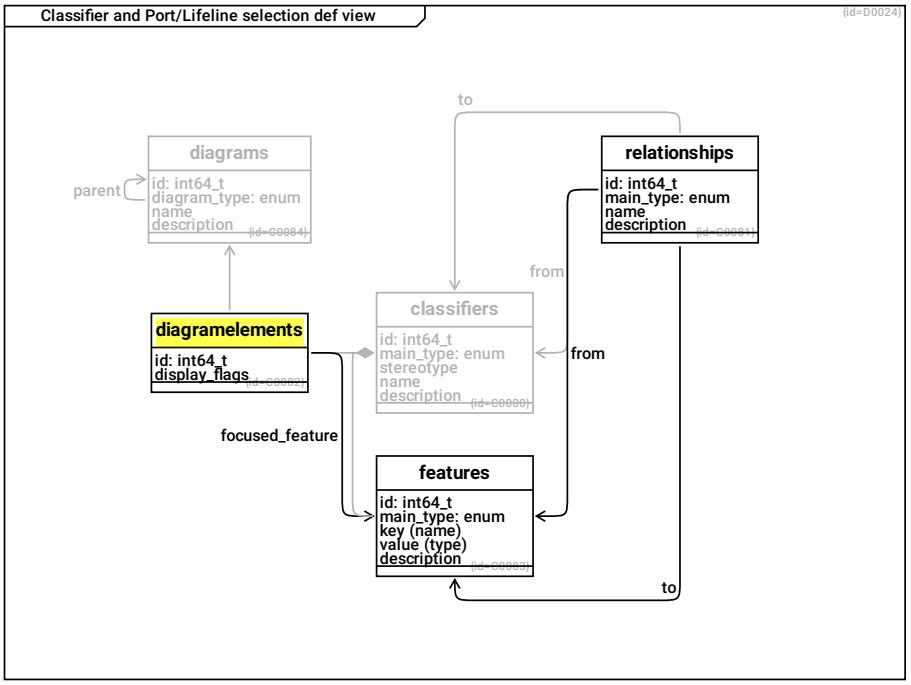
to --> classifiers R0097

from --> features R0205

to --> features R0206

1.9.3.3 Classifier and Port/Lifeline selection def view

This (finally chosen and implemented) solution selects timing and sequence lifelines by a focused_feature link. To enhance usability, this cannot be selected but the program creates an own lifeline for every scenario-based diagram.



features C0083

id: int64_t F0013

main_type: enum F0024

key (name) F0025

value (type) F0026

description F0027

classifiers C0080

id: int64_t F0012

main_type: enum F0017

stereotype F0018

name F0019

description F0020

--> features R0210

diagrams C0084

id: int64_t F0015

diagram_type: enum F0028

name F0029

description F0030

parent --> diagrams R0099

relationships C0081

id: int64_t F0014

main_type: enum F0021

name F0022

description F0023

from --> classifiers R0096

to --> classifiers R0097

from --> features R0205

to --> features R0206

diagramelements C0082

id: int64_t F0016

display_flags F0031

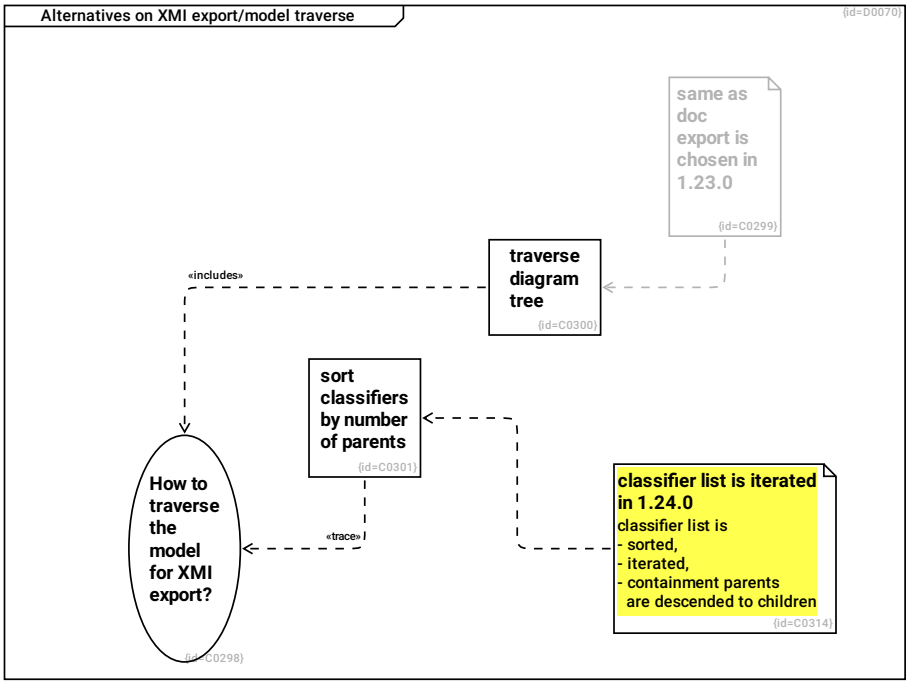
--> diagrams R0100

--> classifiers R0101

focused_feature --> features R0204

to select one of possibly several lifelines

1.9.4 Alternatives on XMI export/model traverse



same as doc export is chosen in 1.23.0 C0299

--> traverse diagram tree R0402

classifier list is iterated in 1.24.0 C0314

classifier list is

- sorted,

- iterated,

- containment parents are descended to children

--> sort classifiers by number of parents R0451

traverse diagram tree C0300

as for other document exports, simply traverse the diagram tree in depth first order. pro: simple to implement con: the hierarchical xmi structure cannot be produced

--> How to traverse the model for XMI export? R0403

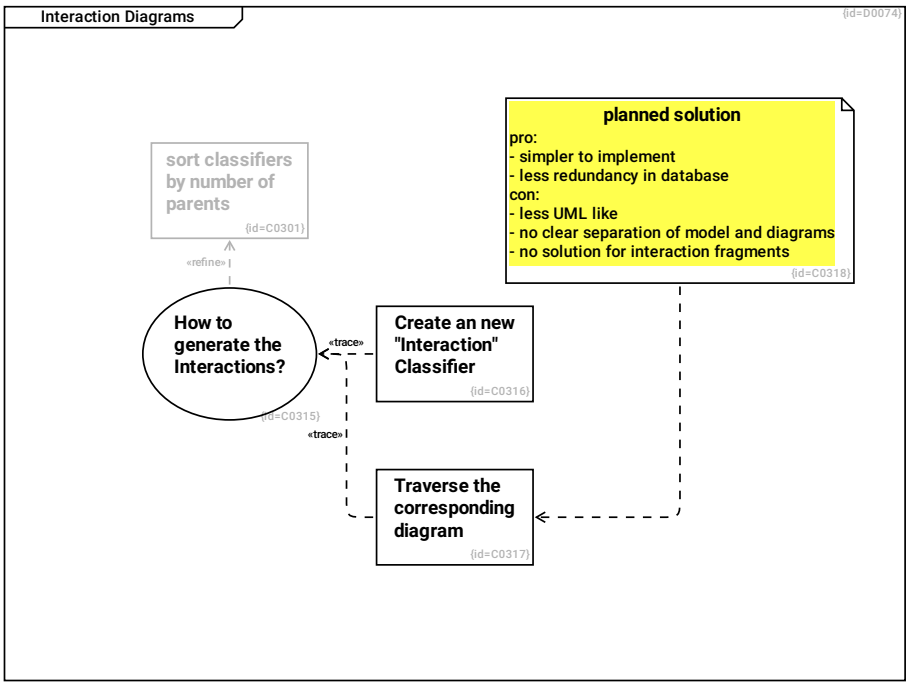
sort classifiers by number of parents C0301

traverse the list of classifiers, not the diagrams; export each classifier together with all children pro: allows to export the expected xmi structure con: difficult to implement, diagram descriptions are lost

--> How to traverse the model for XMI export? R0404

How to traverse the model for XMI export? C0298

1.9.4.1 Interaction Diagrams



sort classifiers by number of parents C0301

traverse the list of classifiers, not the diagrams; export each classifier together with all children pro: allows to export the expected xmi structure con: difficult to implement, diagram descriptions are lost

planned solution C0318

pro:

- simpler to implement
- less redundancy in database con:
- less UML like
- no clear separation of model and diagrams
- no solution for interaction fragments

--> Traverse the corresponding diagram R0455

Create an new "Interaction" Classifier C0316

--> How to generate the Interactions? R0453

Traverse the corresponding diagram C0317

Traverse the corresponding diagram when encountering lifelines

--> How to generate the Interactions? R0454

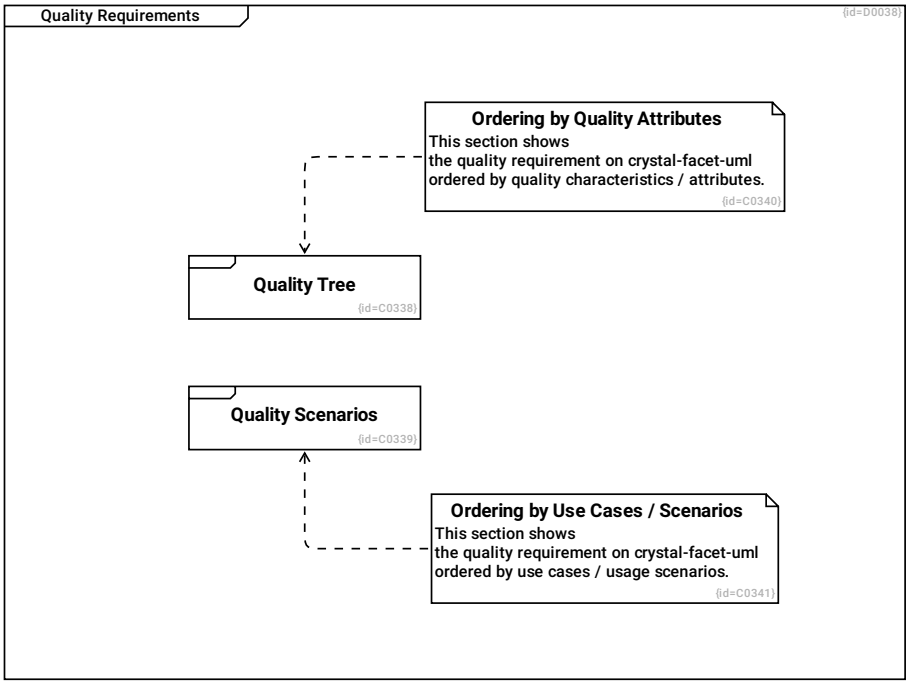
How to generate the Interactions? C0315

Interactions are

- Sequence,
- Timing,
- Communication and
- Interaction Overview Diagrams

--> sort classifiers by number of parents R0452

1.10 Quality Requirements



Ordering by Quality Attributes C0340

This section shows the quality requirement on crystal-facet-uml ordered by quality characteristics / attributes.

--> Quality Tree R0485

Quality Tree C0338

Ordering by Use Cases / Scenarios C0341

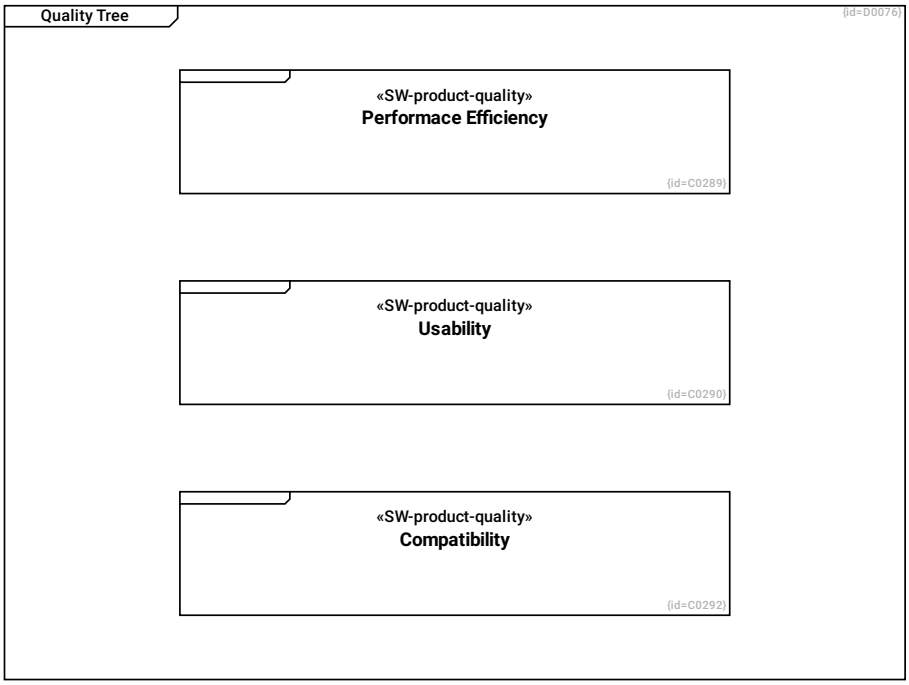
This section shows the quality requirement on crystal-facet-uml ordered by use cases / usage scenarios.

--> Quality Scenarios R0486

Quality Scenarios C0339

1.10.1 Quality Tree

This section shows quality requirements (structured into a quality tree)



Performace Efficiency C0289

see Section 1.10.1.1: Performance Efficiency (Time+Resource).

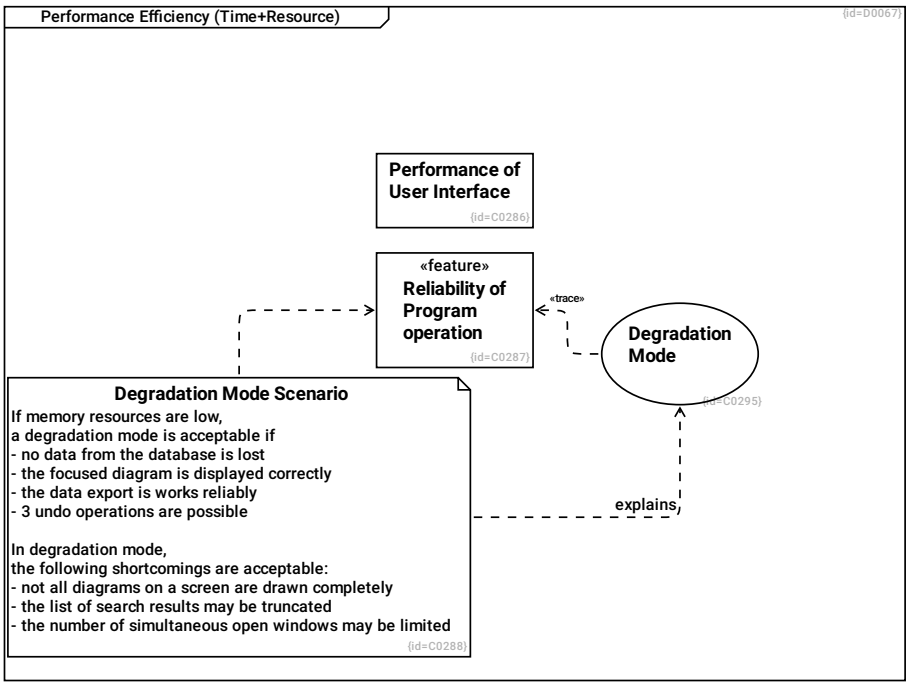
Usability C0290

see Section 1.10.1.2: Usability (Attactiveness,...).

Compatibility C0292

see Section 1.10.1.3: Compatibility (Replace,Interop.,...).

1.10.1.1 Performance Efficiency (Time+Resource)



Performance of User Interface C0286

Degradation Mode C0295

In case of low memory, the base functionality of crystal_facet_uml shall still work reliably.

--> Reliability of Program operation R0397

Degradation Mode Scenario C0288

If memory resources are low, a degradation mode is acceptable if

- no data from the database is lost
- the focused diagram is displayed correctly
- the data export is works reliably
- 3 undo operations are possible

In degradation mode, the following shortcomings are acceptable:

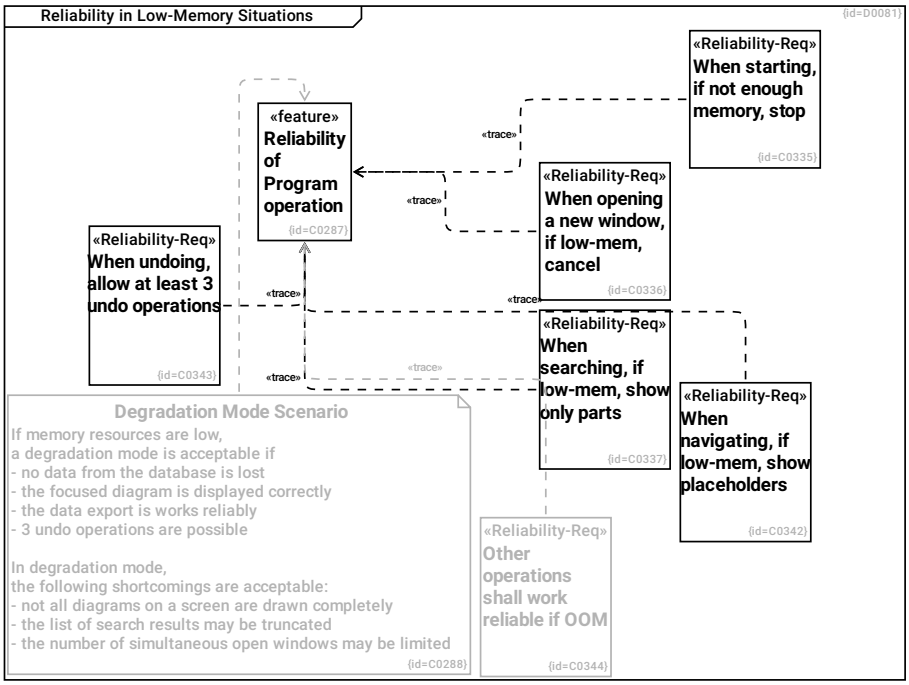
- not all diagrams on a screen are drawn completely
- the list of search results may be truncated
- the number of simultaneous open windows may be limited

--> Reliability of Program operation R0395

explains --> Degradation Mode R0396

Reliability of Program operation C0287

1.10.1.1.1 Reliability in Low-Memory Situations



When starting, if not enough memory, stop C0335

When starting, if not enough memory is available to operate, the program shall inform the user and stop.

--> Reliability of Program operation R0482

When opening a new window, if low-mem, cancel C0336

When opening a new main window, if low-mem, the program shall inform the user and cancel the operation.

--> Reliability of Program operation R0483

When searching, if low-mem, show only parts C0337

When searching, if not enough memory available for storing the search result list, the program shall indicate that some results are omitted and show only parts of the result list.

--> Reliability of Program operation R0484

When navigating, if low-mem, show placeholders C0342

When navigating, if low-mem, the program shall show placeholder rectangles where rendering of diagrams is not possible.

--> Reliability of Program operation R0487

Degradation Mode Scenario C0288

If memory resources are low, a degradation mode is acceptable if

- no data from the database is lost

- the focused diagram is displayed correctly

- the data export is works reliably
- 3 undo operations are possible

In degradation mode, the following shortcomings are acceptable:

- not all diagrams on a screen are drawn completely
- the list of search results may be truncated
- the number of simultaneous open windows may be limited

--> Reliability of Program operation R0395

When undoing, allow at least 3 undo operations C0343

When undoing/redoing past actions, the program shall perform at least 3 undo/redo operations before informing the user that undo/redo is not possible anymore.

--> Reliability of Program operation R0488

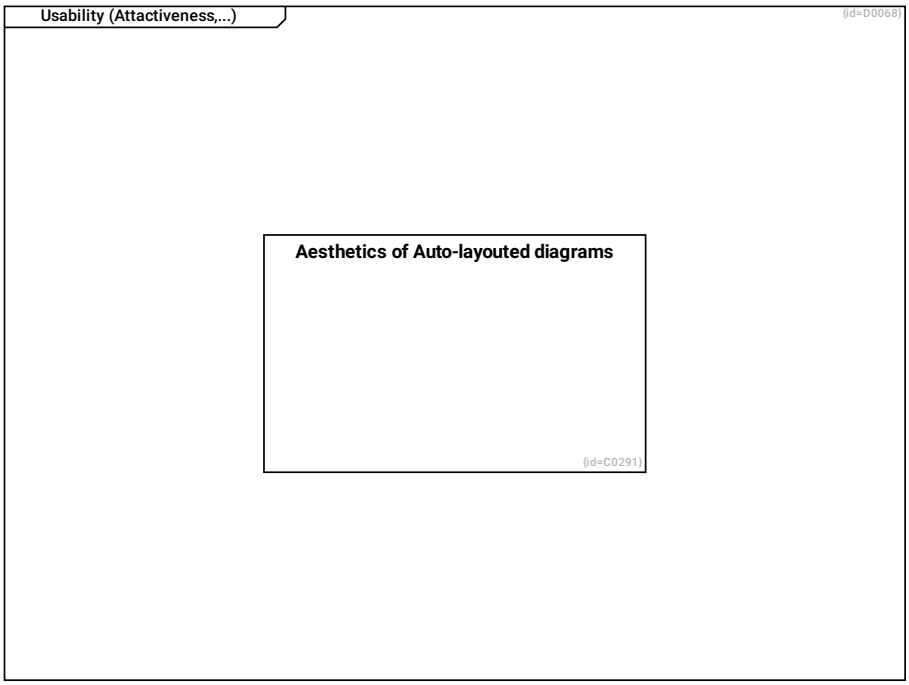
Reliability of Program operation C0287

Other operations shall work reliable if OOM C0344

When the user operates the program, if low-mem, the program shall work reliably for all use cases where no exception is stated.

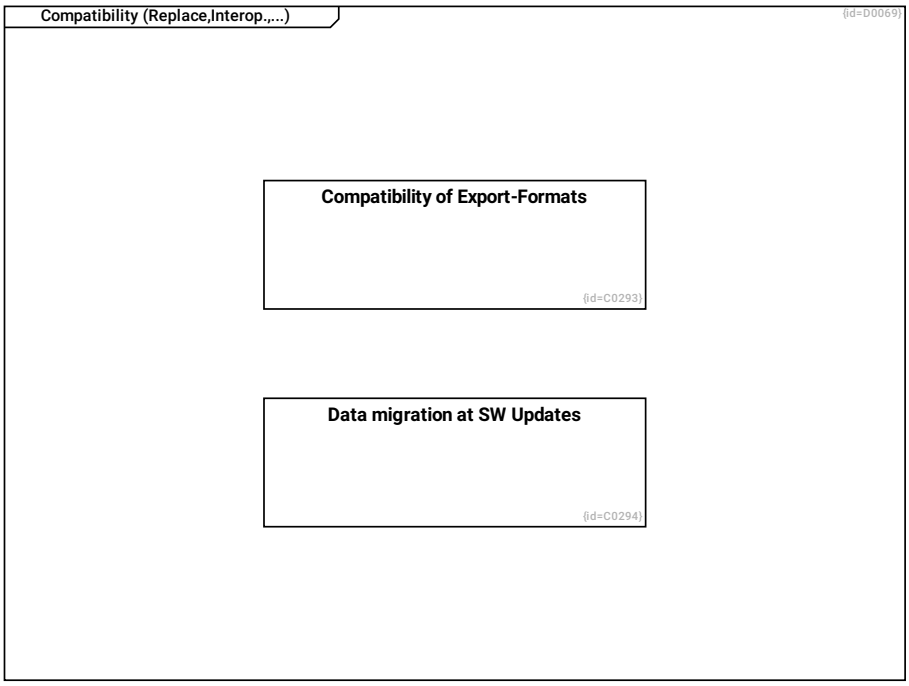
--> Reliability of Program operation R0489

1.10.1.2 Usability (Attactiveness,...)



Aesthetics of Auto-layouted diagrams C0291

1.10.1.3 Compatibility (Replace,Interop.,...)

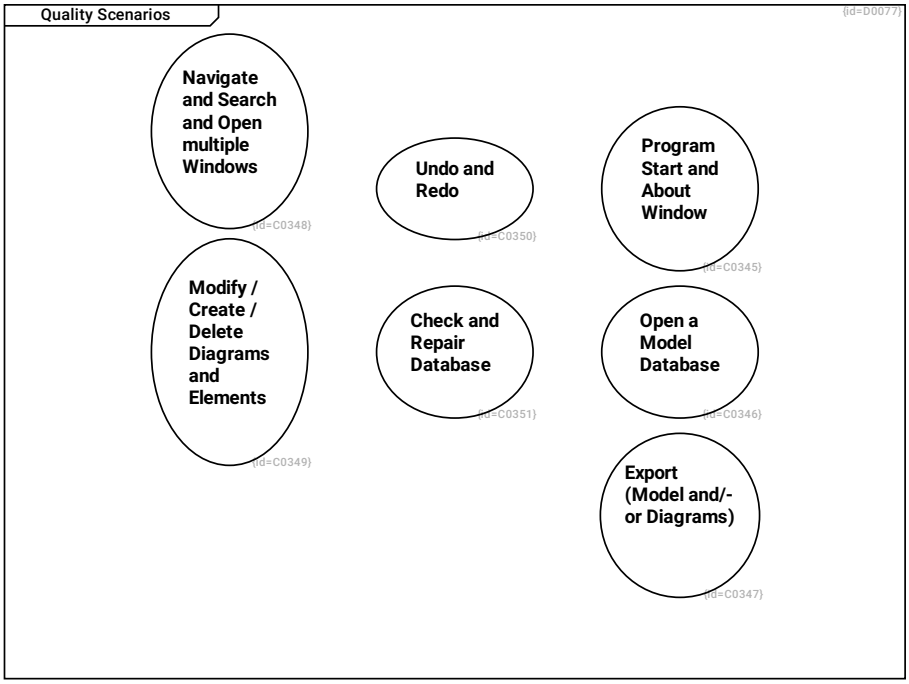


Compatibility of Export-Formats C0293

Data migration at SW Updates C0294

1.10.2 Quality Scenarios

This section shows scenarios which are of special importance for the quallity requirements stated in Section 1.10.1: Quality Tree.



Export (Model and/or Diagrams) C0347

Navigate and Search and Open multiple Windows C0348

Modify / Create / Delete Diagrams and Elements C0349

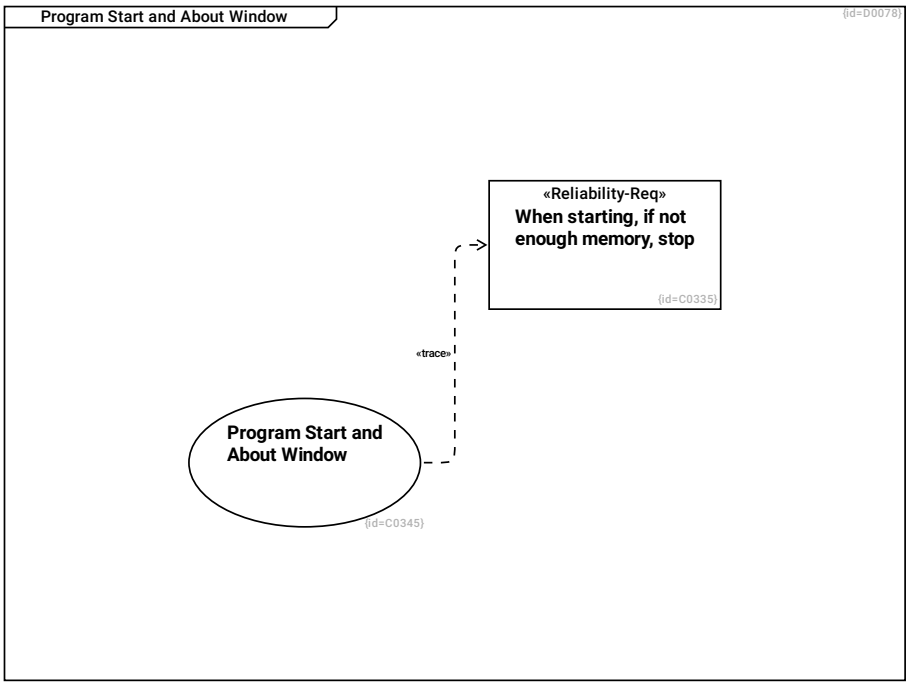
Open a Model Database C0346

Check and Repair Database C0351

Undo and Redo C0350

Program Start and About Window C0345

1.10.2.1 Program Start and About Window



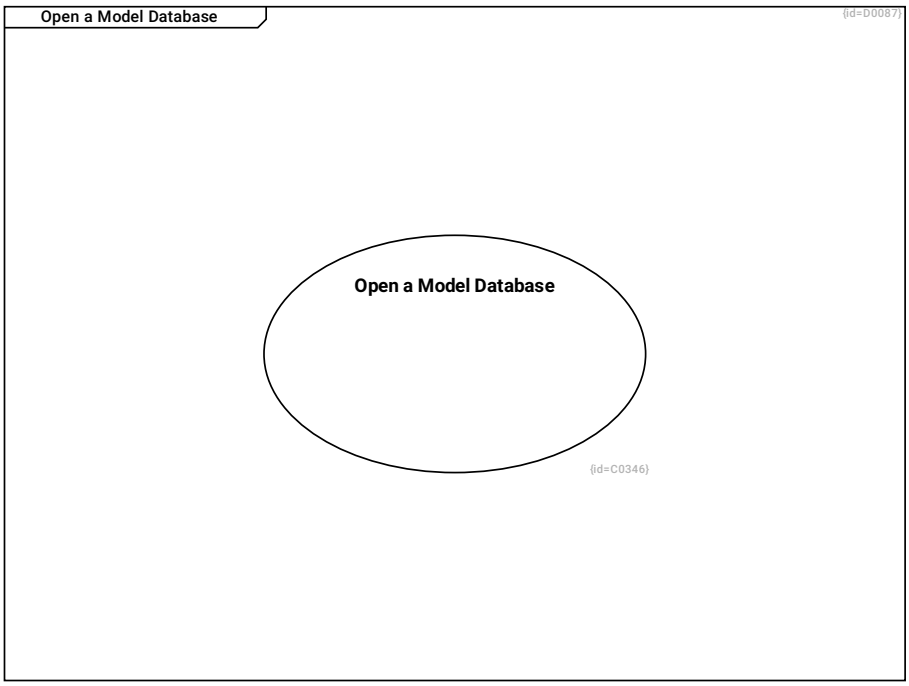
When starting, if not enough memory, stop C0335

When starting, if not enough memory is available to operate, the program shall inform the user and stop.

Program Start and About Window C0345

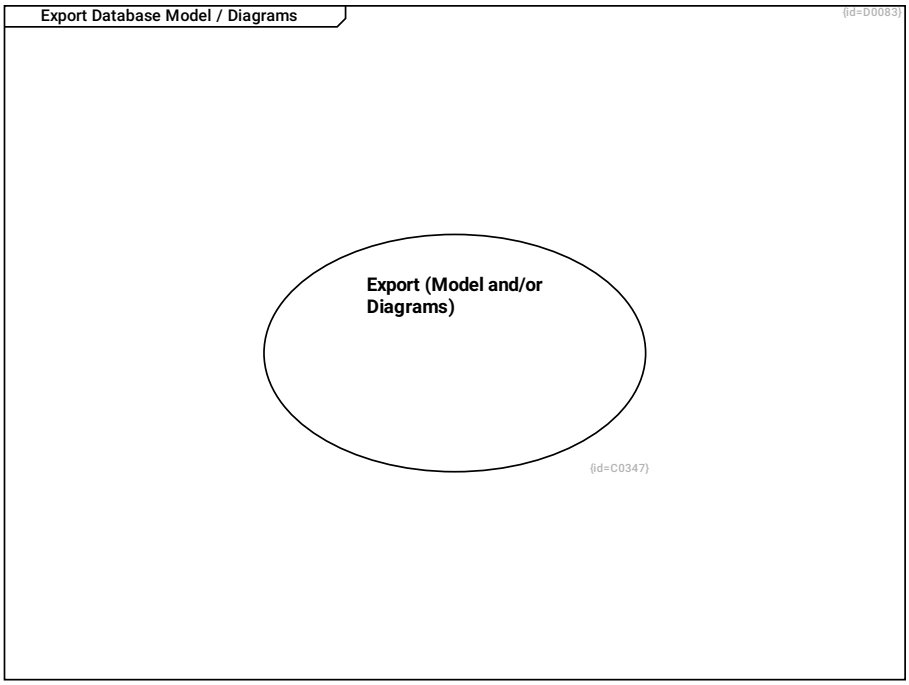
--> When starting, if not enough memory, stop R0490

1.10.2.2 Open a Model Database



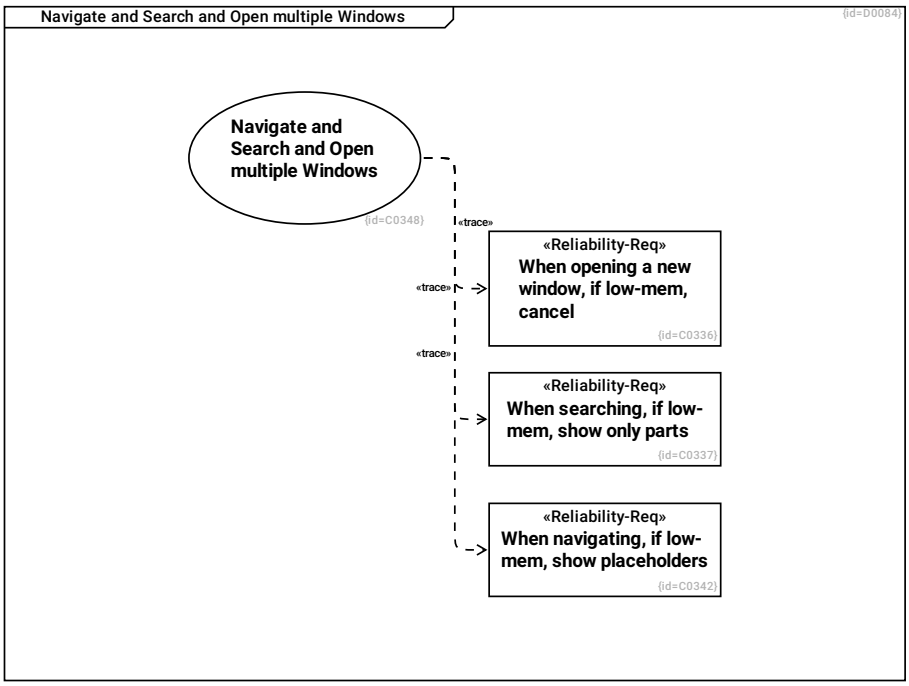
Open a Model Database C0346

1.10.2.3 Export Database Model / Diagrams



Export (Model and/or Diagrams) C0347

1.10.2.4 Navigate and Search and Open multiple Windows



Navigate and Search and Open multiple Windows C0348

--> When opening a new window, if low-mem, cancel R0491

--> When searching, if low-mem, show only parts R0492

--> When navigating, if low-mem, show placeholders R0493

When opening a new window, if low-mem, cancel C0336

When opening a new main window, if low-mem, the program shall inform the user and cancel the operation.

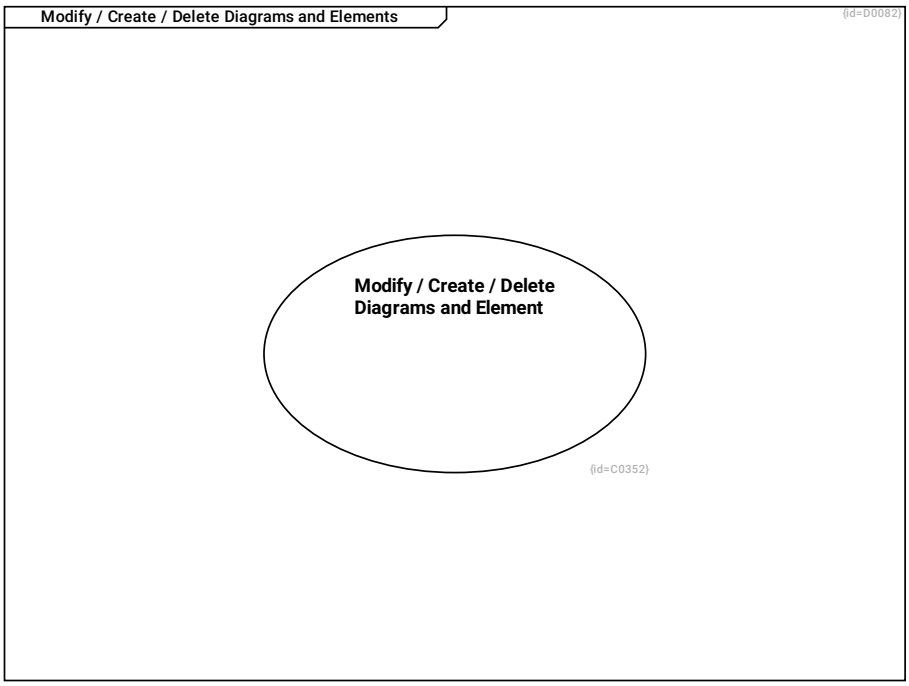
When searching, if low-mem, show only parts C0337

When searching, if not enough memory available for storing the search result list, the program shall indicate that some results are omitted and show only parts of the result list.

When navigating, if low-mem, show placeholders C0342

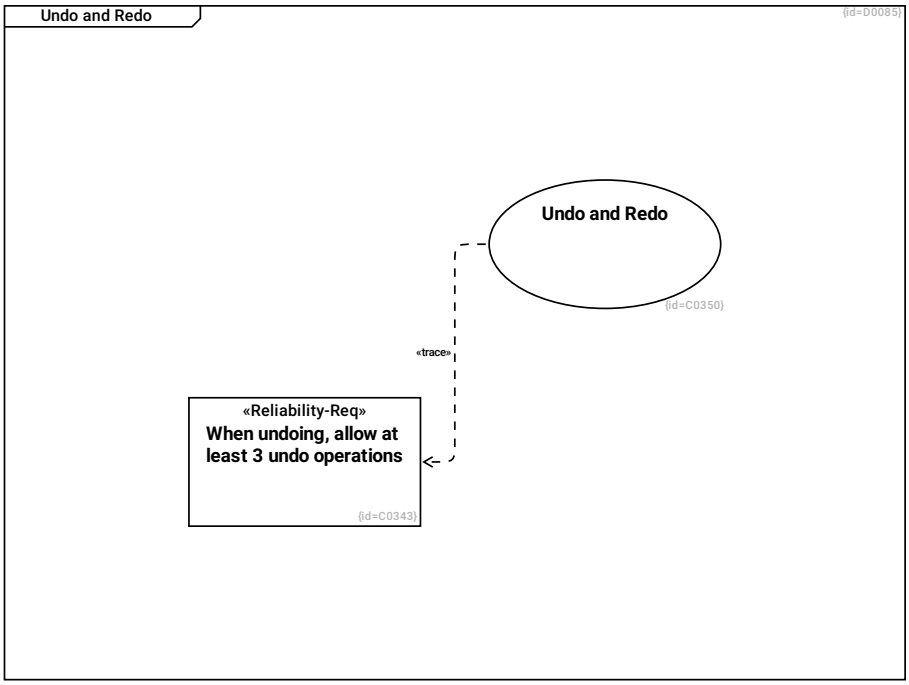
When navigating, if low-mem, the program shall show placeholder rectangles where rendering of diagrams is not possible.

1.10.2.5 Modify / Create / Delete Diagrams and Elements



Modify / Create / Delete Diagrams and Element C0352

1.10.2.6 Undo and Redo



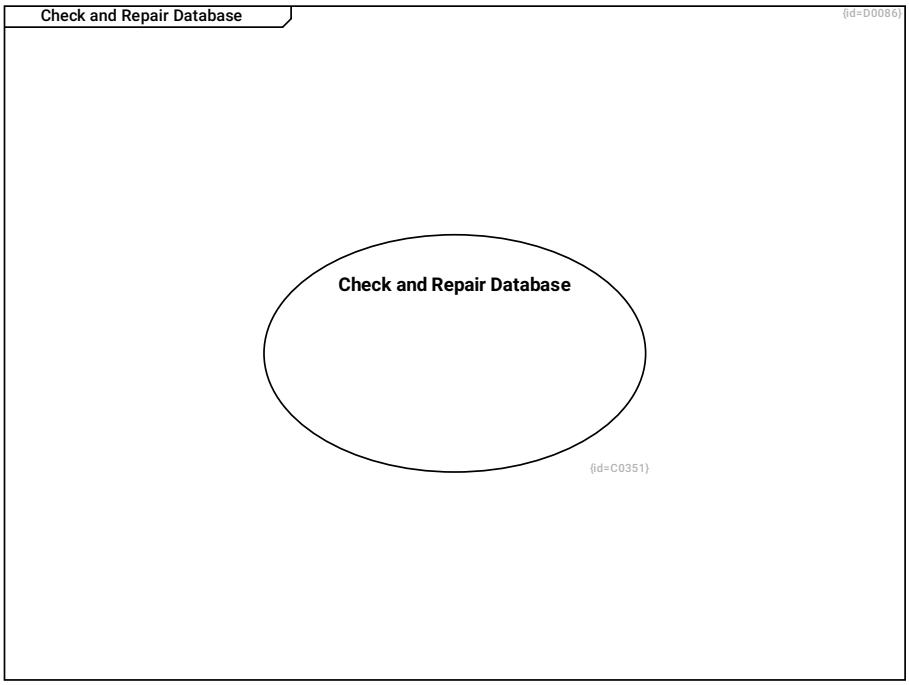
Undo and Redo C0350

--> When undoing, allow at least 3 undo operations R0494

When undoing, allow at least 3 undo operations C0343

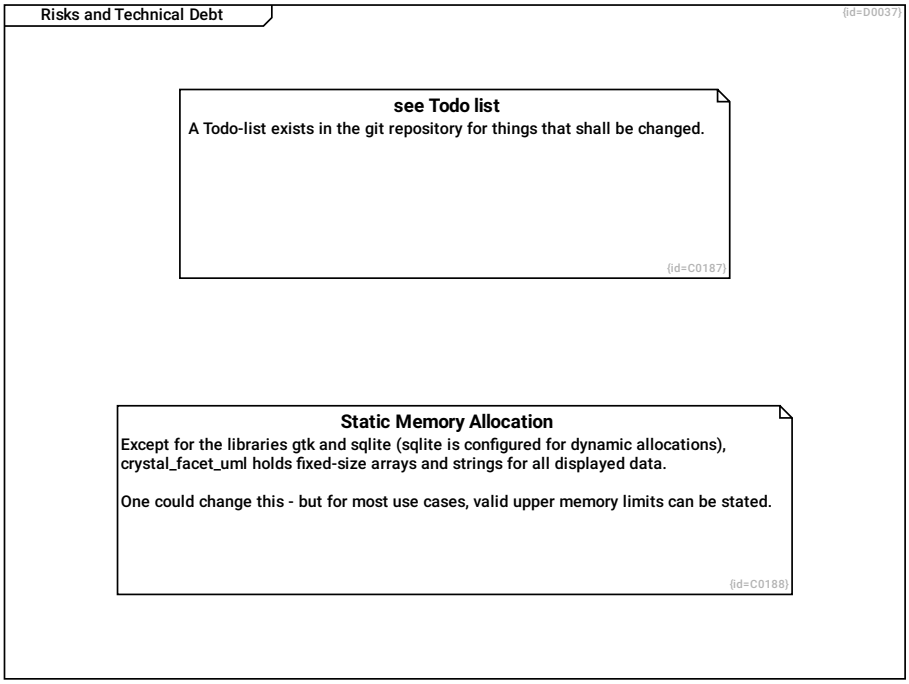
When undoing/redoing past actions, the program shall perform at least 3 undo/redo operations before informing the user that undo/redo is not possible anymore.

1.10.2.7 Check and Repair Database



Check and Repair Database C0351

1.11 Risks and Technical Debt



see Todo list C0187

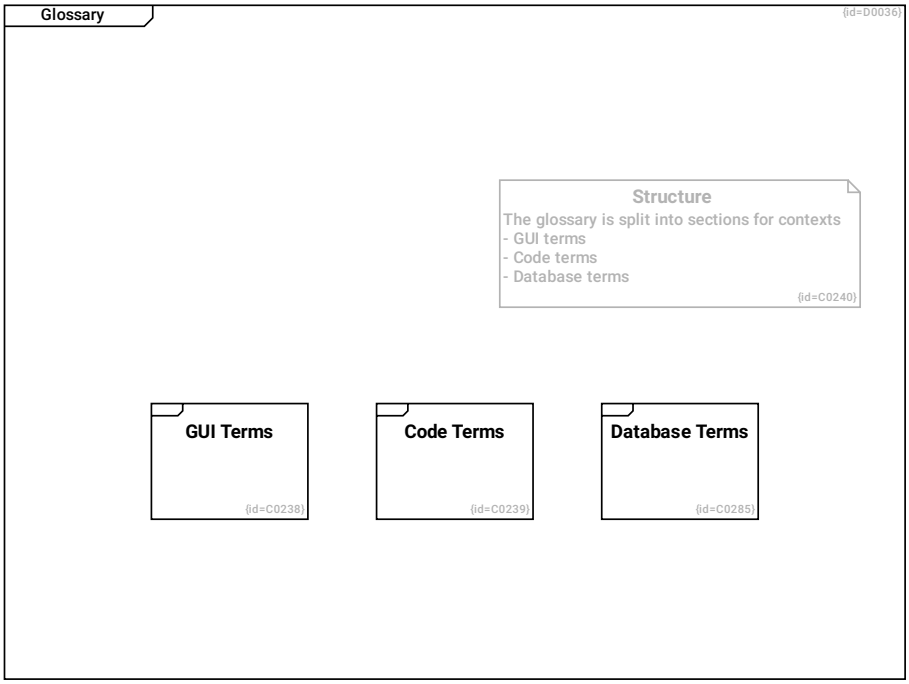
A Todo-list exists in the git repository for things that shall be changed.

Static Memory Allocation C0188

Except for the libraries gtk and sqlite (sqlite is configured for dynamic allocations), crystal_facet_uml holds fixed-size arrays and strings for all displayed data.

One could change this - but for most use cases, valid upper memory limits can be stated.

1.12 Glossary



Structure C0240

The glossary is split into sections for contexts

- GUI terms

- Code terms

- Database terms

GUI Terms C0238

see Section 1.12.1: GUI Terms.

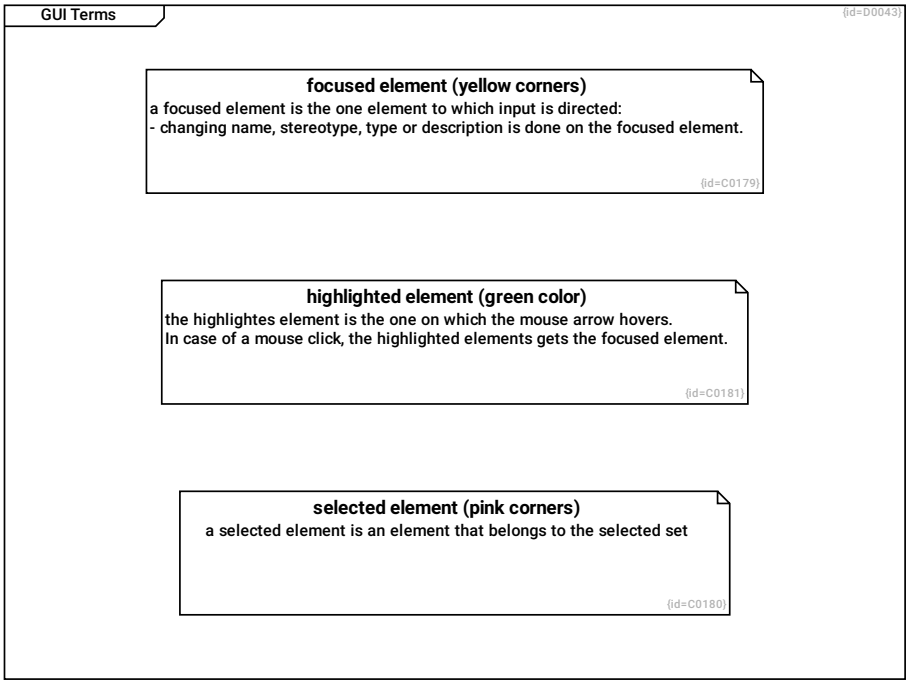
Code Terms C0239

see Section 1.12.2: Code Terms.

Database Terms C0285

see Section 1.12.3: Database Terms.

1.12.1 GUI Terms



focused element (yellow corners) C0179

a focused element is the one element to which input is directed:

- changing name, stereotype, type or description is done on the focused element.

highlighted element (green color) C0181

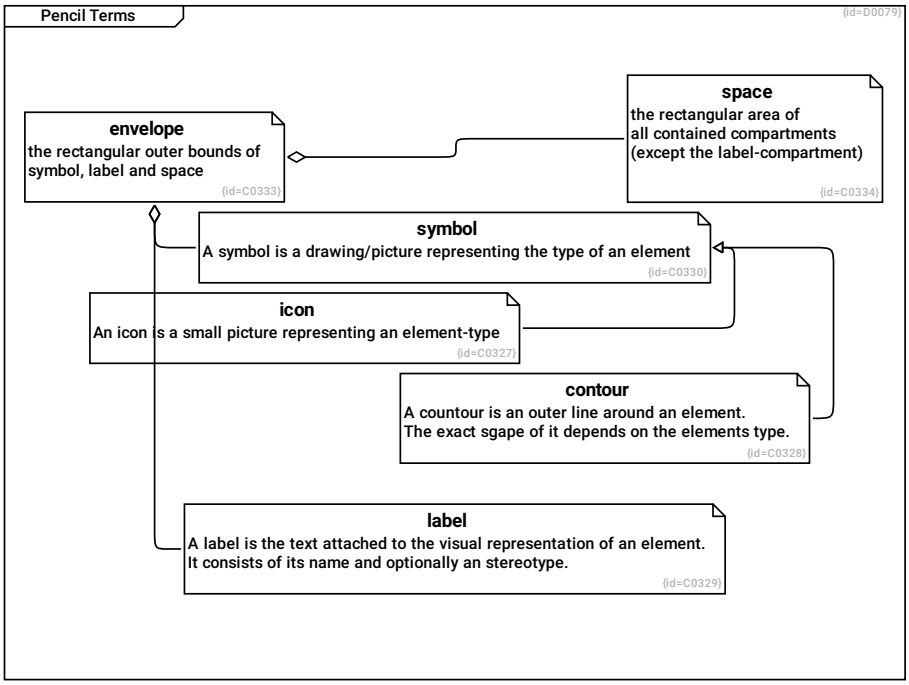
the highlightes element is the one on which the mouse arrow hovers. In case of a mouse click, the highlighted elements gets the focused element.

selected element (pink corners) C0180

a selected element is an element that belongs to the selected set

1.12.1.1 Pencil Terms

This diagram explains terms used mainly in pencil module.



envelope C0333

the rectangular outer bounds of symbol, label and space

--> symbol R0478

--> space R0479

--> label R0480

space C0334

the rectangular area of all contained compartments (except the label-compartment)

icon C0327

An icon is a small picture representing an element-type

--> symbol R0473

contour C0328

A countour is an outer line around an element. The exact sgape of it depends on the elements type.

--> symbol R0474

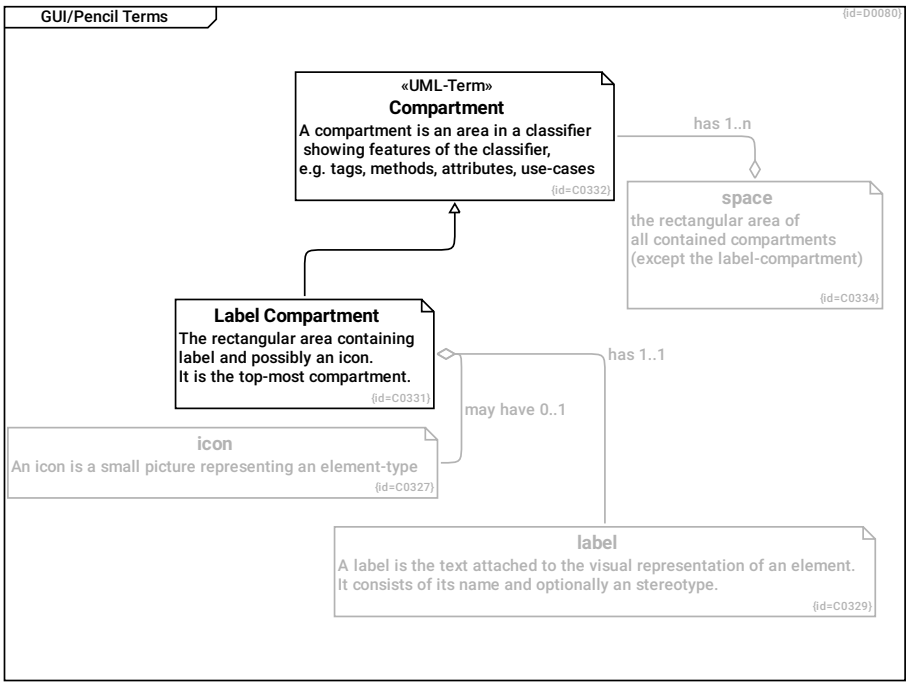
label C0329

A label is the text attached to the visual representation of an element. It consists of its name and optionally an stereotype.

symbol C0330

A symbol is a drawing/picture representing the type of an element

1.12.1.2 GUI/Pencil Terms



Compartment C0332

A compartment is an area in a classifier showing features of the classifier, e.g. tags, methods, attributes, use-cases

space C0334

the rectangular area of all contained compartments (except the label-compartment)

has 1..n --> Compartment R0481

Label Compartment C0331

The rectangular area containing label and possibly an icon. It is the top-most compartment.

--> Compartment R0475

may have 0..1 --> icon R0476

has 1..1 --> label R0477

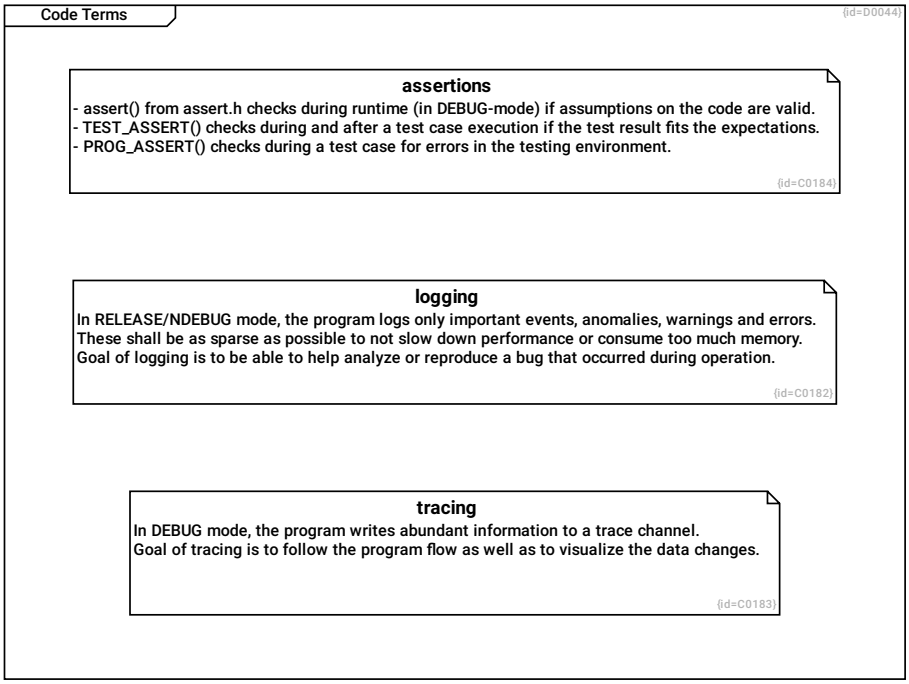
icon C0327

An icon is a small picture representing an element-type

label C0329

A label is the text attached to the visual representation of an element. It consists of its name and optionally an stereotype.

1.12.2 Code Terms



assertions C0184

- assert() from assert.h checks during runtime (in DEBUG-mode) if assumptions on the code are valid.
- TEST_ASSERT() checks during and after a test case execution if the test result fits the expectations.
- PROG_ASSERT() checks during a test case for errors in the testing environment.

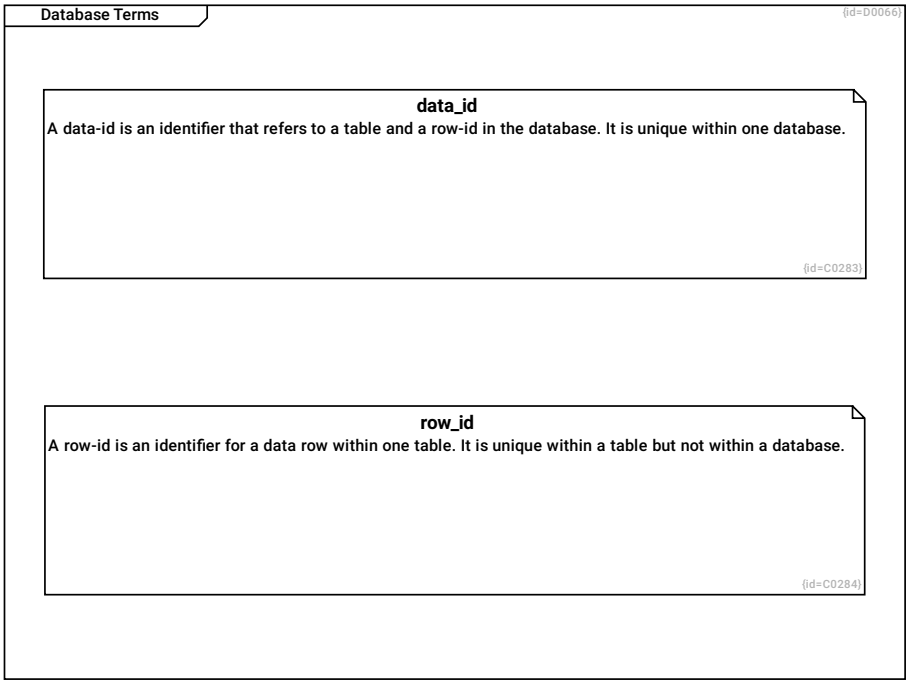
logging C0182

In RELEASE/NDEBUG mode, the program logs only important events, anomalies, warnings and errors. These shall be as sparse as possible to not slow down performance or consume too much memory. Goal of logging is to be able to help analyze or reproduce a bug that occurred during operation.

tracing C0183

In DEBUG mode, the program writes abundant information to a trace channel. Goal of tracing is to follow the program flow as well as to visualize the data changes.

1.12.3 Database Terms



data_id C0283

A data-id is an identifier that refers to a table and a row-id in the database. It is unique within one database.

row_id C0284

A row-id is an identifier for a data row within one table. It is unique within a table but not within a database.