Sample Design

to explain the Specification Card Approach for

Conceptual Software Design:

WissDB

A system

to store & retrieve Knowledge Items

Part 3

The Data Layer (Business Object Storage)

URI = D_ (WissDB / Design / The Data Layer)

This page is empty

Contents

Purpose of this Document **6**

Document Status **6**

Management Summary **8**

Notation and Terminology **9**

WissDB Business Object Services **10**

R_ Unknown_Semantics 11

Data Views for C_WissDB_DL_API 12

B_ WissDB_Database 12
V_ Locators 13
R_ Allowed and Disallowed Nesting of Units 14
V_ Subsets of the B_WissDB_Database 14
B_ URL 15
B_ Domain_Update_Format 15
B_ Process_Update 16
B_ Aspect_Update 16
B_ Association_Update 17

Services offered by C_WissDB_DL_API 18

R_ Requirements on DL_API API Implementation 18
R_ Project Locator Presentation 19
S_ Update_Process_Knowledge 20
S_ Update_Aspect_Knowledge 22
S_ Update_Result_Knowledge 24
S_ Ensure_Association 26
S_ Forget_Association 27
S_ Reclassify_Item 28
S_ Set_Content 30
S_ Get_Content 32
S_ Get_Skeleton 33
S_ Select_Results 35
S_ Select_Items 37
S_ Select_Processes 39
S_ Exclude_Aspects 40
S_ Exclude_Processes 41
S_ Exclude_Results 42
S_ Describe_DB_DomainValues 43
S_ Replace_or_create_DB_DomainValues 44
S_ Set_Alias 45
S_ Get_Alias 46
S_ Testsupport_New_DB 47
S_ Testsupport_DL_API_Reaction_Scope 48

This page is empty

Purpose of this Document

This paper might be part of a series of papers which constitute the conceptual and logical design of the WissDB Archive System which is

- to structure, store and index software engineering knowledge as well as results, such as best practices, sample design or black boxes containing reusable code
- and support the user in finding and retrieving such knowledge in a sufficiently selective, **activity, role, or association related** way.

Associations in this sense are binary associations of different, freely configurable semantics.

Basis of the Design are:

- D_(WissDB / Design / The Data Model)
- D_(WissDB / Design / The Application Service)

This Document's URI is:

- D_(WissDB / Design / The Data Layer)

Document Status

Revision	0.1
Last Update	06/08/2016
Author	Gebhard Greiter
Purpose	This document is to serve as a not too simple example how to create software design in form of Specification Cards, and how to present them in a Project Web (i.e. in HTML, well indexed and heavily hyper-linked across arbitrarily many documents).

This page is empty

Management Summary

This document is to specify the

WissDB Data Layer API

but may – later on – also contain implementation documentation.

All parts of the Data Layer API that have a name starting with **S_** are to be implemented as self-contained methods that, if called, have to have atomic effect on the systems's physical database.

The document contains essentially two parts:

- Section **Data Views** is a description of all necessary data views, especially of those that represent business objects.

  Each view is described and discussed only in as far as it is not already discussed in D_(WissDB / Design / Data Model).

  Objects to be stored are metadata, i.e. they are instances of one of these types:

  - **E_KnowledgeItem**
  - **E_Process**
  - **E_Role**
  - **E_Aspect**

  Each of these metadata objects may, or may not, have associated to it a value of type FILE (the so-called NodeValue). NodeValues are to be stored in the knowledge base itself.

- Section **Services** describes functionality available as a high level data storage interface on top of which at least the services described in D_(WissDB / Design / The Business Layer) can easily be implemented.

Notation and Terminology

Each concept specified in this design paper has an identifier starting with a prefix telling you the type of the concept. Prefix semantics are:

- C_… = A WissDB Component
- S_ = A service offered by a WissDB Component in form of at least an API
- m_… = Is so far a manual process only
- P_… = Business Process
- A_… = Actor (a user role or a system role)
- D_… = The logical name of a Document
- V_… = A logical data view on top of the physical data stored in the DL_API database. Such a view must be definable via SQL (so that standard reporting tools can be applied).
- B_… = Business Object type (a B_x is a V_x such that all instances of type B_x are owned by a unique WissDB Component. Ownership is the right to define this view.

WissDB Data Layer Services

Component:	**C_WissDB_DL_API** part of C_WissDB
Abstract:	This component is the abstract DBMS used e.g. by C_WissDB_API to implement a B_Knowledge_Base.
Has to:	**Own and manage** – B_WissDB_Database. **Implement** the following services: – S_Update_Process_Knowledge – S_Update_Aspect_Knowledge – S_Update_Result_Knowledge – S_Ensure_Association – S_Forget_Association – S_Set_Content – S_Get_Content – S_Get_Skeleton – S_Select_Results – S_Select_Items – S_Select_Processes – S_Exclude_Aspects – S_Exclude_Processes – S_Exclude_Results – S_Describe_DB_DomainValues – S_Replace_or_create_DB_DomainValues – S_Set_Alias – S_Get_Alias – S_Testsupport_New_DB – S_Testsupport_DL_API_Reaction_Scope

	Make all these functions available via a well documented API that could be used by any kind of application. **Each of these functions must be protectable via** C_Access_Control **as a separate resource** (so that, depending on their role, users and applications might be restricted to use only some of these functions). **User Roles to be supported** by C_WissDB_DL_API are – A_Application – A_Tester
Because of:	The Data Layer of WissDB should be reusable for applications similar but not identical to the Business Layer of WissDB.

R_ Unknown_Semantics

The code implementing C_WissDB_DL_API must not depend on any concrete semantics of the values that are, at any given time, valid values for

- D_ItemType
- D_PracticeType
- D_ViewType
- D_AbstractionType
- D_UsageType
- D_CorrelationType.

Data Views for C_WissDB_DL_API

B_ WissDB_Database

Data view owner is C_WissDB_DL_API

Value Specification:

- For each WissDB installation there is one and only one such value.
- The data structures it is to support are discussed in D_(WissDB/ Result/ WissDB/ Design/ Data Model).
- This value can be updated and read only via the services described in this document (so that applications do not know whether this database is implemented via Enabler or a relational DBMS). They need to be general enough to implement a B_Knowledge_Base.
- B_WissDB_Database takes the form of a tree-like structured unit containing subunits that are instances of one of the following entity types (semantics unknown):
  - E_Process
  - E_Role
  - E_Practice
  - E_Result
  - E_KnowledgeItem
  - E_Aspect
  - E_DomainValues
  - E_Alias
- The treelike structure is a logical structure (not a physical one).
- In addition to the tree-like structure (given by attributes A_Loc) there is correlation structure implemented in the form of binary relations
  - R_Is_related_to
  - R_Is_keyword_for

V_ Locators

When reading the rest of this document, please keep in mind the following definitions:

A **Locator** is a value of type D_Locator (which is a sequence of D_Name values each followed by a slash).

Reserved names are:

- Aspect
- Result
- Role
- Process
- Package
- Description
- Selector
- Structure

A locator X is called

- **Project locator,** if it does not contain a reserved name,
- **Aspect locator,** if the last reserved name in X is **Aspect**,
- **Process locator**, if the last reserved name in X is **Process**,
- **Role locator**, if the last reserved name in X is **Role**,
- **Result Locator**, if the last reserved name in X is **Result**,
- **Description Locator**, if the last reserved name in X is **Description**, **Selector**, or **Structure**.
- **Package Locator,** if matching the pattern **Package**/ **D_Name.**

X is said to by a **preLocator** of another locator Z if either X = Z or X is a prefix of Z.

A **true preLocator** of Z is a preLocator of Z that is shorter than Z.

A locator is said to be **in use** if and only if the B_WissDB_Database contains on object Y such that X is preLocator of Y.a_Loc.

The implementation of C_WissDB_DL_API has to guarantee that each result locator in use has a preLocator that is a project locator.

Each project locator has an **Alias** that is a positive integer. It is to be created automatically and is to be seen as being **a variable of type Project Locator**. The value of this variable is case-sensitive and may be changed using S_Set_Alias.

Locators **L** being prefixed by a project locator need to be stored in the form **N**/ **X** such that **N** is the alias of a project locator and **X** is the unique rest of **L** always starting with prefix **Aspect/**, **Process/**, **Role/** or **Result/**. We then call **N**/ **X** the **Standard Locator** currently equivalent to locator L.

R_ Allowed and Disallowed Nesting of Units

- Given any locator **X/ Aspect**, **X/ Role**, or **X/ Result**, the **X** may contain **Result** or **Process** (but not **Aspect** or **Role**).
- Given any locator **X/ Process**, **X** may contain **Result** or **Process** (but not **Aspect**, **Role**, or **Process**).
- The default project unit(**1**) has three direct subunits. Their locators are:
  - **1/ Aspect**
  - **1/ Process**
  - **1/ Role**
  - **1/ Package**

  The code implementing the B_WissDB_Database is to guarantee that these locators are always in use (so that the corresponding units exist even when they are empty).

  The code will further have to guarantee that unit(**1/ Package**) is the only unit having the name **Package**.

V_ Subsets of the B_WissDB_Database

When reading the rest of this document, please keep in mind the following definitions:

- **B_ ALL_Knowledge** is the set of all records X in the database that have an attribute a_Loc.
- **B_ ALL_Aspects** is the subset of records such that a_Loc is an Aspect Locator.
- **B_ ALL_Processes** is the subset of records such that a_Loc is a Process Locator.
- **B_ ALL_Roles** is the subset of records such that a_Loc is a Role Locator.
- **B_ ALL_Results** is the subset of records such that a_Loc is a Result Locator.
- **B_ ALL_Packages** is the subset of records such that a_Loc is a Package Locator.

The implementation of C_WissDB_DL_API is to guarantee that

- Each element of B_ALL_Aspects is a E_Aspect.
- Each element of B_ALL_Processes is a E_Process.
- Each element of B_ALL_Roles is a E_Role.
- Each element of B_ALL_Results is a E_Result.
- Each element of B_ALL_Packages is a E_KnowledgeItem.

B_ URL

Data view owner is C_WissDB_DL_API

Value Specification:

Each B_URL is a string starting with one of the following prefixes:

- http://
- ftp://
- file://
- x-unc://

If starting with file:// it is to match the template file://host/path (i.e. it is not allowed to contain a drive letter).

B_ Domain_Update_Format

Data view owner is C_WissDB_DL_API

Value Specification:

This is ASCII text that may contain lines of the form

```
-   DomainName,ValueName
+   DomainName,ValueName,Semantics
```

with the plus or minus sign in column 1 and the DomainName starting in column 4.

As far as there are more lines in the text they are assumed to be comment.

B_ Process_Update

Data view owner is C_WissDB_DL_API

Value Specification:

This is ASCII text that may (or may not) contain lines of the form

```
=  Un=D_Locator
=  Pm=D_Locator

-  Role Locator
+  Role Locator

-  Process Locator
+  Process Locator
```

Here **Pn** and **Rm** (always starting in column 4 of the text line) are abbreviation identifiers such that **n** and **m** are unique positive integer.

Abbreviation identifiers can be used to replace frequently occurring preLocators.

B_ Aspect_Update

Data view owner is C_WissDB_DL_API

Value Specification:

This is ASCII text that may (or may not) contain lines of the form

```
=  An=D_Locator

-  Aspect Locator
+  Aspect Locator
```

Here **An** (always starting in column 4 of the text line) is an abbreviation identifier such that **n** is positive integer.

Abbreviation identifiers can be used to replace frequently occurring preLocators.

B_ Association_Update

Data view owner is C_WissDB_DL_API

Value Specification:

This is ASCII text that may (or may not) contain lines of the form

```
=  An=D_Locator/Aspect
=  Rm=D_Locator/Result

-  Aspect Locator
   .  Result Locator
   .  Result Locator

+  Aspect Locator
   .  Result Locator
   .  Result Locator

-  Item Locator
   .  Relation: Item Locator

+  Item Locator
   .  Relation: Item Locator
```

Here **An** and **Rn** (always starting in column 4 of the text line) are abbreviation identifiers such that **n** and **m** are positive integers.

Abbreviation identifiers can be used to replace frequently occurring preLocators.

Each line with a plus or minus sign in column 1 has to be preceeded by an empty line and can be followed by any number of lines starting with a dot in column 4 and two spaces between the dot and the rest of the line.

Each **Relation** is to be a D_CorrelationType value which (in case of +) must be a currently valid value.

Services offered by C_WissDB_DL_API

This section contains the logical design of the WissDB DL_API API. We start by describing important requirements on the form this API has to take:

R_ Requirements on DL_API API Implementation

Each call of one of the services (= functions) specified in the following is to guarantee that the value of the B_WissDB_Database remains unchanged if the ReturnCode value is neither RC_ok nor RC_seeWarnings.

Another important requirement on these services is that they must be powerful enough to implement C_WissDB_API in a painless way.

Each service is specified with two specific output parameters called AppErrors and ReturnCode:

- **Out: AppErrors**
- **Out: ReturnCode** is one of the following values:
  - RC_ok
  - RC_syserror
  - RC_seeAppErrors

Depending on the programming langue used, these two parameters may be implemented quite differently. In Java, e.g. at least RC_syserror would best be implemented in form of a runtime exception, whereas RC_seeAppErrors might be implemented differently from case to case:

- If an exception such as ObjectNotFound is enough error description, to throw an exception implementing both the returncode RC_seeAppErrors and also the AppErrors would be fine (and should be preferred).
- If however syntax errors (or logical errors hard to explain) are detected somewhere, the AppErrors need to be a dynamically generated text the application could then write to either a file or the console.

A value of type **AppErrors** is a string containing one or more text sections of the form

AppError in S_x: ErrorType: Diagnostic data

or

Warning out of **S_x:** any text

such that **S_x** is the service activated by an application of the DL_API API (it must not be the name of an auxiliary method that is part of the hidden implementation of this service: applications are to see the system they use as a black box).

The service is to return RC_seeAppErrors if and only if parameter AppErrors is describing at least one AppError.

The service should describe as many AppErrors as possible before returning.

R_ Project Locator Presentation

With the exception of the two services

- S_Set_Alias
- S_Get_Alias

the API of C_WissDB_DL_API is not allowed to either accept or show project locators in any other form than in the form of a number.

S_ Update_Process_Knowledge

Component:	**S_Update_Process_Knowledge** part of C_WissDB_DL_API
Abstract:	This is the function to update the B_WissDB_Database with respect to process knowledge. Parameters: – **In: Process** Is ASCII text of type B_Process_Update. – **In: Request** Must be CREATE, or REPLACE, or FORGET. Value CREATE means: Return with RC_seeAppErrors if at least one of the locators in **Process** is in use already. – **Out: AppErrors** If the value of B_WissDB_Database did not change, users should find a description here telling them why. Describe an AppError here if Request = CREATE though Process is containing lines with a minus sign in column 1. Describe an AppError here if Request = FORGET though Process is containing lines with a plus sign in column 1. Create a warning whenever an object to be deleted does not exist. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors – RC_seeWarnings
Has to:	Implement the required update of the B_WissDB_Database, i.e. ▪ Delete in B_WissDB_Database all objects X of type E_Process such that some preLocator of X is listed in Process with a minus sign. ▪ For each object Z in B_WissDB_Database.E_Result reduce Z.a_of to the effect that the new value of Z.a_of is the longest preLocator of the old Z.a_of value that is still in use. Set Z.a_of to NULL if this preLocator is not a role locator.

	▪ Then, for each line listed in Process with a plus sign, create the corresponding E_Process object. Return with an AppError is the given Role Locator is currently not in use. RC_ok is allowed only if the **Update** requested could be implemented fully. If the ReturnCode is different from RC_ok, the value of the knowledge base did not change.
Because of:	Applications of C_WissDB_DL_API need a way to update the B_WissDB_Database with respect to process knowlege described therein.

S_ Update_Aspect_Knowledge

Component:	**S_Update_Aspect_Knowledge** part of C_WissDB_DL_API
Abstract:	This is the function to update the B_WissDB_Database with respect to the knowledge classification schema (which is based on so-called **Aspects**: read section 2.2 in D_ (WissDB/ Result/ WissDB/ Design/ Data Model). Parameters: – **In: Update** Is ASCII text of type B_Aspect_Update. – **In: Request** Must be CREATE, or REPLACE, or FORGET. Value CREATE means: Return with RC_seeAppErrors if at least one of the locators in **Aspect** is in use already. – **Out: AppErrors** If the value of B_WissDB_Database did not change, users should find a description here telling them why. Describe an AppError here if Request = CREATE though **Update** is containing lines with a minus sign in column 1. Describe an AppError here if Request = FORGET though **Update** is containing lines with a plus sign in column 1. Create a warning whenever an object to be deleted does not exist. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors – RC_seeWarnings
Has to:	Implement the required update of the B_WissDB_Database. RC_ok is allowed only if the **Update** requested could be implemented fully. If the ReturnCode is different from RC_ok, the value of the knowledge base did not change.
Because of:	Applications of C_WissDB_DL_API need a way to update the B_WissDB_Database

	with respect to aspects to be supported.

S_ Update_Result_Knowledge

Component:	**S_Update_Result_Knowledge** part of C_WissDB_DL_API
Abstract:	This is the function to update the B_WissDB_Database with respect to relationships of type R_Is_related_to or R_Is_keyword_for. Parameters: – **In: Update** Is ASCII text of type B_Association_Update. – **In: Request** Must be CREATE, or REPLACE, or FORGET. – **Out: AppErrors** If the value of B_WissDB_Database did not change, users should find an explanation here. Describe an AppError here if Request = CREATE though **Update** is containing lines with a minus sign in column 1. Describe an AppError here if Request = FORGET though **Update** is containing lines with a plus sign in column 1. Create a warning whenever a relationship to be deleted does not exist. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors – RC_seeWarning
Has to:	Implement the required update of the B_WissDB_Database. X.a_LastUpdate is now the current date for each E_Result X such that some part of unit(X) was referred to in **Update**. RC_ok is allowed only if the **Update** requested could be implemented fully. If the ReturnCode is different from RC_ok, the value of the knowledge base did not change.
Because of:	Applications of C_WissDB_DL_API need a way to update the B_WissDB_Database

	with respect to Result data.

S_ Ensure_Association

Component:	**S_Ensure_Association** part of C_WissDB_DL_API
Abstract:	This is the functions to create a new instance of type R_Is_related_to. Parameters: – **In: Locator_A** Is an instance of type E_KnowledgeItem.A_Loc. – **In: Locator_B** Is an instance of type E_KnowledgeItem.A_Loc. – **In: CorrelationType** Is an instance of type D_Correlation. – **Out: AppErrors** Must not be empty if the correlation could not be created (and did not yet exist). – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors
Has to:	Ensure the existence of a relationship X satisfying ▪ X.a_A = **Locator_A** ▪ X.a_B = **Locator_B** ▪ X.a_Correlation = **CorrelationType** **Please note:** If exactly this relationship already exists, the function is to return RC_ok.
Because of:	C_WissDB_API needs a way to create associations of type R_Is_related_to.

S_ Forget_Association

Component:	**S_Forget_Association** part of C_WissDB_DL_API
Abstract:	This is the functions to forget instance of type R_Is_related_to. Parameters: – **In: Locator_A** Is an instance of type E_KnowledgeItem.A_Loc. – **In: Locator_B** Is an instance of type E_KnowledgeItem.A_Loc. – **In: CorrelationType** Is an instance of type D_Correlation. – **Out: AppErrors** Must not be empty if at least one of the two given locators is not an element of E_KnowledgeItem.A_Loc. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors
Has to:	Ensure that, from now on, a relationship X satisfying ▪ X.a_A = **Locator_A** ▪ X.a_B = **Locator_B** ▪ X.a_Correlation = **CorrelationType** will not exist.
Because of:	C_WissDB_API needs a way to let B_WissDB_Database forget associations of type R_Is_related_to.

S_ Reclassify_Item

Component:	**S_Reclassify_Item** part of C_WissDB_DL_API
Abstract:	This is the function to update the attributes of an existing E_KnowledgeItem and/or to change the set of aspects associated to this item. Parameters: – **In: Locator** Is an instance of type E_KnowledgeItem.A_Loc. – **In: Type** Is either NULL or an instance of type D_ItemType. – **In: View** Is either NULL or an instance of type D_ViewType. – **In: Abstraction** Is either NULL or an instance of type D_AbstrRequest. – **In: Usage** Is either NULL or an instance of type D_UsageType. – **In: Practice** Is either NULL or an instance of type D_PracticeType. – **In: Process** Is either NULL or a process locator (that must exist in B_WissDB_Database). – **In: Aspects** Is either NULL or a – possibly empty – set of aspect locators (all of them must exist in the B_WissDB_Database) – **Out: AppErrors** Is to describe AppErrors if the given Locator is currently not in use, or if one of the new attribute values is a value currently not allowed according to the data found in the table B_WissDB_Database.E_DomainValues. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors

Has to:	Update the B_WissDB_Database to the effect that the E_KnowledgeItem instance identified by the given Locator will get new attribute values. Attribute values with the restriction NOT NULL will be updated only in as far the new value is not NULL. The same holds for attributes not existing on the item in question because it is not a Result/ (or a practice instance). If the item to be updated is of type E_Result or E_Practice, the function has to return with RC_seeAppErrors if at least on of the locators in **ofProcess** or **Aspects** is not an element of E_Process.A_Loc resp. E_Aspect.A_Loc. Whether an item is a Result/ or not can be can be learned from the Locator. **Please note:** Depending on the new value for A_PracticeType, the item updated may be upgraded from E_Result to E_Practice, or may be downgraded from E_Practice to E_Result.
Because of:	C_WissDB_API needs a way to reclassify knowledge items. To support upgrade resp. downgrade of E_Result resp. E_Practice instances is important because technology is changing fast, and so a Result that is today a Best Practice may no longer deserve this quality one or two years later.

S_ Set_Content

Component:	**S_Set_Content** part of C_WissDB_DL_API
Abstract:	Each object X in the B_Knowledge_Base that has an attribute named a_Loc may also have a presentation of type FILE or URL. This value is referred to as the **Content** addressed by the locator **X.a_Loc** (or, equivalently, as **the value of unit(X)**). Parameters: – **In: Locator** Is to be a locator for data of type E_Aspect, E_Process, E_Role, or E_KnowledgeItem. – **In: Content** Is either NULL or the B_URL to an existing file (of which to store a copy in form of a NodeValue). – **In: Author** Is either NULL or a value of type D_EmailAddress. – **Out: AppErrors** Is to describe an AppError if the given Locator is currently not **in use** or if the file identified by the B_URL could not be read. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors
Has to:	Ensure that, from now on, – **Content** is the new content of the given **Locator L**, and – NULL is the content of all locators of which **L** is a true preLocator. – **X**.a_contact = **Author** and **X**.a_LastUpdate = **today** for each E_Result **X** such that **X**.a_Loc is preLocator of **L**. Reading content created this way is possible via S_Get_Content.

	Note: For each element **X** of B_WissDB_Database.E_Result the attribute **X**.a_since is set only once – it is the day the record was created (so that we see how old a result seen as a unit actually is).
Because of:	C_WissDB_API needs a way to store content (or an URL seen to be a pointer to content). One reason for using an URL instead of a file may be that the content in question is to be dynamic content created on request, or being stored, by some other system.

S_ Get_Content

Component:	**S_Get_Content** Part of C_WissDB_DL_API
Abstract:	Given a locator **X in use**, this is the function to ask the B_WissDB_Database for the content that is referred to as **the value of unit(X)**. Parameters: – **In: Locator** Is to be a D_Locator currently in use. – **In: storeAt** Is to be the B_URL to the file that shall be created or re-created to represent a copy of the content addresses by the Locator (in case of content value NULL this file, if existing, is to be deleted). – **Out: Content** Is either NULL or a the B_URL to a file just created or re-created to represent the content addressed by the given **Locator**. – **Out: AppErrors** Is to describe an AppError – if the given Locator is not in use – or if **storeAt** could not be created (if the **Content** is not NULL) – or if **storeAt** could not be destroyed (if the **Content** is NULL). – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors
Has to:	Retrieve the content addressed by the given **Locator** (or – if this content is NULL – ensure that the file or folder **storeAt** does not exist any more).
Because of:	C_WissDB_API needs a way to retrieve the value of **unit(X)**, **X** a locator in use.

S_ Get_Skeleton

Component:	**S_Get_Skeleton** part of C_WissDB_DL_API
Abstract:	This is the service to be used for reading the unit-oriented structure of objects existing in the B_WissDB_Database. To explain this service we need some definitions: ▪ In the **node-oriented view** an object is a record of type E_Process, E_Role, E_Aspect, or E_KnowledgeItem. ▪ Every such object X has a locator (which is the value found in X.a_Loc). ▪ In the **unit-oriented view** the object is the tree of all object that have a locator identical to or prefixed by the locator of X. ▪ The **unit-oriented structure** of X (also called the **skeleton** of X) is defined to be the set of all these locators presented in form of a lexicographically ordered sequence of strings. ▪ A **pseudo object** is one of the following sets of objects in the sense above: – B_ALL_Knowledge – B_ALL_Processes – B_ALL_Roles – B_ALL_Aspects – B_ALL_KnowledgeItems – B_ALL_Results – B_ALL_Practices ▪ The **skeleton of a pseudo object** is the union of the skeletons of all the objects contained in the pseudo object. Parameters: – **In: Locator** Is to be the locator for an object X that is either a pseudo object, or an object that has in the B_WissDB_Database, an attribute X.a_Loc. The locator for a pseudo object is /NameOfPseudoObject (e.g. /E_Roles). – **Out: Skeleton** Is either NULL or a sequence of N strings, N a positive number. – **Out: AppErrors** In case of Skeleton = NULL, the AppErrors must not be empty. The Skeleton returned must be either complete or NULL. – **Out: ReturnCode** is one of the following values: – RC_ok

	– RC_syserror – RC_seeAppErrors
Has to:	Retrieve the skeleton of the object or pseudo object identified by the given **Locator** value.
Because of:	C_WissDB_API needs a way to retrieve the unit-oriented structure of all the objects stored in the B_WissDB_Database. At least for generating test cases it will be helpful to have a service that is able to show the complete skeleton of the knowledge base itself.

S_ Select_Results

Component:	**S_Select_Results** part of C_WissDB_DL_API
Abstract:	Using this functions you can search the B_WissDB_Database for Result objects satisfying given search criteria. Parameters: – **In: SeeType** Is to be a set of strings (as they are allowed to be checked in the Type section of a B_Query_Specification). – **In: SeeScope** Is to be a set of strings (as they are allowed to be checked in the Scope section of a B_Query_Specification). – **In: SeeView** Is to be a set of strings (as they are allowed to be checked in the View section of a B_Query_Specification). – **In: SeeAbstraction** Is to be a set of strings (as they are allowed to be checked in the Abstraction section of a B_Query_Specification). – **In: SeeUsage** Is to be a set of strings (as they are allowed to be checked in the Usage section of a B_Query_Specification) – **In: SeeAspect** Is either NULL or a set of strings representing Aspect Locators (even the locator Aspect/ itself is allowed). Note: If one of these locators is a preLocator of another one, this other one is redundant and will be ignored. – **In: SeeRole** Is either NULL or a not empty set of strings representing Role or Process Locators. Note: If one of these locators is a preLocator of another one, this other one is redundant and will be ignored.

	– **Out: Skeleton** Is either NULL or a not empty set of values of type D_Locator, in which case it is to satisfy the following condition: For each element **X** of B_WissDB_Database.E_Result, **X**.a_Loc is element of the **Skeleton** if and only if – **X**.a_Type is element of **SeeType**, – **X**.a_ViewType is element of **SeeView**, – **X**.a_AbstractionType is element of **SeeAbstraction**, – **X**.a_UsageType is element of **SeeUsage**, – **X**.a_PracticeType is element if SeePractice (if X is not a E_Practice, the value if this attribute is assumed to be **Result**), – R_Is_keyword_for says that **X** is associated with a keyword **K** such that at least one element of **SeeAspect** is a prefix of **K**, and – One of the process locators found in **SeeRole** is preLocator of **X**.a_of (so that the **X** represents a result of this process), or X.a_of is the locator of a E_Process **P** such that an element of **SeeRole** is preLocator of **P**.a_RoleLoc. – **Out: AppErrors** If **SeeRole** is containing locators currently not in use, or not being a Role Locator, an AppError is to explain this. If at least one of the locators found in **SeeAspect** is not an Aspect Locator currently in use, an AppError is to explain this. AppErrors is to contain warnings if at least one string found in **SeeType**, **SeeScope**, **SeeView**, **SeeAbstraction**, or **SeeUsage** is a value not allowed therein and therefore ignored. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors – RC_seeWarnings
Has to:	Create the Skeleton specified above as a subset of D_Locator. Then, if this set is not empty, return RC_ok in **ReturnCode** and this set in **Skeleton**. If the set is empty, return NULL in **Skeleton** and again RC_ok in **ReturnCode**. Return NULL in **Skeleton** and RC_seeAppErrors **AppErrors** is explaining errors.
Because of:	C_WissDB_API needs a way to implement S_Search_for_Knowledge.

S_ Select_Items

Component:	**S_Select_Items** part of C_WissDB_DL_API
Abstract:	This is the function to ask the B_WissDB_Database for items of a specific type. Parameters: – **In: SeeType** Is to be a set of strings (as they are allowed to be checked in the Type section of a B_Query_Specification). – **In: Locator** Is either NULL or a D_Locator containing **Result/** as a substring. – **Out: Skeleton** Is either NULL or a not empty set of values of type D_Locator, in which case it is to satisfy the following condition: For each object **X** of type E_KnowledgeItem existing in B_WissDB_Database, **X**.a_Loc is element of the **Skeleton** if and only if – **X**.a_Type is element of **SeeType**, and – The given **Locator** is NULL or is prefix of or identical to **X**.a_Loc. – **Out: AppErrors** Is to explain errors if the given **Locator** is neither NULL nor an element of the set B_WissDB_Database.E_Result.A_Loc. Is to contain warnings if at least one string found in **SeeType** is a value not allowed therein. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors – RC_seeWarnings
Has to:	Create the Skeleton specified above as a subset of D_Locator. Then, if this set is not empty, return RC_ok in **ReturnCode** and this set in **Skeleton**. If the set is empty, return NULL in **Skeleton** and again RC_ok in **ReturnCode**. Return NULL in **Skeleton** and RC_seeAppErrors **AppErrors** is explaining errors.

Because of:	C_WissDB_API needs a way to implement S_Search_for_Knowledge.

S_ Select_Processes

Component:	**S_Select_Processes** part of C_WissDB_DL_API
Abstract:	This is the function to ask the B_WissDB_Database for process skeletons. Parameters: – **In: Locator** Is to be either NULL or a string matching the pattern **D_Locator**/ **Process**/ **Name**. – **Out: Skeleton** Is either NULL or a not empty set of values of type D_Locator, in which case it is to satisfy the following condition: For each object **X** of type E_Process existing in B_WissDB_Database, **X**.a_Loc is element of the **Skeleton** if and only if – the given **Locator** is NULL or is prefix of or identical to **X**.a_Loc. – **Out: AppErrors** Is to explain errors if the given Locator is neither NULL nor an element of B_WissDB_Database.E_Process.A_Loc. Is to contain warnings if B_WissDB_Database.E_Process is empty. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors – RC_seeWarnings
Has to:	Create the Skeleton specified above as a subset of D_Locator. Then, if this set is not empty, return RC_ok in **ReturnCode** and this set in **Skeleton**. If the set is empty, return NULL in **Skeleton** and again RC_ok in **ReturnCode**. Return NULL in **Skeleton** and RC_seeAppErrors **AppErrors** is explaining errors.
Because of:	C_WissDB_API needs a way to implement S_Search_for_Knowledge.

S_ Exclude_Aspects

Component:	**S_Exclude_Aspects** part of C_WissDB_DL_API
Abstract:	This is the function to ask the B_WissDB_Database for certain aspect and/or result skeletons. Parameters: – **In: Locators** Is a set of values of type D_Locator. – **In: Aspects** Is a set of Aspect Locators (which are D_Locators prefixed by **Aspect**/). – **Out: Skeleton** Is the largest subset of the given set of **Locators** such the for each element **X** in **Skeleton** one of the following conditions holds: – **X** is an aspect locator currently in use, and some element of **Aspects** is preLocator of **X**. – **X** is a result locator currently in use, and the relation Database.R_Is_keyword_for.(A_keywordLoc, A_forLoc) contains a pair (**A**, **K**) such that **K** is a preLocator of **X**, and some element of **Aspects** is preLocator of **A**. – **Out: AppErrors** Is to contain a warning for each element of **Aspects** that is either not an aspect locator or a locator currently not in use. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors – RC_seeWarnings
Has to:	Create the **Skeleton** as explained above.
Because of:	C_WissDB_API needs a way to implement S_Search_for_Knowledge (and, especially, the effect of the Exclude section in a B_Query_Specification).

S_ Exclude_Processes

Component:	**S_Exclude_Processes** part of C_WissDB_DL_API
Abstract:	This is the function to ask the B_WissDB_Database to select certain process and/or result skeletons. Parameters: – **In: Locators** Is a set of values of type D_Locator. – **In: Processes** Is to be either NULL or a set of process locators (which are D_Locators containing **Process**/ as a substring). – **Out: Skeleton** Is the largest subset of the given set of **Locators** such the for each element **X** in **Skeleton** one of the following conditions holds: – **X** is a process locator currently in use, and some element of **Processes** is preLocator of **X**.a_Loc. – **X** is a result locator currently in use, and some element of **Processes** is preLocator of **X**.a_ofAct. – **Out: AppErrors** Is to contain a warning for each element of **Processes** that is either not a process locator or a locator currently not in use. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors – RC_seeWarnings
Has to:	Create the Skeleton as specified above.
Because of:	C_WissDB_API needs a way to implement in S_Search_for_Knowledge the effect of the Exclude section in a B_Query_Specification.

S_ Exclude_Results

Component:	**S_Exclude_Results** part of C_WissDB_DL_API
Abstract:	This is the function to ask the B_WissDB_Database for process skeletons. Parameters: – **In: Locators** Is a set of values of type D_Locator. – **In: Results** Is to be either NULL or a not empty set of result locators (which are D_Locators containing **Result**/ as a substring). – **Out: Skeleton** Is the largest subset of the given set of **Locators** such the for each element **X** in **Skeleton** there is an element **R** of **Results** such that **P** is preLocator of **R**. – **Out: AppErrors** Is to contain a warning for each element of **Results** that is either not a result locator or a locator currently not in use. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors – RC_seeWarnings
Has to:	Create the Skeleton as specified above.
Because of:	C_WissDB_API needs a way to implement S_Search_for_Knowledge (and, especially, the effect of the Exclude section in a B_Query_Specification).

S_ Describe_DB_DomainValues

Component:	**S_Describe_DB_DomainValues** part of C_WissDB_DL_API
Abstract:	This is the function to ask the B_WissDB_Database for a specification of the values allowed for domains that are enumeration types. Parameters: – **In: CSV_Dir** This is the B_URL to an existing folder in the file system. – **Out: CSV** This is the content of the file **CSV_Dir**/ **DomainValues.csv** describing in B_Domain_Update_Format all valid values of type – D_ItemType – D_PracticeType – D_ViewType – D_AbstractionType – D_UsageType – D_CorrelationType. – **Out: AppErrors** Is to describe an AppError is the **CSV_Dir** could not be found or if the file therein could not be written. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors
Has to:	Create or re-create the file **CSV_Dir**/ **DomainValues.csv**.
Because of:	Applications of C_WissDB_DL_API need a way to ask which values are currently allowed for enumeration domain types. Domain types must be allowed to be re-configured.

S_ Replace_or_create_DB_DomainValues

Component:	**S_Replace_or_create_DB_DomainValues** part of C_WissDB_DL_API
Abstract:	This is the function to ask the B_WissDB_Database for a specification of the values allowed for domains that are enumeration types. Parameters: – **In: Update** This is the B_URL to an existing ASCII file in either B_Domain_Update_Format or B_XML_Domain_Update_Format. – **Out: AppErrors** Is to describe an AppError if the **Update** file could not be found, could not be read, or did not contain values of type B_Domain_Update_Format or B_XML_Domain_Update_Format. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors
Has to:	Update the set B_WissDB_Database.E_DomainValues as described in **Update**. This is to be done as follows: ▪ Values described in the **Update** file in lines starting with a minus sign will be marked as being obsolete. ▪ Values described in the **Update** file in lines starting with a plus sign will be created using a number not in use so far (not even in use for obsolete values).
Because of:	Applications of C_WissDB_DL_API need a way to re-configure domain types that are enumerations.

S_ Set_Alias

Component:	**S_Set_Alias** part of C_WissDB_DL_API
Abstract:	This is the function by which you can define or change the value of an **Alias N** representing **a Project Locator that may or may not be currently in use**. Use S_Get_Alias in order to see the current value of **N**. Other services are to show standard locators (because these are the values of A_Loc attributes). Parameters: – **In: Alias** Is to be a positive integer. – **In: LocatorName** Is to be NULL or a D_Locator of type Project Locator that may or may not start with the alias of another project locator. – **Out: AppErrors** Is to describe an AppError if the given **LocatorName** does not have the form of a Project Locator, is starting with an alias that has value NULL, or is not NULL though the Alias is currently not prefix of a locator in use. – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror – RC_seeAppErrors
Has to:	The B_WissDB_Database has assigned to each positive number either NULL or a value of type D_Locator not starting with a number. From now on **LocatorName** is to be this value.
Because of:	Applications of C_WissDB_DL_API need a way to re-configure the value of a project locator alias.

S_ Get_Alias

Component:	**S_Get_Alias** part of C_WissDB_DL_API
Abstract:	This is the function allowing us to see the current value of a Project Locator Alias (such an alias being any positive integer). Parameters: – **In: Alias** Is a positive integer. – **Out: LocatorName** Is either NULL or a string of type D_Locator not starting with a number (in which case this string is a locator currently in use). – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror
Has to:	The B_WissDB_Database has assigned to each positive number either NULL or a value of type D_Locator not starting with a number. LocatorName is to be a copy of this value. Use S_Set_Alias if you want to change the LocatorName S_Get_Alias is to return.
Because of:	Applications of C_WissDB_DL_API need a way to see the current value of a project locator alias.

S_ Testsupport_New_DB

Component:	**S_Testsupport_New_DB** part of C_WissDB_DL_API
Abstract:	This function is test support. Parameters: – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror
Has to:	Empty all tables – E_DomainValues – E_Process – E_Role – E_KnowledgeItem – E_Aspect.
Because of:	Test drivers need a way to give the B_WissDB_Database a well known initial value.

S_ Testsupport_DL_API_Reaction_Scope

Component:	**S_Testsupport_DL_API_Reaction_Scope** part of C_WissDB_DL_API
Abstract:	This function is test support. Parameters: – **Out: Scope** This is the sequence of all strings that could occur as **S_x**.**ReturnCode** or – in AppError descriptions – as **S_x**.**ErrorType**. Here **S_x** is to cover all services that are part of the DL_API API (i.e. all services described in this document). – **Out: ReturnCode** is one of the following values: – RC_ok – RC_syserror
Has to:	Produce and return the **Scope** value.
Because of:	Test drivers need a way to quantify test coverage with respect to the so-called System Reaction Metric: Given a test suite for C_WissDB_DL_API, test coverage is said to be N% if and only if running the complete test reproduced N% of all values found in **Scope**.