SIS Thesaurus Management System

Update Application Programmatic Interface Reference Manual

Version 1.1

Institute of Computer Science

Foundation for Research and Technology - Hellas

TABLE OF CONTENTS

1. Introduction

1.1 SIS-TMS Technical Characteristics

The SIS Thesaurus Management System (SIS-TMS) consists of a tool to develop multilingual thesauri and a terminology server for cataloguers and for distributed access to heterogeneous electronic collections. The distinct features of the TMS are its capability to store, develop and access multiple thesauri and their interrelations under one database schema, to create any relevant view thereon and to specialize dynamically any kind of relation into new ones.

The SIS-TMS server can be integrated in a distributed, heterogeneous environment. As a central, eventually repeated component, it can replace the cumbersome implementation and population of thesaurus management features in collection databases and library systems, due to access through its programmatic interface. It further allows automatic term expansion and translation in distributed access environment. This use requires consistency of the equivalence relations established between thesauri. The means of consistency control provided by SIS-TMS is a unique feature.

The SIS-TMS system is an application of the Semantic Index System, a general-purpose object-oriented semantic network database, product of ICS-FORTH. Its schema is based on the principles of the ISO2788 and ISO5964 standards for the establishment and documentation of monolingual and multilingual thesauri. It is outcome of international co-operation with cultural organizations.

1.2 SIS-TMS Programmatic Interface Technical Characteristics

The SIS-TMS Application Programmatic Interface (hereafter TMS-API), designed and implemented by ICS-FORTH, offers a complete set of update operators, which implement frequently used combinations of primitive operations in respect to the SIS-TMS schema knowledge.

This report presents the set of functions the TMS-API consists of and the functionality of the supported functions. Note that these functions can be used to update the SIS-TMS base (thesaurus database). To retrieve information from the thesaurus database, the programmer should use the SIS-API functions, which are a complete set of query operators for accessing the SIS-TMS base. For more details on the SIS-API see *"SIS - Application Programmatic Interface Reference Manual"*.

SIS-TMS API libraries are available in PC versions in C++ and C (Borland 5.01 libraries and dll) and in Java. In this document we present the C version of the SIS-TMS programmatic interface. The differences between C interface and the C++ and Java interfaces are presented in "Appendix C - C++ Programmatic Interface" and "Appendix D – Java Programmatic Interface".

1.3 SIS-TMS client-server model

The SIS-TMS is based on a client-server model. An application may use TMS-API for querying and/or modifying the SIS-TMS base, as a client. A second process (the SIS server) has to run in parallel to the application and this process gives the answers to any question from the application, and modifies the data in the SIS-TMS base. The server is responsible for reading and writing the SIS-TMS base files.

The main advantage of this model is that the client (or clients) need not run on the same machine the SIS server is running. The communication between the client process and the server process is achieved through sockets.

1.4 SIS sessions and TMS sessions

TMS-API functions can be used to update the SIS-TMS base (thesaurus database); to retrieve information from the thesaurus database, the programmer should use the SIS-API functions.

In order to provide real multi-threading to the clients that were using the TMS-API, we introduced the notion of sessions as we did to the SIS-API. The application developer that needs to provide multi-thread access to different servers or the same server (making simultaneous queries or updates) should create multiple sessions to implement this.

2 The programmatic scenario

An application that uses the TMS-API to update the SIS-TMS database should, in general terms, do the following:

1. Determine the SIS-TMS database and create a sis-session to connect with the SIS database with **create_SIS_CS_Session**() function.
2. Establish the connection with the SIS-TMS database with **open_connection**() function.
3. Create a tms-session with **create_TMS_API_Session**() function associated with the sis-session created in step 1 and set the name of the thesaurus to work with, with **SetThesaurusName**() function.
4. Start a query or a transaction session to access or modify the SIS data with **begin_query**() and **begin_transaction**() functions.
5. Set a current node with the **set_current_node**(*node_name*) function.
6. Use a set of query functions (described in *"SIS - Application Programmatic Interface Reference Manual"*), or update functions (described below), which retrieve information and collect the answer into temporary sets at the server-site or modify information in respect to the SIS-TMS schema.
7. Repeat steps 5 and 6 and optionally performs some set operations on the temporary sets.
8. Terminate the query or the transaction session with **end_query**() or **end_transaction**() functions.
9. Terminate connection with the server with the **close_connection**() function.
10. Release the tms-session and sis-session with **release_TMS_API_Session**() and **release_SIS_Session**() functions.

3 Thesaurus Management Schema Considerations

3.1 Thesaurus Structures

The following sections are describing the SIS Thesaurus Management System modeling concepts as presented in "*SIS - TMS: A Thesaurus Management System for Distributed Digital Collections*", published in the Proceedings of the. 2nd European Conference, ECDL'98 (September 1998) [9].

3.1.1 Assumptions on Concepts

According to [1] one of the major purposes of a thesaurus is to "provide a map of a given field of knowledge, indicating how *concepts* or ideas about concepts are related to one another, which helps an indexer or a searcher to understand the structure of the field.

We distinguish concepts from terms, in contrast to IS2788. Cognitive scientists have proposed several definitions for the notion of "concept" (e.g. [2]). According to one point of view, a concept is perceived as a set of entities, called "concept instances" characterized as such by common agreement rather than formal reasoning on the properties that characterize an individual entity as an instance of a concept. We adopt this view for thesauri, considering a concept as a notion by which some people agree to refer in a well defined manner to a set of real world objects with the same properties, without necessarily defining properties. Consequently, certain semantic relations between concepts are interpreted as relations between sets as will be presented below. For more details see [3], [4].

Following ISO2788, we regard terms as nouns or noun-phrases, by which groups of people use to refer to certain concepts in a certain context. Due to varying groups and contexts, concepts and terms are related many to many.

3.1.2 Modelling Thesaurus Notions

The SIS-TMS schema is extensible at run-time. New semantic relations can be created or existing ones can be specialized. The current conceptual model of the SIS-TMS for the representation of multiple interlinked thesauri incorporates the thesaurus notions and intrathesaurus relations of the ISO2788 for monolingual thesauri and an extended version of the ISO5964 interthesaurus relations [4]. In prototype versions, this schema has been extended for the ULAN and TGN, vocabularies of the Getty Information Institute and the Library of Congress Subject Headings. We mainly use ISO2788 terminology for the names of the classes and relations in the SIS-TMS schema. In the manner of semantic networks, these names are directly presented in the user interface together with the respective data and read quite naturally.

We model *Preferred Terms* for indexing and *Non-Preferred Terms* as synonyms and entry points for the user. In addition, *Non-Preferred Terms* may be used for full-text retrieval. We adopt the notion of *Descriptor* of the Art & Architecture Thesaurus [5] according to which: "a descriptor is the term that uniquely identifies the concept". Hence a *Descriptor* is a term and a concept identifier in double nature. All other

terms, preferred or not, are related to the concept and not further described, as we are not interested in linguistics.

As the *concept* is identified by a *descriptor*, i.e. by a linguistic expression that best expresses the common understanding of experts or public and it must be unique within the context in which it has been defined, it may not be exactly the word an expert uses. For instance, "pink (color)" and "pink (vessel)" would be good descriptors, but experts would say "pink" in both cases. In the SIS-TMS, all terms and descriptor names are enforced to be unique throughout the database. Terms may be multiply related to different concepts (*Descriptors*), but if a good term appears to conflict with a descriptor, the descriptor has to be renamed, i.e. usually extended for disambiguation.

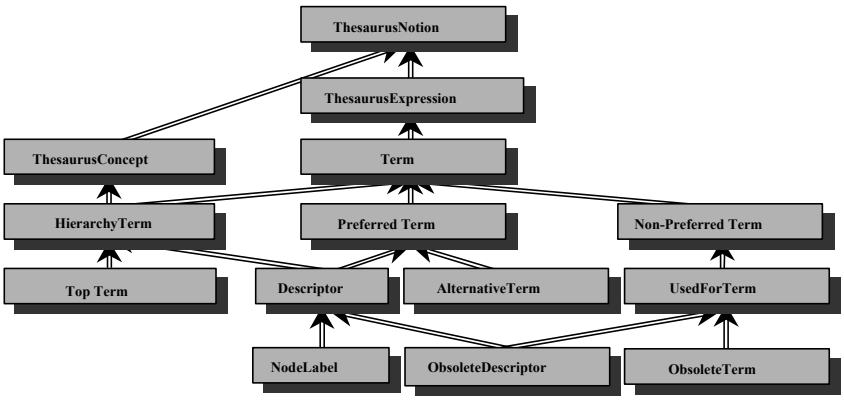


Figure1. IsA hierarchy of the SIS-TMS C lasses of thesaurus notions

Concepts carry all the intra and interthesaurus relations that make up the semantic structure of the thesaurus contents, they carry the administrational information, and they can be described by scope notes and understood language independently.

Figure 1 shows the isA hierarchy of the *SIS-TMS* Classes of thesaurus notions. We use in the following "abstract" for classes which are not directly instantiated, and "abstract hook" for abstract classes, which are designed to be superclasses of classes in future extensions. «ThesaurusNotion» is the abstract root. «ThesaurusExpression» is the abstract hook for terms, person names, date expressions etc. «ThesaurusConcept» is the abstract hook for concepts in the above sense, persons, places etc. «HierarchyTerm» is the class for concept in the above sense, those that can be generalized or specialized into broader/narrower meaning. It combines *Node Labels*, or "guide terms" and descriptors. We do not distinguish functionally between both (see e.g. [6]). «AlternativeTerm» is the complement of «Descriptor». «Topterm» are those having no broader terms. «ObsoleteDescriptor» are abandoned concepts (sometimes thesauri decrease, e.g. in favor of dynamic concept formation) and finally «ObsoleteTerm» are deleted noun-phrases. The latter two serve version management for referential integrity incremental update.

3.1.3 Intrathesaurus Relations

The semantic relations in a thesaurus can be divided into intrathesaurus relations within a coherent terminological system and interthesaurus relations between independent terminological systems. They are used to represent relationships between concepts and between concepts and terms, i.e. from the class HierarchyTerm to the class HierarchyTerm or Term.

The intrathesaurus relations identified by ISO2788 are: the *hierarchical relationships,* distinguishing a systematic thesaurus from an unstructured list of terms (glossary or dictionary), associating concepts bearing broader/narrower meanings, identified by the *BT (broader term)* relation, the *associative relationships*, relating concepts that are not members of an equivalence set nor can they be organized in a hierarchy, identified by the *RT (related term)* relation, and finally the *equivalence relationship* established between preferred and non-preferred terms, considered to refer to the same concept, and it is identified by the *use* and its inverse *UF* (used for) relations. As it does not distinguish between terms and concept, and we reinterpret these relations as the link between the conceptual and linguistic level. We refer in the following their functional role and specializations.

BT is used for semantic generalization or specialization of query terms. From a knowledge representation approach, the *BT* relation carries isA semantics, and a query term may be expanded by its narrower terms, if we ask for objects of this kind. Consequently SIS-TMS enforces that all HierarchyTerms have a broader Term except for TopTerms, and that the BT relation is acyclic. A Term may have multiple broader terms in the sense of multiple supersets. Thesaurus maintainers may distinguish between the main and alternate broader terms.

RT is used for the detection of relevant concepts by users. It plays a role like a general attribute category in KR systems. Dozens of useful specializations can be found, as the "subdivisions" of ISO2709, whole-part relations, and rule-related relations. In the latter case, machine interpretation may occur. *Art & Architecture Thesaurus* team has identified more that 20 different meanings of the RT relation.

UF (use for) can be used by users as entry points in a thesaurus. Actually most thesauri distinguish the *ALT* relation to preferred terms from the *UF* to non-preferred terms. The EET (European Education Thesaurus) [7] consequently regards any translation of a concept to some language as a kind of UF.

In SIS-TMS, *hierarchical association, equivalence association* and *associative relation* are modeled as metacategories of intrathesaurus links. These generic categories group and control the specialization of relations to preserve compatibility and to maintain the related global consistency rules. I.e. the application code can refer to those for constraint enforcement and for export of data in a compatible format. Hence the application code is robust against extensions. User defined extensions as the above mentioned specializations of *BT*, *RT*, and *UF* links are substantial for specific applications and the maintenance of their logical consistency, as well as for the conceptual evolution of thesaurus structures into knowledge bases despite format standardization.

Other systems allow for new user defined links, but do not relate them to existing semantics and hence do not automatically imply standard conformant viewing,

handling and constraint enforcement. Specialization of relations is a distinct feature of the SIS-TMS.

3.1.4 Representing Multiple Interlinked Thesauri

A challenge in the SIS-TMS was the development of the conceptual model to incorporate multiple interlinked thesauri under one database schema. For the development of the currently implemented conceptual schema two approaches were studied. The problems is, how to distinguish between the work of each thesaurus editor on one side, on the other side to see the common things as common without running continuously search routines, and on the third side to get a global view on how the different thesauri fit together. For more details see [8].

According to the second, a global name space is made up for all terms and concepts of one language. For each thesaurus, a separate schema is generated. Due to multiple instantiation in SIS, these schemata can overlap conflict-free on the data. The system tables stay limited per thesaurus. Gradual merge is possible without duplicating the records, as the same term can participate in any thesaurus. (Terms are not regarded as the invention of the thesaurus editor). Each thesaurus may have different semantic structures. Links are not confused, as they belong to individual schemata. Terms of different languages are distinguished by prefixes. Concepts of an interlingua can be dealt likewise.

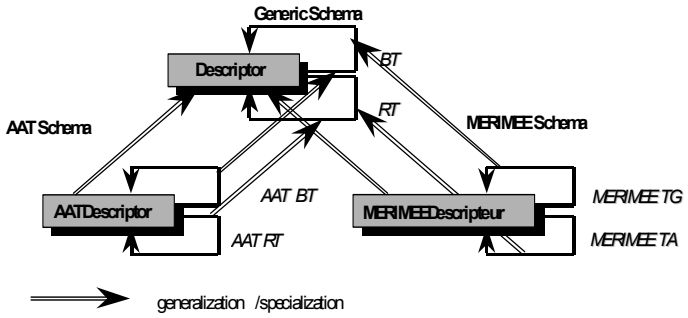


Figure 2. Subschema Creation per Thesaurus

On top of this world, one generic schema is developed as superclasses to provide global views. "Supercategories", abstractions of links and attributes from the individual schema as presented in the previous chapter, provide the notion of global semantics. For each new thesaurus, the generic schema is duplicated by specialization of its generic classes and relations into thesaurus specific ones, and the thesaurus data are loaded under these (figure 2). For deviations in semantics, appropriate extensions can be made. This model results in a large schema. As an SIS schema is declarative, and not by storage allocation, no space is wasted. We selected this novel approach.

As a consequence, we may see on one common descriptor all links made by other thesauri, achieving a kind of trivial merge. Each schema provides an isolated view of one thesaurus, and the generic schema a unified view of all. Basically, each thesaurus is handled as an annotation, an *opinion* of a group on a common term and concept space. This method can be elaborated into much more sophistication [8].

3.1.5 Interthesaurus Relations

The interthesaurus relations modeled in the SIS-TMS are an extended version of the ISO5964 links as presented in [4]. Those refined relations are the outcome of a discussion that took place in the framework of the AQUARELLE project between experts in multilingual thesaurus creation and ICS-FORTH, the provider of the multilingual thesaurus management component used in the project. The specific problem was to embed a system of multiple independent thesauri in different languages into a system for access to heterogeneous databases containing objects of material culture, supporting automatic term expansion/translation under a Z39.50 protocol. The conclusion of this discussion was that ISO5964 does not define precise enough semantics for that purpose.

We define the following relations: *exact equivalence, broader equivalence, narrower equivalence, inexact equivalence, union* and *intersection* of concepts. A detailed presentation of the semantics of above relations are presented in [4]. These links are from concept to concept (HierarchyTerm), and should not confused with linguistic translations, which use any suitable word from the other language rather than the specific thesaurus descriptors.

Obviously equivalence relations are opinions of one group, or at least under the responsibility of one group. Of course, good teams seek advice from each other. But the geographical distance and other local needs hinder synchronous updates. We therefore foresee different equivalence relations for group A from Thesaurus A to B, than for group B from B to A. If group A or B withdraws a concept, it remains marked as obsolete in the database, giving the other group a chance to redirect their links later. New concepts are marked as new, and should not be referred to until released. Suitable permissions can be set up in the SIS-TMS, so that such a database can be maintained cooperatively without conflicts through the net.

3.2 Version Control and Data Consistency

The purpose of the version control is the information of thesaurus editors about previous discussions and states, and the capability to incrementally update another Collection Management System or term server with the changes done in the TMS. Thesaurus releases are created at a slow rate, months or years. A rollback features is therefore not necessary, backups are sufficient for that purpose. Individual changes can be withdrawn at any time. A function is however provided, which inserts the latest changes into the last release for «last minute changes».

Consequently, the idea of the SIS-TMS implementation is to keep in the database only the least versioning information for the above purposes as *backward differences* for scalability reasons. All other version data may be put in history logs in future versions. Versioning is based on releases rather than dates. The "current" release is being edited, and no history of changes is kept within it. Rather, the results of individual changes are merged. In contrary to version control systems, always the current version is displayed together with all registered backward changes per entity. The latter can be filtered out. Under this perspective, we register whether

- a descriptor has been introduced (a new concept is described)

- an existing descriptor has been abandoned (the concept is regarded inappropriate for classification or should be composed dynamically from other concepts)
- an existing descriptor is renamed
- any semantic information around a descriptor has changed.

We distinguish between operations on released concepts and on unreleased. In the unreleased, the user can introduce descriptors, which are classified as "*new descriptor"*. He/she can perform all operations on descriptors and undo them.

The operations on released concepts are constraint, because they contain data that have been communicated to other systems and may have been used for indexing. Introduction of descriptors as well as deletions is not permitted. A descriptor can be abandoned by the following procedure: It is classified as *"obsolete descriptor"* and its *broader/narrower* associations to the others are deleted but it remains a member of the term list of its hierarchy, retaining the context in which it has been defined. Further, the «gap» in the hierarchy is «closed» by drawing *BT* relations between the narrower terms of the *obsolete* descriptor and its broader terms. We do not constrain any changes in the associated information, semantic and administrational links and attributes, as this is not necessary to keep other systems up-to-date or to avoid dangling references.

If editors regard the noun phrase, which identifies a descriptor, as inappropriate for the semantics of the concept, or it is going to cause name conflicts, it can be renamed. Renaming an object in SIS does not alter its system identifier and its properties remain attached to it. SIS-TMS maintains uniqueness of all term-names in the system. Whereas ALT and UF terms can be deleted at any time. Released descriptor names, which come out of use, become «obsolete terms». Further, for each release, links are maintained where the names have gone to. The complete algorithm is not trivial, as within one release all rename actions must be merged in order to be unique, and obsolete names may be reused as ALT or UF terms. Currently, following the AAT philosophy, we do not allow a concept (descriptor) to refer another descriptor as UF, but we expect in that case, that the respective descriptor is renamed to disambiguate it from the other concept.

4 The functionality of the Update Application Programmatic Interface functions

4.1 General Description

The TMS-API provides a set of functions used to make updates in the database in respect to the SIS-TMS schema. The operations that can be performed are:

- General operations
- Addition operations
- Renaming operations
- Undo operations
- Abandoning operations
- Deletion operations
- Comment handling operations

The relation between the above groups of operations is guided by the following principles:

- For each allowed operation there is an inverse function to undo its effect. So any series of changes can be taken back in inverse order. This holds as long as no new version (release) is issued.
- There is no registration of undone changes.
- At the issuing of a new release, the effective changes on concepts are registered.
- In the sequence, taking them back, is a change in itself, an "abandon" operation.

The operations on released concepts are constraint, because they contain data that have been communicated to other systems and may have been used for indexing. Deletion of descriptors is not permitted. A descriptor can be abandoned: classified as *"obsolete descriptor"* and its *broader/narrower* associations to the others are deleted but it remains a member of the term list of its hierarchy, retaining the context in which it has been defined. Renaming is also constrained to released concepts: the system keeps the renaming history of a released concept. The operations on released concepts can again be undone with other special operations as long as no release is issued.

In order to execute updates on a database a transaction session must be initiated. A transaction session can either be initiated directly or from within a query session. When a transaction session is in progress a write lock is applied to the database and only the writing client can access it. A transaction session begins with the **begin_transaction**() function and ends with either the **end_transaction**() function (commits changes) or with the **abort_transaction**() function (does not commit changes). When the transaction session is initiated from within a query session, on termination of the transaction session the query session continues, and thus a read lock exists on the database. In order to release the read lock end the query session by calling **end_query**().

The functions presented below return `TMS_APIFail(-1)` on error and `TMS_APISucc(0)` on success.

NOTE: *In the following sections all functions described take as first argument the TMSsessionID of the session (access point to the database) that they are quering or updating.*

4.2 General Operations

4.2.1 Create a session

Operation : **int create_TMS_API_Session**(int* TMSsessionID, int SISsessionID)

Input : TMSsessionID, SISsessionID

This operation creates a session for operating on a thesaurus database, designated by the *SISsessionsID* argument. It associates the tms-session created with a sis-session. The operation fails in the following case:

- Given sis-session is not valid.

Example:

```
int sis_session, tms_session;
create_SIS_Session(&sis_session, .....);
```

```
create_TMS_API_Session(&tms_session, sis_session);
```

4.2.2 Release a session

Operation : **int release_TMS_API_session**(int TMSsessionID)

Input : TMSsessionID

This operation releases a session designated by the *TMSsessionID* argument. The operation fails in the following case:

- Given tms-session is not valid.

Example:
```
release_TMS_API_Session(tms_session);
```

4.2.3 Get the sis-session associated with a tms-session

Operation : **void get_associated_SIS_Session**(int TMSsessionID, int* SISsessionID)

Input : TMSsessionID, SISsessionID

This operation returns the sis-session ID (in argument *SISsessionID*), which is associated with the specified tms-session (*TMSsessionID*). The operation fails in the following case:

- Given tms-session is not valid and returns –1 in *SISsessionID.*

Example:
```
get_associated_SIS_Session(tms_session, &sis_session);
```

4.2.4 Set a thesaurus name

Operation : **int SetThesaurusName**(int TMSsessionID, char *ThesaurusName)

Input : TMSsessionID, ThesaurusName

This operation sets the name of the thesaurus, which TMS-API will work with. The operation fails in the following case:

- Given thesaurus name is not the name of any existing thesaurus.

Example:
```
SetThesaurusName("MERIMEE");
```

4.2.5 Get current thesaurus name

Operation : **void GetThesaurusName** (int TMSsessionID, char ThesaurusName)

Input : TMSsessionID, ThesaurusName

This operation returns in *ThesaurusName* the current name of the thesaurus, which TMS-API works with.

Example:
```
l_name thesaurusName;
GetThesaurusName(tms_session, thesaurusName);
```

4.2.6 Get TMS-API error message

Operation : **char *GetTMS_APIErrorMessage**(int TMSsessionID)

Input : TMSsessionID

This operation returns the reason of failure of last unsuccessful TMS-API function call.

Example:
```
fprintf(stderr, "%s", GetTMS_APIErrorMessage(tms_session));
```

4.3 Addition Operations

4.3.1 Create a new facet

Operation : **int CreateFacet**(int TMSsessionID, char *FacetName)

Input : TMSsessionID, FacetName

This operation creates a new facet. The operation fails in the following cases:

- A facet with the same name already exists.
- Input *FacetName* does not contain the correct prefix for a facet of currently used thesaurus.

Example:
```
CreateFacet(tms_session, "MERIMEEClass`test_api_facet");
```

4.3.2 Create a facet attribute

Operation : **int CreateFacetAttribute**(int TMSsessionID, char *linkName, char *facetName, cm_value *toValue, int catSet)

Input : TMSsessionID, LinkName, facetName, toValue, catSet

This operation creates a new attribute for a facet. *linkName* is the name of the new attribute (‘\0’ for unnamed), *facetName* is the from-value of the attribute, *toValue* its to-value and *catSet* the set with the categories of the attribute. If *catSet* is –1 then no categories are used. The operation fails in the following cases:

- *facetName* is not the name of any existing facet of currently used thesaurus.
- The given *catSet* contains categories which are not allowed to be used for the creation of a facet attribute. Available categories for the creation of a facet attribute:

Category from-class	*Category name*
Facet	letter_code

Example:
```
cm_value toValue;
assign_string(&toValue, ”letter_code_to_value");
int catSet;
catSet = set_get_new();
reset_name_scope();
set_current_node(“Facet”);
set_current_node(“letter_code”);
set_put(catSet);
CreateFacetAttribute(tms_session,  "my_letter_code","MERIMEEClass`
test_api_facet", &toValue, catSet);
free(toValue.value.s);
free_set(catSet);
```

4.3.3 Create a new hierarchy

Operation : **int CreateHierarchy**(int TMSsessionID, char *HierarchyName, char *FacetName)

Input : TMSsessionID, HierarchyName, FacetName

The hierarchy is added in the knowledge base and classified in the specified facet. The top term of the hierarchy is created and appropriately associated with it. The operation fails in the following cases:

- A hierarchy with the same name already exists.
- *FacetName* is not the name of any existing facet of currently used thesaurus.
- Input *HierarchyName* does not contain the correct prefix for a hierarchy of currently used thesaurus.

Example:

```
CreateHierarchy(tms_session,  "MERIMEEClass`test_api_hier",
"MERIMEEClass`test_api_facet");
```

4.3.4 Create a hierarchy attribute

Operation : **int CreateHierarchyAttribute**(int TMSsessionID, char *linkName, char *hierarchyName, cm_value *toValue, int catSet)

Input : TMSsessionID, linkName, hierarchyName, toValue, catSet

This operation creates a new attribute for a hierarchy. *linkName* is the name of the new attribute ('\0' or *NULL* for unnamed), *hierarchyName* is the from-value of the attribute, *toValue* its to-value and *catSet* the set with the categories of the attribute. If *catSet* is –1 then no categories are used. The operation fails in the following cases:

- *hierarchyName* is not the name of any existing hierarchy of currently used thesaurus.
- The given *catSet* contains categories which are not allowed to be used for the creation of a hierarchy attribute. Available categories for the creation of a hierarchy attribute:

Category from-class	*Category name*
Facet	letter_code

Example:

```
cm_value toValue;
assign_string(&toValue, "letter_code_to_value");
int catSet;
catSet = set_get_new();
reset_name_scope();
set_current_node("Facet");
set_current_node("letter_code");
set_put(catSet);
CreateHierarchyAttribute(tms_session,  "my_letter_code",
"MERIMEEClass` test_api_hier", &toValue, catSet);
free(toValue.value.s);
free_set(catSet);
```

4.3.5 Create a new concept

Operation : **int CreateDescriptor**(int TMSsessionID, char * DescriptorName, char * BroaderTerm)

Input : TMSsessionID, DescriptorName, BroaderTerm

This operation creates a new descriptor. The descriptor is added in the knowledge base and is associated with the given broader term with a BT relation. It is also classified in its broader term hierarchies. The operation fails in the following cases:

- A descriptor with the given name already exists
- The given broader term does not exist.
- The given broader term is not a descriptor.
- Input *DescriptorName* does not contain the correct prefix for a descriptor of currently used thesaurus.

Example:

```
CreateDescriptor(tms_session,   "TermeFr`test_descriptor",  "TermeFr`
ALLEE");
```

4.3.6 Associate a new concept with terms from same thesauri

Operation : **int CreateNewDescriptorAttribute** (int TMSsessionID, char *linkName, char * descriptorName, cm_value *toValue, int catSet)

Input : TMSsessionID, linkName, descriptorName, toValue, catSet

This operation creates a new attribute for a new concept. *linkName* is the name of the new attribute (‘\0’ or *NULL* for unnamed), *descriptorName* is the from-value of the attribute, *toValue* its to-value and *catSet* the set with the categories of the attribute. If *catSet* is –1 then no categories are used. The operation fails in the following cases:

- *descriptorName* is not the name of any existing new concept of currently used thesaurus.
- The given *catSet* contains categories which are not allowed to be used for the creation of a new concept attribute. Available categories for the creation of a new concept attribute (<thes_nameU> and <thes_nameL> are the upper and lower case names of the currently selected thesaurus, for example: *MERIMEE, merimee*):

Category from-class	*Category name*
<thes_nameU>HierarchyTerm	<thes_nameU>_ALT
<thes_nameU>HierarchyTerm	<thes_nameL>_display
<thes_nameU>HierarchyTerm	<thes_nameL>_editor
<thes_nameU>HierarchyTerm	<thes_nameL>_found_in
<thes_nameU>HierarchyTerm	<thes_nameL>_modified
<thes_nameU>HierarchyTerm	<thes_nameL>_found_in
<thes_nameU>HierarchyTerm	<thes_nameU>_RT
<thes_nameU>HierarchyTerm	<thes_nameU>_UF
<thes_nameU>ThesaurusConcept	<thes_nameU>_translation, to_EN
<thes_nameU>ThesaurusConcept	<thes_nameU>_translation, to_GR
<thes_nameU>ThesaurusConcept	<thes_nameU>_translation, to_IT
<thes_nameU>HierarchyTerm	<thes_nameL>_created

Example:

```
cm_value toValue;
assign_string(&toValue, "TermeFr`CELLIER");
int catSet;
catSet = set_get_new();
reset_name_scope();
set_current_node("MERIMEEHierarchyTerm");
set_current_node("MERIMEE_ALT");
set_put(catSet);
CreateNewDescriptorAttribute (tms_session,  "myALT",
"TermeFr`test_descriptor", &toValue, catSet);
free(toValue.value.s);
free_set(catSet);
```

4.3.7 Associate a released concept with terms from same thesauri

Operation : **int CreateDescriptorAttribute** (int TMSsessionID, char *linkName, char * descriptorName, cm_value *toValue, int catSet)

Input : TMSsessionID, linkName, descriptorName, toValue, catSet

This operation creates a new attribute for a released concept. *linkName* is the name of the new attribute ('\0' or *NULL* for unnamed), *descriptorName* is the from-value of the attribute, *toValue* its to-value and *catSet* the set with the categories of the attribute. If *catSet* is –1 then no categories are used. The operation fails in the following cases:

- *descriptorName* is not the name of any existing released concept of currently used thesaurus.
- The given *catSet* contains categories which are not allowed to be used for the creation of a new concept attribute. Available categories for the creation of a new concept attribute (<thes_nameU> and <thes_nameL> are the upper and lower case names of the currently selected thesaurus, for example: *MERIMEE, merimee*):

Category from-class	*Category name*
<thes_ nameU>HierarchyTerm	<thes_ nameU>_ALT
<thes_ nameU>HierarchyTerm	<thes_nameL>_display
<thes_ nameU>HierarchyTerm	<thes_nameL>_editor
<thes_ nameU>HierarchyTerm	<thes_nameL>_found_in
<thes_ nameU>HierarchyTerm	<thes_nameL>_modified
<thes_ nameU>HierarchyTerm	<thes_nameL>_found_in
<thes_ nameU>HierarchyTerm	<thes_ nameU>_RT
<thes_ nameU>HierarchyTerm	<thes_ nameU>_UF
<thes_ nameU>ThesaurusConcept	<thes_ nameU>_translation, to_EN
<thes_ nameU>ThesaurusConcept	<thes_ nameU>_translation, to_GR
<thes_ nameU>ThesaurusConcept	<thes_ nameU>_translation, to_IT

Example:

```
cm_value toValue;
assign_string(&toValue, "TermeFr`CELLIER");
int catSet;
catSet = set_get_new();
reset_name_scope();
set_current_node("MERIMEEHierarchyTerm");
set_current_node("MERIMEE_ALT");
set_put(catSet);
```

```
CreateDescriptorAttribute(tms_session,  "myALT", "TermeFr`ABREUVOIR",
&toValue, catSet);
free(toValue.value.s);
free_set(catSet);
```

4.3.8 Associate a concept with terms from other thesauri

Operation : **int CreateInterThesRelation**(int TMSsessionID, char * FromTerm, char * Category, char * ToTerm)

Input : TMSsessionID, FromTerm, Category, ToTerm

This operation adds inter-thesaurus links to a descriptor. An inter-thesaurus link of type *Category* is created to associate the *FromTerm* with the *ToTerm*. The *ToTerm* can also be a collective concept: it can be a concept to express a union or an intersection of concepts of the target-thesaurus. The appropriate broader term links are constructed so as to associate the collective concept with its component concepts using the following syntax:

concept_name1 + *concept_name2* (for union)
concept_name1 & concept_name2 (for intersection)

The operation fails in the following cases:

◊ In case the *ToTerm* is not a collective concept
- The *ToTerm* does not exist. In this case, the operation fails because it is not legal to introduce new concepts in the target-thesaurus.
- The *ToTerm* is not a descriptor.

◊ In case the *ToTerm* is a collective concept.
- The components of the *ToTerm* do not exist in the target-thesaurus. In this case the operation fails because it is not legal to introduce new concepts in the target-thesaurus.
- The components of the *ToTerm* are not descriptors.

Example:

```
CreateInterThesRelation(tms_session,  "TermeFr`BERGERIE",
"MERIMEE_exact_equivalence , to_RCHME", "EnTerm`garden & lake");
```

4.3.9 Create a new alternative term

Operation : **int CreateAlternativeTerm** (int TMSsessionID, char * term)

Input : TMSsessionID, term

This operation creates a new alternative term. The term is added in the knowledge base and is instantiated under the *AlternativeTerm* class of the current thesaurus (for example *MERIMEEAlternativeTerm*). The operation fails in the following cases:
- A term with the given name already exists
- Input *term* does not contain the correct prefix for a term of currently used thesaurus.

Example:

```
CreateAlternativeTerm (tms_session,  "TermeFr`myAlternativeTerm");
```

4.3.10 Create a new used for term

Operation : **int CreateUsedForTerm** (int TMSsessionID, char * term)

Input : TMSsessionID, term

This operation creates a new used for term. The term is added in the knowledge base and is instantiated under the *UsedForTerm* class of the current thesaurus (for example *MERIMEE UsedForTerm*). The operation fails in the following cases:

- A term with the given name already exists
- Input *term* does not contain the correct prefix for a term of currently used thesaurus.

Example:

```
CreateUsedForTerm (tms_session,  "TermeFr`myUsedForTerm");
```

4.3.11 Create a new editor

Operation : **int CreateEditor** (int TMSsessionID, char * editor)

Input : TMSsessionID, editor

This operation creates a new editor. The editor is added in the knowledge base and is instantiated under the *Editor* class of the current thesaurus (for example *MERIMEE Editor*). The operation fails in the following cases:

- An editor with the given name already exists
- Input *editor* does not contain the correct prefix for an editor of currently used thesaurus (Person`).

Example:

```
CreateEditor(tms_session,  "Person`myEditor");
```

4.3.12 Create a new source

Operation : **int CreateSource**(int TMSsessionID, char * source)

Input : TMSsessionID, source

This operation creates a new source. The source is added in the knowledge base and is instantiated under the *Source* class. The operation fails in the following cases:

- A source with the given name already exists
- Input *source* does not contain the correct prefix for a source (Literature`).

Example:

```
CreateSource(tms_session,  "Literature`mySource");
```

4.3.13 Create a new word

Operation : **int CreateAmericanWord** (int TMSsessionID, char * word)
int CreateDanishWord(int TMSsessionID, char * word)
int CreateCatalanWord(int TMSsessionID, char * word)
int CreateSpanishWord(int TMSsessionID, char * word)
int CreatePortugueseWord(int TMSsessionID, char * word)
int CreateGermanWord(int TMSsessionID, char * word)
int CreateGreekWord(int TMSsessionID, char * word)
int CreateItalianWord(int TMSsessionID, char * word)

int CreateFrenchWord(int TMSsessionID, char * word)
int CreateEnglishWord(int TMSsessionID, char * word)

Input : TMSsessionID, word

These operations create a new word. The word is added in the knowledge base and is instantiated under the *AmericanWord, DanishWord, CatalanWord, SpanishWord, PortugueseWord, GermanWord, GreekWord, ItalianWord, FrenchWord, or EnglishWord* class. The operation fails in the following case:

- A word with the given name already exists

Example:

```
CreateGreekWord(tms_session,  "myGreekWord");
```

4.4 Classification Operations

4.4.1 Classify a new descriptor

Operation : **int ClassifyNewDescriptor**(int TMSsessionID, char *descriptorName, char *className)

Input : TMSsessionID, descriptorName, className

This operation classifies a new descriptor. The descriptor is classified under the given *className* class. The operation fails in the following cases:

- *descriptorName* is not the name of any existing new descriptor of currently used thesaurus.
- *className* is not one of the allowed classes for new descriptors classification. Available classes for the classification of a new descriptor (<thes_nameU> is the upper case name of the currently selected thesaurus, for example: *MERIMEE*):

  1. <thes_nameU>GuideTerm

Example:

```
ClassifyNewDescriptor(tms_session,  "TermeFr`test_desciptor",
"MERIMEEGuideTerm");
```

4.4.2 Declassify a new descriptor

Operation : **int DeClassifyNewDescriptor**(int TMSsessionID, char *descriptorName, char *className)

Input : TMSsessionID, descriptorName, className

This operation declassifies a new descriptor. The descriptor is declassified by the given *className* class. The operation fails in the following cases:

- *descriptorName* is not the name of any existing new descriptor of currently used thesaurus.
- *className* is not one of the allowed classes for new descriptors declassification. Available classes for the declassification of a new descriptor (<thes_nameU> is the upper case name of the currently selected thesaurus, for example: *MERIMEE*):

  1. <thes_nameU>GuideTerm

Example:
```
DeClassifyNewDescriptor(tms_session,"TermeFr`test_desciptor","MERIMEE
GuideTerm");
```

4.4.3 Classify a hierarchy in a facet

Operation : **int ClassifyHierarchyInFacet**(int TMSsessionID, char * hierarchyName, char * facetName)

Input : TMSsessionID, hierarchyName, facetName

This operation classifies a hierarchy in a facet. The hierarchy is classified under the given *facetName* facet. The operation fails in the following cases:

- *hierarchyName* is not the name of any existing hierarchy of currently used thesaurus.
- *facetName* is not the name of any existing facet of currently used thesaurus.

Example:
```
ClassifyHierarchyInFacet(tms_session,
"MERIMEEClass`test_api_hier",              "MERIMEEClass`<architecture
scolaire>");
```

4.4.4 Declassify a hierarchy from a facet

Operation : **int DeClassifyHierarchyFromFacet**(int TMSsessionID, char *hierarchyName, char * facetName)

Input : TMSsessionID, hierarchyName, facetName

This operation declassifies a hierarchy from a facet. The hierarchy is declassified by the given *facetName* facet. The operation fails in the following cases:

- *hierarchyName* is not the name of any existing hierarchy of currently used thesaurus.
- *facetName* is not the name of any existing facet of currently used thesaurus.

Example:
```
DeClassifyHierarchyFromFacet(tms_session,
"MERIMEEClass`test_api_hier", "MERIMEEClass`<architecture
scolaire>");
```

4.4.5 Classify a source

Operation : **int ClassifySource**(int TMSsessionID, char *sourceName, char *className)

Input : TMSsessionID, sourceName, className

This operation classifies a source. The source is classified under the given *className* class. The operation fails in the following cases:

- *sourceName* is not the name of any existing source of currently used thesaurus.
- *className* is not the class *Source* or one of its subclasses (*Citation*, *Serial*, or *Monograph*)

Example:
```
ClassifySource(tms_session,  "Literature`mySource", "Citation");
```

4.4.6 Declassify a source

Operation : **int DeClassifySource**(int TMSsessionID, char * sourceName, char *className)

Input : TMSsessionID, sourceName, className

This operation declassifies a source. The source is declassified by the given *className* class. The operation fails in the following cases:

- *sourceName* is not the name of any existing source of currently used thesaurus.
- *className* is not the class *Source* or one of its subclasses (*Citation*, *Serial*, or *Monograph*)

Example:

```
DeClassifySource(tms_session,  "Literature`mySource", "Citation");
```

4.5 Renaming Operations

4.5.1 Rename a facet

Operation : **int RenameFacet** (int TMSsessionID, char * OldFacetName, char * NewFacetName)

Input : TMSsessionID, OldFacetName, NewFacetName

This operation renames an existing facet. The operation fails in the following cases:

- A facet with the same new name already exists.
- *OldFacetName* is not the name of any existing facet of currently used thesaurus.
- The top term with the same new name already exists.
- Input *NewFacetName* does not contain the correct prefix for a facet of currently used thesaurus.

Example:

```
RenameFacet(tms_session,  "MERIMEEClass`test_api_facet",
"MERIMEEClass`new_test_api_facet");
```

4.5.2 Rename a hierarchy

Operation : **int RenameHierarchy**(int TMSsessionID, char *OldHierarchyName, char * NewHierarchyName)

Input : TMSsessionID, OldHierarchyName, NewHierarchyName

The hierarchy and its top term are appropriately renamed given the new hierarchy name. The operation fails in the following cases:

- A hierarchy with the same new name already exists
- *OldHierarchyName* is not the name of any existing hierarchy of currently used thesaurus.
- The top term with the same new name already exists.
- Input *NewHierarchyName* does not contain the correct prefix for a hierarchy of currently used thesaurus.

Example:

```
RenameHierarchy(tms_session,"MERIMEEClass`test_api_hier","MERIMEEClas
s`new_test_api_hier");
```

4.5.3 Rename a new concept

Operation : **int RenameNewDescriptor** (int TMSsessionID, char * OldDescriptorName, char * NewDescriptorName)

Input : TMSsessionID, OldDescriptorName, NewDescriptorName

This operation renames an existing descriptor. The operation fails in the following cases:

- A descriptor with the same new name already exists
- *OldDescriptorName* is not the name of any existing descriptor of currently used thesaurus.
- Input *NewDescriptorName* does not contain the correct prefix for a descriptor of currently used thesaurus.

Example:

```
RenameNewDescriptor(tms_session,  "TermeFr`test_descriptor",
"TermeFr`new_test_descriptor");
```

4.5.4 Rename a released concept

Operation : **int RenameDescriptor**(int TMSsessionID, RenameNamesCouples NameCouples)

Input : TMSsessionID, NameCouples which is an array of structures:

```
struct RenameNamesCouple {
        char oldName[LOGINAM_SIZE];
        char newName[LOGINAM_SIZE];
};
```

To end the rename chain the oldName of the last structure must set to 0.

This operation renames an existing descriptor. Cyclic and linear renames can be performed with the specified operation. The following cases of renaming can be performed:

- $Term_A$ renamed to $Term_B$, $Term_B$ is a not yet existing name
- $Term_A$ renamed to $Term_B$. In case $Term_B$ is an existing name not belonging in the ObsoleteTerm class of the current thesaurus, the user is asked to give a new name for $Term_B$. The user can give $Term_A$ (performing a cyclic rename) or a not yet existing name. The number of renames performed can be arbitrary.

Example:

```
RenameNamesCouples nameCouples;
strcpy(nameCouples[0].oldName, "TermeFr`COLOMBIER");
strcpy(nameCouples[0].newName, "TermeFr`newCOLOMBIERname");
*(nameCouples[1].oldName) = '\0';
RenameDescriptor(tms_session,  nameCouples);
```

4.5.5 Undo Rename a released concept

Operation : **int UndoRenameDescriptor**(int TMSsessionID, char *DescriptorName)

Input : TMSsessionID, DescriptorName

This operation cancels the "rename descriptor" operation. The necessary renames are performed so that the knowledge base returns to its previous state (before the renames took place). The operation fails in the following case:

- Used for links are targeting to one of the terms participating in the sequence of terms to be renamed. In this case the user is informed and the "gave name to" links should be deleted.

Example:

```
UndoRenameDescriptor(tms_session,  "TermeFr`newCOLOMBIERname");
```

4.5.6 Rename a source

Operation : **int RenameSource**(int TMSsessionID, char * sourceName, char *newSourceName)

Input : TMSsessionID, sourceName, newSourceName

This operation renames an existing source. The operation fails in the following cases:

- A source with the same new name already exists
- *sourceName* is not the name of any existing source of currently used thesaurus.
- Input *newSourceName* does not contain the correct prefix for a source of currently used thesaurus (for example "Literature`" for thesaurus MERIMEE).

Example:

```
RenameSource(tms_session,                    "Literature`mySource",
"Literature`mySrc");
```

4.5.7 Rename an editor

Operation : **int RenameEditor**(int TMSsessionID, char * editorName, char *newEditorName)

Input : TMSsessionID, editorName, newEditorName

This operation renames an existing editor. The operation fails in the following cases:

- An editor with the same new name already exists
- *editorName* is not the name of any existing editor of currently used thesaurus.
- Input *newEditorName* does not contain the correct prefix for an editor of currently used thesaurus (for example "Person`" for thesaurus MERIMEE).

Example:

```
RenameEditor(tms_session,  "Person`myEditor", "Person`myNewEditor");
```

4.6 Abandoning Operations

4.6.1 Abandon a released facet

Operation : **int AbandonFacet**(int TMSsessionID, char * FacetName)

Input : TMSsessionID, FacetName

This operation abandons a facet. The facet is classified as "obsolete". The operation fails in the following cases:

- *FacetName* is not the name of any existing released facet of currently used thesaurus.
- *FacetName* is the name of an already abandoned facet.

Example:

```
AbandonFacet(tms_session,  "MERIMEEClass`<architecture domestique>");
```

4.6.2 Abandon a released hierarchy

Operation : **int AbandonHierarchy**(int TMSsessionID, char *HierarchyName)

Input : TMSsessionID, HierarchyName

This operation abandons a hierarchy. The hierarchy is classified as "obsolete". The operation fails in the following cases:

- *HierarchyName* is not the name of any existing released hierarchy of currently used thesaurus.
- *HierarchyName* is the name of an already abandoned hierarchy.

Example:

```
AbandonHierarchy(tms_session, "MERIMEEClass`<architecture
militaire>");
```

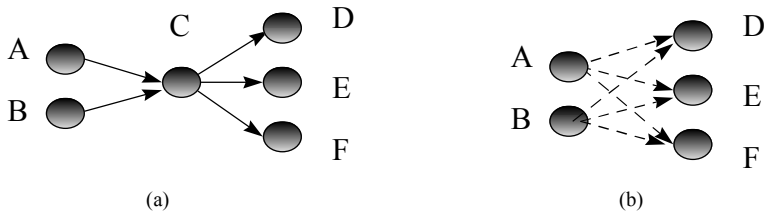
4.6.3 Abandon a released concept

Operation : **int AbandonDescriptor**(int TMSsessionID, char *DescriptorName)

Input : TMSsessionID, DescriptorName

This operation abandons a descriptor. In this case the descriptor is classified as an "Obsolete Descriptor". It remains classified in the hierarchy it belongs to but is detached from it. That is all it's broader and narrower term relations are deleted and appropriate broader term relations are established between its narrower and broader terms as shown in figure 3. The operation fails in the following cases:

- Related, Used For and Alternative term links are originating from or targeted to the descriptor.
- Interthesauri links are originating from the descriptor.
- *DescriptorName* is the name of an already abandoned descriptor.



In (a) descriptor C belongs in hierarchy H and has the broader terms D, E and F and the narrower terms A and B.
In (b) descriptor C is characterized as "Obsolete Descriptor" and is detached from hierarchy H. Its broader and narrower term relations are deleted and appropriate broader term relations (dashed arrows) are established between its narrower and broader terms.

Figure 3. Schema of operation "Abandon Descriptor"

Example:
```
AbandonDescriptor(tms_session,  "TermeFr`CELLIER");
```

4.6.4 Undo Abandon a released facet

Operation : **int UndoAbandonFacet**(int TMSsessionID, char *FacetName)

Input : TMSsessionID, FacetName

This operation cancels the "abandon descriptor" operation for the given facet. In this case the facet is no longer classified as an obsolete.
The operation fails in the following case:

- The facet is not an abandoned facet of currently used thesaurus.

Example:
```
UndoAbandonFacet(tms_session, "MERIMEEClass`<architecture
domestique>");
```

4.6.5 Undo Abandon a released hierarchy

Operation : **int UndoAbandonHierarchy**(int TMSsessionID, char *HierarchyName)

Input : TMSsessionID, HierarchyName

This operation cancels the "abandon descriptor" operation for the given hierarchy. In this case the hierarchy is no longer classified as an obsolete. The operation fails in the following case:

- The hierarchy is not an abandoned hierarchy of currently used thesaurus.

Example:
```
UndoAbandonHierarchy(tms_session,"MERIMEEClass`<architecture
militaire>");
```

4.6.6 Undo Abandon a released concept

Operation : **int UndoAbandonDescriptor**(int TMSsessionID, char *DescriptorName, char *BroaderTerm, char *HierarchyName)

Input : TMSsessionID, DescriptorName, BroaderTerm, HierarchyName

This operation cancels the "abandon descriptor" operation for the given descriptor of the specific hierarchy. In this case the descriptor is no longer classified as an "Obsolete Descriptor", and a broader term relation is established with the given broader term. The descriptor and the given broader term must belong to the same hierarchy. The operation fails in the following cases:

- The *DescriptorName* is not an abandoned descriptor of currently used thesaurus.
- The descriptor does not belong to the specific hierarchy.
- The given broader term is not a descriptor.
- The given broader term is an obsolete descriptor.
- The descriptor and the broader term do not belong to the same hierarchy.

Example:
```
UndoAbandonDescriptor(tms_session,  "TermeFr`CELLIER",
"TermeFr`ABREUVOIR", "MERIMEEClass`<architecture agricole>");
```

4.7 Delete Operations

4.7.1 Delete a new facet

Operation : **int DeleteFacet** (int TMSsessionID, char *FacetName)

Input : TMSsessionID, FacetName

This operation deletes a new facet. The operation fails in the following case:

- *FacetName* is not the name of any existing new facet of currently used thesaurus.

Example:
```
DeleteFacet(tms_session,  "MERIMEEClass`test_api_facet");
```

4.7.2 Delete a facet attribute

Operation : **Int DeleteFacetAttribute**(int TMSsessionID, int linkSysid, char *facetName)

Input : TMSsessionID, linkSysid, facetName

This operation deletes a facet's attribute. The operation fails in the following cases:

- *facetName* is not the name of any existing facet of currently used thesaurus.
- *linkSysid* is not the sysid of any existing link pointing from given facet.

Example:
```
DeleteFacetAttribute(tms_session,1052,"MERIMEEClass`test_api_facet");
```

4.7.3 Delete a new hierarchy

Operation : **Int DeleteHierarchy**(int TMSsessionID, char *HierarchyName)

Input : TMSsessionID, HierarchyName

The new hierarchy and its top term are deleted from the knowledge base. The operation fails in the following cases:

- One or more descriptors are classified in the specific hierarchy.
- *HierarchyName* is not the name of any existing new hierarchy of currently used thesaurus.
- There are links targeting to the top term of the hierarchy originating from other descriptors.

Example:
```
DeleteHierarchy(tms_session,  "MERIMEEClass`test_api_hier");
```

4.7.4 Delete a hierarchy attribute

Operation : **int DeleteHierarchyAttribute** (int TMSsessionID, int linkSysid, char * HierarchyName)

Input : TMSsessionID, linkSysid, HierarchyName

This operation deletes a hierarchy's attribute. The operation fails in the following cases:

- *HierarchyName* is not the name of any existing hierarchy of currently used thesaurus.
- *linkSysid* is not the sysid of any existing link pointing from given hierarchy.

Example:
```
DeleteHierarchyAttribute(tms_session,1052,"MERIMEEClass`test_api_hier
");
```

4.7.5 Delete a new descriptor

Operation : **Int DeleteNewDescriptor** (int TMSsessionID, char * DescriptorName)

Input : TMSsessionID, DescriptorName

This operation deletes a new descriptor. The operation fails in the following case:

- *DescriptorName* is not the name of any existing new descriptor of currently used thesaurus.

Example:
```
DeleteNewDescriptor(tms_session,  "TermeFr`test_descriptor");
```

4.7.6 Delete a new descriptor's attribute

Operation : **int DeleteNewDescriptorAttribute** (int TMSsessionID, int linkSysid,

char * DescriptorName)

Input : TMSsessionID, linkSysid, DescriptorName

This operation deletes a new descriptor's attribute. The operation fails in the following cases:

- *DescriptorName* is not the name of any existing new descriptor of currently used thesaurus.
- *linkSysid* is not the sysid of any existing link pointing from given descriptor.

Example:

```
DeleteNewDescriptorAttribute(tms_session,1054,"TermeFr`test_descr");
```

4.7.7 Delete a released descriptor's attribute

Operation : **int DeleteDescriptorAttribute** (int TMSsessionID, int linkSysid, char *DescriptorName)

Input : TMSsessionID, linkSysid, DescriptorName

This operation deletes a released descriptor's attribute. The operation fails in the following cases:

- *DescriptorName* is not the name of any existing released descriptor of currently used thesaurus.
- *linkSysid* is not the sysid of any existing link pointing from given descriptor.

Example:

```
DeleteDescriptorAttribute(tms_session,  8389224, "TermeFr`ALLEE");
```

4.7.8 Disassociate a concept with terms from other thesauri

Operation : **int DeleteInterThesRelation**(int TMSsessionID, char *FromTerm, char *Category, char *ToTerm)

Input : TMSsessionID, FromTerm, Category, ToTerm

This operation deletes interthesaurus links to a descriptor. The interthesaurus link of the given category associating the *FromTerm* with the *ToTerm* is deleted. In case the *ToTerm* is a collective concept, then it is deleted only when it is not associated with other descriptors. The operation fails in the following cases:

- A link from *FromTerm* to *ToTerm* of type *Category* does not exist.
- The *ToTerm* does not exist.
- *FromTerm* is not the name of any existing descriptor of currently used thesaurus.

Example:

```
DeleteInterThesRelation(tms_session,  "TermeFr`BERGERIE",
"MERIMEE_exact_equivalence , to_RCHME", "EnTerm`garden & lake");
```

4.7.9 Move a concept to another hierarchy

Operation : **int MoveToHierarchy**(int TMSsessionID, char *TargetTerm, char

*CurrentHierarchy, char *NewHierarchy, char *NewBTterm, int Option)

Input : TMSsessionID, TargetTerm, CurrentHierarchy, NewHierarchy, NewBTterm, Option

Depending on the value of input *Option*:

- MOVE_NODE_ONLY
  In this case, the concept is detached from the selected hierarchy (as in the case of "Abandon Descriptor" and is classified in the new hierarchy. A broader term relation is established between the concept and the given broader term.
- MOVE_NODE_AND_SUBTREE
  In this case, the concept and its subtree of broader term relations are detached from the selected hierarchy and are reclassified in the new hierarchy. A broader term relation is established between the concept and the given broader term.
- CONNECT_NODE_AND_SUBTREE
  In this case, the concept and its subtree of broader term relations are NOT detached from the selected hierarchy (as in previous case) and are multiply classified in the new hierarchy. A broader term relation is established between the concept and the given broader term.

The operation fails in the following cases:

- The broader term relation that is going to be added creates a directed cycle of broader term relations.
- *TargetTerm* or *NewBTterm* is not the name of any existing descriptor of currently used thesaurus.
- *CurrentHierarchy* or *NewHierarchy* is not the name of any existing hierarchy of currently used thesaurus.

Example:

```
MoveToHierarchy(tms_session,  "TermeFr`ENTREPOT AGRICOLE",
"MERIMEEClass`<architecture agricole>", "MERIMEEClass`<architecture
artisanale>", "TermeFr`BOUCHERIE", MOVE_NODE_ONLY);
```

4.7.10 Delete a broader term relation of a concept

Operation : **int DeleteBroaderTermLink**(int TMSsessionID, char *FromTerm, char *ToTerm)

Input : TMSsessionID, FromTerm, ToTerm

This operation deletes the broader term relation coming from the input concept (*FromTerm*) and pointing to *ToTerm*. The operation fails in the following cases:

- There is no broader term relation coming from the input concept (*FromTerm*) and pointing to *ToTerm*.
- The broader term relation to be deleted is the last.
- *FromTerm* or *ToTerm* is not the name of any existing descriptor of currently used thesaurus.

Example:

```
DeleteBroaderTermLink(tms_session,"TermeFr`BOUCHERIE","TermeFr`MARCHE
");
```

4.7.11 Delete a source

Operation : **int DeleteSource**(int TMSsessionID, char * sourceName)

Input : TMSsessionID, sourceName

This operation deletes a source. The operation fails in the following case:

- *sourceName* is not the name of any existing source of currently used thesaurus.

Example:

```
DeleteSource(tms_session,  "Literature`mySource");
```

4.7.12 Delete an editor

Operation : **int DeleteEditor**(int TMSsessionID, char * editorName)

Input : TMSsessionID, editorName

This operation deletes an editor. The operation fails in the following case:

- *editorName* is not the name of any existing editor of currently used thesaurus.

Example:

```
DeleteEditor(tms_session,  "Person`myEditor");
```

4.8 Comments Handling Operations

4.8.1 Get a descriptor's comment size

Operation : **int GetDescriptorCommentSize**(int TMSsessionID, char *descriptorName, int *comment_size, char *fromCommentCategory, char *commentCategory)

Input : TMSsessionID, descriptorName, comment_size, fromCommentCategory, commentCategory

This operation gets the comment size of the given descriptor. The kind of the returned comment size is defined by the given comment category (*fromCommentCategory*, *commentCategory*). The comment size is returned in integer pointer *comment_size*. The operation fails in the following cases:

- *descriptorName* is not the name of any existing descriptor of currently used thesaurus.
- given comment category (*fromCommentCategory*, *commentCategory*) is not one of the available by the currently used thesaurus and the current TMS model. (for example: MERIMEEThesaurusConcept->merimee_scope_note for thesaurus MERIMEE).
- *descriptorName* has not any comment value of the specified kind.

Example:

```
int comment_size;
GetDescriptorCommentSize(tms_session,  "TermeFr`MOTTE",
&comment_size, "MERIMEEThesaurusConcept", "merimee_scope_note");
```

4.8.2 Get a descriptor's comment

Operation : **int GetDescriptorComment**(int TMSsessionID, char *descriptorName, char *comment, char *fromCommentCategory, char *commentCategory)

Input : TMSsessionID, descriptorName, comment, fromCommentCategory, commentCategory

This operation gets the comment of the given descriptor. The kind of the returned comment is defined by the given comment category (*fromCommentCategory*, *commentCategory*). The comment is returned in string *comment* which must be initially allocated. The operation fails in the following cases:

- *descriptorName* is not the name of any existing descriptor of currently used thesaurus.
- given comment category (*fromCommentCategory*, *commentCategory*) is not one of the available by the currently used thesaurus and the current TMS model. (for example: MERIMEEThesaurusConcept->merimee_scope_note for thesaurus MERIMEE).
- *descriptorName* has not any comment value of the specified kind.

Example:

```
char *stored_comment=(char*)malloc(comment_size*sizeof(char));
GetDescriptorComment(tms_session,  "TermeFr`MOTTE", stored_comment,
"MERIMEEThesaurusConcept", "merimee_scope_note");
```

4.8.3 Set a descriptor's comment

Operation : **int SetDescriptorComment**(int TMSsessionID, char *descriptorName, char *comment, char *fromCommentCategory, char *commentCategory)

Input : TMSsessionID, descriptorName, comment, fromCommentCategory, commentCategory

This operation sets the comment for the given descriptor. The kind of the new comment is defined by the given comment category (*fromCommentCategory*, *commentCategory*). The new comment is given in string *comment.* The operation fails in the following cases:

- *descriptorName* is not the name of any existing descriptor of currently used thesaurus.
- given comment category (*fromCommentCategory*, *commentCategory*) is not one of the available by the currently used thesaurus and the current TMS model. (for example: MERIMEEThesaurusConcept->merimee_scope_note for thesaurus MERIMEE).
- given new comment string *comment* is NULL or empty.

Example:

```
char comment[100];
strcpy(comment, "this is a test comment");
SetDescriptorComment(tms_session,  "TermeFr`MOTTE", comment,
"MERIMEEThesaurusConcept", "merimee_scope_note");
```

4.8.4 Delete a descriptor's comment

Operation : **int DeleteDescriptorComment**(int TMSsessionID, char *descriptorName, char *fromCommentCategory, char *commentCategory)

Input : TMSsessionID, descriptorName, fromCommentCategory, commentCategory

This operation deletes the comment of the given descriptor. The kind of the comment to be deleted is defined by the given comment category (*fromCommentCategory*, *commentCategory*). The operation fails in the following cases:

- *descriptorName* is not the name of any existing descriptor of currently used thesaurus.
- given comment category (*fromCommentCategory*, *commentCategory*) is not one of the available by the currently used thesaurus and the current TMS model. (for example: MERIMEEThesaurusConcept->merimee_scope_note for thesaurus MERIMEE).
- *descriptorName* has not any comment value of the specified kind.

Example:

```
DeleteDescriptorComment(tms_session,  "TermeFr`MOTTE",
"MERIMEEThesaurusConcept", "merimee_scope_note");
```

Appendix A – An example

In the following we present an example of the usage of the SIS-TMS C application programmatic interface functions. In the example we create on a thesaurus (“ENGLISH”) a new facet (“cpp_test_facet”, a new hierarchy (“cpp_test_hierarchy”) under this facet and a new descriptor (“cpp_test_descriptor”) under the new hierarchy.

```
/*********************************************************************
 *
 *                       Semantic Index System
 *
 *   COPYRIGHT (c) 1992 by Institute of Computer Science,
 *                         Foundation of Research and Technology - Hellas
 *                         POBox 1385, Heraklio Crete, GR-711 10 GREECE
 *
 *
 *                       ALL RIGHTS RESERVED
 *
 *   This software is furnished under license and may be used only in
 *   accordance with the terms of that license and with the inclusion
 *   of the above copright notice.  This software may not be provided
 *   or otherwise made available to, or used by, any other person. No
 *   title to or ownership of the software is hereby transferred.
 *
 *
 *   Module  : main.cpp
 *   Version : 203.4
 *
 *   Purpose :
 *
 *   Author  : Karamaoynas Polykarpos
 *   Creation Date : 1/10/1998          Date of last update : 2/11/99
 *
 *   Remarks :
 *
 *
 *********************************************************************/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#ifdef SIS_WIN32
    #include "conio.h"
#endif

#include "sis_kernel/time.h"
#include "cpp_api/cs_defs.h"
#include "cpp_api/identifier.h"
#include "cpp_api/sis_classes.h"

#include "cpp_api/c_session_wrapper.h"
#include "tms_api/tms_api_defs.h"
#include "tms_api/c_tmsapi_wrapper.h"
#include "messages/translate.h"

void c_tms_api_test(int TmsApiSessionId, int SisApiSessionId);
/*--------------------------------------------------------------------
                            main()
--------------------------------------------------------------------*/
int main(int argc,char **argv)
{
   int SisApiSessionId;
   int TmsApiSessionId;
   int    ret;

   // check the arguments
   if (argc != 2) {
        fprintf(stderr, "The syntax is : %s <server name> <server port>\n",
                                    argv[0]);
        getch();
        exit(1);
    }

    create_SIS_CS_Session(&SisApiSessionId, argv[1], atoi(argv[2]), "", "");
```

```
    if (open_connection(SisApiSessionId) < 0) {
        fprintf(stderr, "No server running at machine: %s and port : %d\n"),
                                    argv[1], atoi(argv[2]);
      getch();
      exit(1);
  }

    // initialize tms_api
   create_TMS_API_Session(&TmsApiSessionId, SisApiSessionId);
   ret = SetThesaurusName(TmsApiSessionId, "ENGLISH");
   if (ret == TMS_APIFail) {
      printf(GetTMS_APIErrorMessage(TmsApiSessionId));
      getch();
      exit(1);
   }

    // test_function
   c_tms_api_test(TmsApiSessionId, SisApiSessionId);

   // close tms_api
   release_TMS_API_Session(TmsApiSessionId);

    // close the connection with the server
   close_connection(SisApiSessionId);
   // Close down the API
   release_SIS_Session(SisApiSessionId);

   return 0;
}

/*---------------------------------------------------------------------
                            c_tms_api_test()
---------------------------------------------------------------------*/
void c_tms_api_test(int TmsApiSessionId, int SisApiSessionId)
{
   int ret = TMS_APISucc;

   printf("-------------------------------------------------\n");
   printf("----------- C TMS API TESTING PROGRAM -----------\n");
   printf("-------------------------------------------------\n");

   // begin of transaction
   begin_transaction(SisApiSessionId);

   // CreateFacet()
   ret = CreateFacet(TmsApiSessionId, "ENGLISHClass`c_test_facet");
   if (ret == TMS_APISucc) printf("Succeded to CreateFacet()\n");
   else goto test_exit_point;

   // CreateHierarchy()
   ret = CreateHierarchy(TmsApiSessionId, "ENGLISHClass`c_test_hierarchy",
"ENGLISHClass`c_test_facet");
   if (ret == TMS_APISucc)
       printf("Succeded to CreateHierarchy()\n");
   else
       goto test_exit_point;

    // CreateDescriptor()
    ret = CreateDescriptor(TmsApiSessionId, "EnTerm`c_test_descriptor",
"EnTerm`c_test_hierarchy");
   if (ret == TMS_APISucc)
       printf("Succeded to CreateDescriptor()\n");
   else
       goto test_exit_point;

   // end of transaction
    end_transaction(SisApiSessionId);

// exit point
test_exit_point:
    if (ret == TMS_APIFail) {
       printf(GetTMS_APIErrorMessage(TmsApiSessionId));
       abort_transaction(SisApiSessionId);
    }

    printf("end\n");
    getch();
}
```

On WIN32 systems the include files that should be used to compile this code are:

```
sis_kernel/time.h,
cpp_api/cs_defs.h,
cpp_api/identifier.h,
cpp_api/sis_classes.h,
cpp_api/c_session_wrapper.h,
tms_api/tms_api_defs.h,
tms_api/c_tmsapi_wrapper.h.
```

On WIN32 systems the libraries (Borland 5.01 libraries) that should be used to link this code are:

Client – Server:

- lib_c_tmsapi_cs_2b.lib (The C interface SIS-TMS)
- lib_cpp_tmsapi_cs_2b.lib (The C++ interface SIS-TMS)
- lib_c_api_session_cs_2b.lib (The C interface SIS API)
- cpp_api_cs_2b.lib (The C++ interface SIS API)
- lib_sis_kernel_2b.lib (The SIS Kernel)
- lib_time_2b.lib (Time functions library)
- ccomms_2b.lib (Client Communications)
- connection_2b.lib (Used to open a connection to the server)
- libl.lib

Direct Access:

No direct access interface (using sessions) is provided for SIS-TMS API.

C API on DLL:

- tmsapi_dll.dll (The C interface SIS-TMS API)

Note: this dll library contains the SIS-TMS API functions as described in section 4 and all the SIS-API functions as described in the *"SIS - Application Programmatic Interface Reference Manual"*.

Appendix B - Changes from previous versions

In the process of upgrading the functionality of the Thesaurus Management System Application Programmatic Interface (SIS-TMS API) some functions were added, other changed name, while other changed the number or the order of their arguments, in order to be more readable or to be in accordance with the API function naming and argument passing conventions.

Changes from version 1.0 to version 1.1

In order to provide real multi-threading to the clients that were using the SIS-TMS, we introduced the notion of sessions for SIS-TMS and SIS C and Java programmatic interface.

Creating multiple instances of TMSAPIClass and QClass (JAPI) was not enough to provide multi-thead access to the SIS-Server, since the underlying libraries (dll's) did not support it. Now these libraries support such mechanism via sessions. Thus now creating multiple TMSAPIClass and QClass instances (JAPI) and creating separate sessions for each instance enables the application developer to have real multi-thread access to the SIS-Server.

All C-API functions have changed: they all take as first argument the TMSsessionID (integer). A session is created by the functions **create_TMS_API_Session(),** which creates a session and returns its ID. These functions have replaced the function **init_TMS_API()**. A session that is no longer needed may be released by **release_TMS_API_Session(),** which replaced the function **close_TMS_API().**

Appendix C - C++ Programmatic Interface

In the following we present the basic differences between the C application programmatic interface (SIS-TMS C-API) and the C++ application programmatic Interface (SIS-TMS C++ API).

The interface is built on `tms_api` class, which provides as public member functions all the functions of described in this document.

Functions such as `create_TMS_API_Session()`, `release_TMS_API_Session()` have no meaning since multi-threading can be achieved by multiple instances of classes tms_api. Thus the first argument of the functions of this SIS-TMS API (TMSsessionID) is omitted.

Applications build with C++API need **different include files** and **libraries** to be compiled. Below we present an example, which is the C++ implementation of the example presented in "Appendix A – An example"

```
/*****************************************************************
 *
 *                       Semantic Index System
 *
 *   COPYRIGHT (c) 1992 by Institute of Computer Science,
 *                         Foundation of Research and Technology - Hellas
 *                         POBox 1385, Heraklio Crete, GR-711 10 GREECE
 *
 *
 *                       ALL RIGHTS RESERVED
 *
 *   This software is furnished under license and may be used only in
 *   accordance with the terms of that license and with the inclusion
 *   of the above copright notice.  This software may not be provided
 *   or otherwise made available to, or used by, any other person. No
 *   title to or ownership of the software is hereby transferred.
 *
 *
 *   Module  : main.cpp
 *   Version : 203.4
 *
 *   Purpose :
 *
 *   Author  : Karamaoynas Polykarpos
 *   Creation Date : 1/10/1998          Date of last update : 2/11/99
 *
 *   Remarks :
 *
 *
 *****************************************************************/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#ifdef SIS_WIN32
    #include "conio.h"
#endif

#include "sis_kernel/time.h"
#include "cpp_api/cs_defs.h"
#include "cpp_api/identifier.h"
#include "cpp_api/sis_classes.h"
#include "sis_kernel/telos_ro.h"
#include "sis_kernel/obj_check.h"
#include "sis_kernel/initial.h"
#include "cpp_api/cs_defs.h"
#ifdef CLIENT_SERVER
    #include "cpp_api/cs_comms.h"
#endif
#include "cpp_api/query_func.h"
#include "cpp_api/set_tuple.h"
#include "cpp_api/q_tmpsets.h"
```

```
#include "cpp_api/q_expstack.h"
#include "cpp_api/cs_errcodes.h"
#include "cpp_api/q_ccache.h"
#include "cpp_api/q_class_header.h"
#include "cpp_api/connection.h"

#include "tms_api/tms_api.h"

sis_api *QC = NULL;
SIS_Connection *CC = NULL;
tms_api *tmsAPI; // global

void test();

/*--------------------------------------------------------------------
                            main()
--------------------------------------------------------------------*/
int main(int argc,char **argv)
{
    int    ret;

   // check the arguments
   if (argc != 2) {
     fprintf(stderr, "The syntax is : %s <server name> <server port>\n",
                                       argv[0]);
     getch();
     exit(1);
   }


   // create_SIS_CS_Session()
   SOCKET S = 0;
   QC = new sis_api(S);
   if(QC == NULL) return -1;
   CC = new SIS_Connection(QC, argv[1], atoi(argv[2]), "", "", "");
   if(CC == NULL) return -1;

   // open_connection()
   CC->open_connection();
   S = CC->GetSocket();
   QC->SetSocket(S);

   // create tms_api object
   tmsAPI = new tms_api(QC, CC);

   ret = tmsAPI->SetThesaurusName("ENGLISH");
   if (ret == TMS_APIFail) {
       printf("%s\n", tmsAPI->GetTMS_APIErrorMessage());
      getch();
      exit(1);
   }

    // test_function
   test();

   // deallocate tms_api object
   delete tmsAPI;

   // close the connection with the server
   CC->close_connection();

   // Close down the API
   if(QC != NULL) delete QC;
   if(CC != NULL) delete CC;

   return 0;
}

/*--------------------------------------------------------------------
                            test()
--------------------------------------------------------------------*/
void test()
{
    int ret = TMS_APISucc;

    printf("-------------------------------------------------\n");
    printf("---------- C++ TMS API TESTING PROGRAM ----------\n");
    printf("-------------------------------------------------\n");

    // begin of transaction
    CC->begin_transaction();
```

```
    // CreateFacet()
    ret = tmsAPI->CreateFacet("ENGLISHClass`cpp_test_facet");
    if (ret == TMS_APISucc) printf("Succeded to CreateFacet()\n");
    else goto test_exit_point;

    // CreateHierarchy()
    ret = tmsAPI->CreateHierarchy("ENGLISHClass`cpp_test_hierarchy",
"ENGLISHClass`cpp_test_facet");
   if (ret == TMS_APISucc) printf("Succeded to CreateHierarchy()\n");
   else goto test_exit_point;

    // CreateDescriptor()
    ret = tmsAPI->CreateDescriptor("EnTerm`cpp_test_descriptor",
"EnTerm`cpp_test_hierarchy");
    if (ret == TMS_APISucc) printf("Succeded to CreateDescriptor()\n");
    else goto test_exit_point;

    // end of transaction
    CC->end_transaction();

   // exit point
test_exit_point:
    if (ret == TMS_APIFail) {
        printf("%s\n", tmsAPI->GetTMS_APIErrorMessage());
        CC->abort_transaction();
    }

    printf("end\n");
    getch();
}
```

On WIN32 systems the include files that should be used to compile this code are:

`sis_kernel/time.h`,
`cpp_api/cs_defs.h`,
`cpp_api/identifier.h`,
`cpp_api/sis_classes.h`,
`sis_kernel/telos_ro.h`,
`sis_kernel/obj_check.h`,
`sis_kernel/initial.h`,
`cpp_api/cs_defs.h`,
`cpp_api/cs_comms.h`,
`cpp_api/query_func.h`,
`cpp_api/set_tuple.h`,
`cpp_api/q_tmpsets.h`,
`cpp_api/q_expstack.h`,
`cpp_api/cs_errcodes.h`,
`cpp_api/q_ccache.h`,
`cpp_api/q_class_header.h` (contains the definition of sis_api),
`cpp_api/connection.h` (contains the definition of SIS_Connection),
`tms_api/tms_api.h` (contains the definition of tms_api).

On WIN32 systems the libraries (Borland 5.01 libraries) that should be used to link this code are:

Client – Server:

- lib_cpp_tmsapi_cs_2b.lib (The C++ interface SIS-TMS API)
- cpp_api_cs_2b.lib (The C++ interface SIS API)
- lib_sis_kernel_2b.lib (The SIS Kernel)
- lib_time_2b.lib (Time functions library)
- ccomms_2b.lib (Client Communications)
- connection_2b.lib (Used to open a connection to the server)
- libl.lib

Direct Access:

No direct access interface (using sessions) is provided for SIS-TMS API.

C API on DLL:

There is no C++ interface on provided on dll.

Appendix D – Java Programmatic Interface

In the following we present the basic differences between the C application programmatic interface (SIS-TMS C-API) and Java application programmatic interface (SIS-TMS JAPI).

The interface is built on class `TMSAPIClass`, which provides as public member functions all the functions of described in this document.

The structures defined for argument passing, such as `RenameNamesCouple`, `IntegerObject`, etc. are implemented as separate classes and all the classes and dll's are provided in a jar file (called "japi13.jar" for version 1.3 of the Java-API, which is the version presented here). We use these structures as arguments to the functions described in this document. The Java classes that replace them are:

- `IntegerObject` replaces `int*` (an integer argument, whenever is used 'pass-by-reference')
- `StringObject` replaces `char*` (a string argument, whenever is used 'pass-by-reference')
- `CMValue` replaces `cm_value`
- `CategorySet` replaces `category_set`
- `Identifier` replaces `IDENTIFIER`
- `Time` replaces `TIME`
- `RenameNamesCouple` replaces `RenameNamesCouple`

Below we present an example which is the Java implementation of the example presented in "Appendix A – An example".

```
import java.io.*;

class tms_test {
    static QClass Q;
    static TMSAPIClass TA;
    static IntegerObject sis_session;
    static IntegerObject tms_session;
    static int transaction_ok = 0;

public tms_test() {
    Q = new QClass();
    TA = new TMSAPIClass();
    sis_session = new IntegerObject();
    tms_session = new IntegerObject();
    int ret;

    Q.create_SIS_CS_Session(sis_session, "agnes",1245, "", "");
    Q.open_connection(sis_session.getValue());

    TA.create_TMS_API_Session(tms_session, sis_session.getValue());

    // begin of transaction

    Q.begin_transaction(sis_session.getValue());

    // CreateFacet()
    descr.setValue("MERIMEEClass`c_test_facet");
    ret = TA.CreateFacet(tms_session.getValue(),descr);
    if (ret == TA.TMS_APISucc)
        System.out.println("Succeded to CreateFacet()");
    check_success(ret);

    // CreateHierarchy()
```

```
    descr1.setValue("MERIMEEClass`c_test_hierarchy");
    descr.setValue("MERIMEEClass`c_test_facet");
    ret = TA.CreateHierarchy(tms_session.getValue(),descr1, descr);
    if (ret == TA.TMS_APISucc)
        System.out.println("Succeded to CreateHierarchy()");
    check_success(ret);

    // CreateDescriptor()
    descr1.setValue("TermeFr`c_test_descriptor");
    descr.setValue("TermeFr`c_test_hierarchy");
    ret = TA.CreateDescriptor(tms_session.getValue(),descr1, descr);
    if (ret == TA.TMS_APISucc)
        System.out.println("Succeded to CreateDescriptor()");
    check_success(ret);

   // end of transaction
    if (transaction_ok == 1)
        Q.end_transaction(sis_session.getValue());

    TA.release_TMS_API_Session(tms_session.getValue());

    Q.close_connection(sis_session.getValue());
    Q.release_SIS_Session(sis_session.getValue());
}

static void check_success(int ret) {
    if (ret == TA.TMS_APIFail) {
        StringObject buf = new StringObject();
        TA.GetTMS_APIErrorMessage(tms_session.getValue(),buf);
        System.out.println("Failed: " + buf);
        Q.abort_transaction(sis_session.getValue());
        transaction_ok = 0;
    } else
        transaction_ok = 1;
}


  public static void main(String[] args) {
  new tms_test();
  System.out.println("press any key to exit....");
  try{
      System.in.read();

    }catch(IOException e){
      System.out.println("Cannot Read!!!");
    }
  }
}
```

INDEX

References

1. D. J. Foskett. Thesaurus. In *Readings in Information Retrieval,* eds. K. Sparck Jones and P. Willet, publisher Morgan Kaufmann, 1997.
2. R. S. Michalski. Beyond Prototypes and Frames: The Two-Tiered Concept Representation. *Categories and Concepts, Theoretical Views and Inductive Data Analysis*, eds. I. Mechelen, J. Hampton, R. Michalski, P. Theuns, 1993.
3. M. Sintichakis and P. Constantopoulos, A Method for Monolingual Thesauri Merging, Proc. of the 20th International Conference on Research and Development in Information Retrieval, ACM SIGIR, July 1997,Philadelphia, PA, USA.
4. M. Doerr and I. Fundulaki. A proposal on extended interthesaurus links semantics. *Technical Report* ICS-FORTH/TR-215, March 1998.
5. Introduction to the Art & Architecture Thesaurus. Published on behalf of The Getty Art History Information Program, Oxford University Press, New York, 1994.
6. D. Soergel. The Arts and Architecture Thesaurus (AAT)-A critical appraisal. *Technical Report*, College of Library and Information Sciences, University of Meryland, 1995.
7. C. Roulin. Sub-Thesauri as part of a metathesaurus. *In International Study Conference on Classification Research, Classification Research for knowledge representation and organisation*, pp. 329-336. Elsevier, 1992.
8. M. Doerr, "Reference Information Acquisition and Coordination", in: "ASIS'97 -Digital Collections: Implications for Users, Funders, Developers and Maintainers", *Proceedings of the 60th Annual Meeting of the American Society for Information Sciences*, " November 1-6 '97, Washington, Vol.34. Information Today Inc.: Medford, New Jersey, 1997. ISBN 1-57387-048-X.
9. M. Doerr and I. Fundulaki, "SIS - TMS: A Thesaurus Management System for Distributed Digital Collections" Proc. 2nd European Conference, ECDL'98, September 1998, Heraklion, Crete, Greece.