Database API

Revision History

Revision 1.0 December 12, 2003 Revised by: afw
Initial version.
Revision 1.01 December 26, 2003 Revised by: afw
Added 'Side Effects' and Transaction Processing Considerations'.
Revision 1.02 April 6, 2004 Revised by: afw
Added two functions: get_cmp_jobs and get_run_jobs
Revision 1.03 June 16, 2004 Revised by: afw
Added two arguments to new_job; added get_method_table function.
Revision 1.04 August 19, 2005 Revised by: jd
Added 'Aborting Job' section and five API functions.
Revision 1.05 May 9, 2006 Revised by: jd
Added five API functions.
Revision 1.06 June 30, 2008 Revised by: jd
Added ten more API functions.

An Application Programming Interface (API) to the Spk Database is defined. The descriptions in this document are independent of any particular programming language.

Table of Contents

Introduction

A *relational database* plays a central role in the architecture of Spk, because it is the means by which the major independent components communicate with one another and because it is the basis for implementation of a central feature of the design, known as the Job History Model [1].

In order to operate a relational database, it is necessary to have a *relational database management system (RDBMS).* Because Spk will be distributed under an open source license, the open source MySQL RDBMS will be used. MySQL has all the features required by Spk and has a very large user base.

All of the widely used RDBMS share many similarities. They implement certain underlying concepts based on the work of E.F.Codd (see for example A Relational Model of Data for Large Shared Data Banks [2]) and many others, which define what a relational database is and does. They all provide a high-level interface language known as the Structured Query Language (SQL). Most also provide standard vendor-independent interfaces based on popular programming languages such as C, Java, and Perl. These interfaces are known as Application Programming Interfaces (API). At a fundamental level, Spk will rely on these standard APIs so that it will be possible to substitute another RDBMS for MySQL, if necessary.

In order to insure the integrity of the database as information is added and modified, it is desirable to limit interactions with the database to a small body of well-tested code. For this reason, Spk includes its own high-level API, which caters directly to the needs of the MDA, Aspk and Cspk, and hence eliminates any need for these

components to use the low-level APIs directly. This Spk-specific API is the topic of this document.

The functions of the Spk API are described in a language independent manner. There are three language-specific implementations of the API, known as language *bindings*. The Spk Database API has a Java binding, a C binding, and a Perl binding to satisfy the needs of the MDA, the Aspk and the Cspk, respectively.

This specification can be best understood with reference to the following related documents:

- Job History Model [3]
- Database Entity-Relationship Model [4]
- Database Schema [5]

Usage Scenario

When a user decides to run a set of scientific data against a model, she calls upon the Spk MDA running on her workstation to submit the model and data as a job. The MDA transmits this information across the Internet to a process known as the MDA Surrogate, which runs on a web server. The MDA Surrogate then creates a job by adding a row to the *job* table of the database, with the model in the *xml_model* field and the data in the *xml_data* field. The *state_code* field of this new row is set to the value 'q2c' thus adding the job to the compiler queue. This queue is a *logical* entity based on values stored in the *job* table, rather than a *physical* entity in which records are added and deleted.

Running independently, either on the machine that runs the web server or on some other machine, an idle Aspk Compiler queries the compiler queue by searching the *job* table for jobs with *state_code* equal to 'q2c' and selects the one which has the highest priority. In a logical sense, it removes the job from the compiler queue when it changes the *state_code* from 'q2c' to 'cmp'. The job is not removed from the *job* table, however. Jobs are never deleted from this table.

The process by which the Aspk Compiler selects the highest priority job that has *state_code* equal to 'q2c' and changes the *state_code* to 'cmp' must occur atomically, because there may be several instances of the Compiler trying to do the same thing simultaneously.

Having selected a job to compile, the Aspk Compiler proceeds to transform the model source into the C++ language. If it encounters errors, it adds an error report to the *report* field of the job, sets the *state_code* field to 'end' and the *end_code* field to 'cerr'. It is then idle and can scan for another job to compile.

If the Aspk Compiler succeeds in compiling the model, it updates *job* by storing its output in the *cpp_source* field and changing the *state_code* to 'q2r'. The Compiler is then idle and can scan for another job to compile.

When a Cspk becomes idle, it queries the run queue by selecting the highest priority job which has *state_code* equal to 'q2r'. It logically removes the job from the queue by setting the *state_code* to 'run'. The selection and the changing of state must occur as an atomic transaction, because there may be another Cspk process trying to select the highest priority job at the same time.

When the Cpsk has finished running the job, it updates the *job* table by storing the results of the computation in the *report* field, setting *state_code* to 'end' and *end_code* to 'srun'. The Cspk is then idle and can scan the queue for another job to run.

The MDA via its MDA Surrogate, periodically queries the database to discover whether or not any of its user's outstanding jobs have reached the *End* state. When

it discovers that one has, it retrieves the contents of the *report* field and presents them to the user.

Aborting Job

When a user decides to abort a job, the MDA sets the *state_code* field of the job row to indicate that the job is aborted or should be aborted by one of the daemons.

If the current *state_code* is 'q2c', the MDA sets the job's *state_code* to 'end' and the job's *end_code* to 'abrt'.

If the current *state_code* is 'cmp', the MDA sets the job's *state_code* to 'q2ac'. The compiler daemon visits the database every second. When the compiler daemon finds the job, it changes the *state_code* to 'acmp' and sends a termination signal to the process executing the compiler instance for the job. When the process terminates, the compiler daemon sets the job's *state_code* to 'end' and the job's end_code to 'abrt'.

If the current *state_code* is 'q2r', the MDA sets the job's *state_code* to 'end' and the job's *end_code* to 'abrt'.

If the current *state_code* is 'run', the MDA sets the job's *state_code* to 'q2ar'. The run-time daemon visits the database every second. When the run-time daemon finds the job, it changes the *state_code* to 'arun' and sends a termination signal to the process executing the SPK driver instance for the job. When the process terminates, the run-time daemon sets the job's *state_code* to 'end' and the job's end_code to 'abrt'.

If the compiler daemon is shutdown in the middle of a job abortion, it may leave jobs with *state_code* 'q2ac' or 'acmp'. When the compiler daemon is restarted, it sets the job's *state_code* to 'end' and the job's *end_code* to 'abrt'.

If the run-time daemon is shutdown in the middle of a job abortion, it may leave jobs with *state_code* 'q2ar' or 'arun'. When the run-time daemon is restarted, it sets the job's *state_code* to 'end' and the job's *end_code* to 'abrt'. In the later case, there may be a checkpoint file in the working directory of the job. The daemon copies the text content of the checkpoint file to the database if there is one.

Functions

In this section the individual API functions will be described. Although they are referred to as *functions* here, they are implemented as *subroutines* in the Perl language and as *class methods* in Java. The descriptions below define those properties that are independent of language binding.

Data Types Used in Function Specifications

Data types are language dependent. Because we are dealing with data fields that are defined in SQL, rather than Perl or Java, the reader should always consult the Database Schema [6] to understand the true specification of a data item.

Here is a glossary of data type names used in the function descriptions below:

database handle

Typically some sort of reference. Each language binding defines a data type which is appropriate.

pair list

A container which holds a list of field names and their associated field values, or references to names and values, depending on the language.

positive integer

A whole number, at least 32 bits in length.

row

A container that can hold a sequence of field values or references to field values, depending on the language.

row-array

An array of rows.

string

A sequence of bytes, this type may contain binary data and may be many megabytes in length.

true, false

Values which, in a particular language, are appropriate for "if" statements.

Function Specifications

connect -- open a connection to the database

Synopsis

handle = connect dbname hostname dbuser dbpassword

Description

Connect opens a connection to a database, returning a handle that represents this connection when passed as an argument to other functions in the API. A process may have several connections open simultaneously. Because each connection ties up resources in the RDBMS, it is important to use the *disconnect* function to close a connection as soon as it is no longer needed.

Side Effects

None.

Transaction Processing Considerations

None.

Table 1. Values returned by connect

Condition	Values	Description
success	handle	Handle to an open database connection.
failure	null or undefined	Failure occurs when the database user does not have permission to connect to the named database on the given host, when the password is incorrect, or when the maximum number of handles has been exceeded. Additional error information is available in the individual language bindings.

Table 2. Arguments to connect

Name of Argument	Data Type	Description
dbname	string	Name of the relational database.
hostname	string	Internet host at which from which user is permitted to make a connection.
dbuser	string	Username for database access. This is not the username of an Spk user.
dbpassword	string	Password for database access. This is not the password of an Spk user.

disconnect -- close a connection to a database

Synopsis

r = disconnect handle

Description

An open database connection is closed, thus releasing resources in the RDBMS.

Side Effects

None.

Transaction Processing Considerations

None.

Table 3. Values returned by disconnect

Condition	Values	Description
success	true	
failure	false	Failure can occur if the handle is invalid. Additional error information is available in the individual language bindings.

Table 4. Arguments to disconnect

Name of Argument	Data Type	Description
handle	database handle	The handle returned by a previous call to the connect function.

new_job -- submit a new job

Synopsis

job_id = new_job handle user_id abstract dataset_id dataset_version model_id model_version xml_source method_code parent is_warm_start is_mail_notice

Description

A new job is created by adding a row to the *job* table. The value of the job_id field of this row is returned.

Side Effects

The state_code field of the job is set to 'q2c'.

The event_time field is set to the time of insertion.

A state transition record is appended to the *history* table.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB) and that the *job_id* field carries the auto_increment modifier.

Table 5. Values returned by new_job

Condition	Values	Description
success	positive integer	Unique value of the job_id field of a newly created row in the *job* table.
failure	0	Failure occurs if the handle, the user_id, or the model_id are invalid. Additional error information is available in the individual language bindings.

Table 6. Arguments to new_job

Name of Argument	Data Type	Description
handle	database handle	A handle to an open database connection.
user_id	positive integer	Key to the user's row in the *user* table.
abstract	string	Short description of the job.
dataset_id	positive integer	Key to a row in the *dataset* table, which contains in its archive field an RCS-compatible archive comprising the current version and all previous versions of a dataset.
dataset_version	string	RCS version of the dataset.
model_id	positive integer	Key to a row in the *model* table, which contains in its archive field an RCS-compatible archive comprising the current version and all previous versions of a model.
model_version	string	RCS version of the model.
xml_source	string	An XML document containing source code for the model, constraints, parameters and presentation directives.
method_code	string	The method that is to be used for the computation.
parent	non-negative integer	The job_id of the job which is the parent of this one. If the job has no parent, this value should be zero.

Name of Argument	Data Type	Description
is_warm_start	true or false	True for being a warm start job; false for otherwise.
is_mail_notice	true or false	True for requesting end-job mail notice; false for otherwise.

job_status -- return the current state of a job

Synopsis

row = job_status handle job_id

Description

The state_code, the event_time of the most recent state transition, and the end_code (which may be null) are returned.

Side Effects

None.

Transaction Processing Considerations

None.

Table 7. Values returned by job_status

Condition	Values	Description
success	row	state_code, event_time, end_code (end_code will be null unless state_code = 'end')
failure	Null or undefined	Failure occurs if the handle or job_id is invalid. Additional error information is available in the individual language bindings.

Table 8. Arguments to job_status

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

Name of Argument	Data Type	Description
job_id	positive integer	Key to a row in the *job* table.

job_history -- retrieve history for a given job

Synopsis

row-array = job_history handle job_id

Description

All rows of the history table that correspond to a given job_id are returned. Each row describes a state transition.

Table 9. Values returned by job_history

Condition	Values	Description
success	row-array	An array containing all history for the given job
failure	Null or unde-fined.	Failure occurs if their is no history for the job_id. Additional error information depends on the individual language bindings.

Table 10. Arguments to job_history

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
job_id	positive integer	Key to a row in the job table.

get_job -- retrieve a row from the job table

Synopsis

row = get_job handle job_id

Description

This function retrieves the entire row from the job table that corresponds to the given job_id.

Table 11. Values returned by get_job

Condition	Values	Description
success	row	An entire row of the job table.
failure	null or undefined	The case where there is no row corresponding to the job id is distinguishable from a failure due to database error.

Table 12. Arguments to get_job

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
job_id	positive integer	A valid key to the job table.

user_jobs -- get status for user's most recent jobs

Synopsis

row-array = user_jobs handle user_id maxnum

Description

Returns an array of rows, each of which contains the current status of one of a given user's jobs. Jobs are sorted in reverse order of job_id, hence the most recently submitted job appears first. The maximum number of jobs to return is given as one of the arguments.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports consistent reads (as does MySQL database, with type=InnoDB).

Table 13. Values returned by user-jobs

Condition	Values	Description
success	row-array	In each row: job_id, abstract, state_code, start_time, event_time, end_code
failure	null or undefined	Failure occurs if the handle or user_id is invalid. Additional error information is available in the individual language bindings.

Table 14. Arguments to user_jobs

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
user_id	positive integer	Key to the user's row in the *user* table.
maxnum	positive integer	Maximum number of rows to be returned. For example, if only the status of the most recent job is desired, this argument would have the value 1.

de_q2c -- select highest priority job from compiler queue

Synopsis

row = de_q2c handle

Description

The *job* table is queried to find the highest priority job with state_code equal to 'q2c'. If the queue is not empty, the fields needed by the Aspk Compiler are returned.

Side Effects

The state code field of the job is set to cmp.

The event_time field is updated to the time of this state transition.

A state transition record is added to the *history* table.

Transaction Processing Considerations

The work of this function is performed within transaction state, between a **begin** database command and a **commit**. The job is found by using a **select for update** command. In case of error, the functions performs a **rollback** before exiting.

Table 15. Values returned by de_q2c

Condition	Values	Description
success	row	job_id, dataset_id, dataset_version, xml_source
failure	null or undefined	Information that can be used to differentiate between an empty queue and an error condition such as an invalid handle is provided by the individual language bindings.

Table 16. Arguments to de_q2c

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

get_cmp_jobs -- get all jobs currently being compiled

Synopsis

row-array = get_cmp_jobs handle

Description

An array of rows containing all jobs with state_code = 'cmp' is returned. The rows contain the fields job_id, dataset_id and xml_source. This function would typically be called by the compiler deamon upon starting up, in order to discover any jobs which were being compiled when the the compiler sub-system was shut down, so that those jobs could be compiled again.

Side Effects

None.

Transaction Processing Considerations

None.

Table 17. Values returned by get_cmp_jobs

Condition	Values	Description
success	row-array	In each row: job_id, dataset_id, dataset_version, and xml_source
failure	null or undefined	Information that can be used to differentiate between a select that returns the empty set and a database error condition is provided by the individual language bindings.

Table 18. Arguments to get_cmp_jobs

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

en_q2r -- add a compiled job to the run queue

Synopsis

r = en_q2r handle job_id cpp_source

Description

The Aspk Compiler uses this function to move a job that it has compiled to the queue of jobs that are ready to be run by the Cspk.

Side Effects

The state_code field of the job is changed to 'q2r'.

The cpp_source field of the job contains c++ source code.

The event_time field is updated to the time of this state transition.

A state transition record is added to the *history* table.

Transaction Processing Considerations

None, assuming that the database supports updates with autocommit (eg. MySQL with type=InnoDB).

Table 19. Values returned by en_q2r

Condition	Values	Description
success	true	
failure	false	Failure occurs if the handle or job_id is invalid. Additional error information is available in the individual language bindings.

Table 20. Arguments to en_q2r

Name of Argument	Data Type	Description
handle	database handle	A handle to an open database connection.
job_id	positive integer	Key to a row in the *job* table.
cpp_source	string	An archive of text files in compressed tar format.

de_q2r -- select the highest priority job from the run queue

Synopsis

row = de_q2r handle

Description

The *job* table is queried for the highest priority job with state_code equal to 'q2r'. If the queue is not empty, the fields needed by Cspk are returned.

Side Effects

The state code field of the job is set to 'run'.

The event_time field is updated to the time of this state transition.

A state transition record is added to the *history* table.

Transaction Processing Considerations

The work of this function is performed within transaction state, between a **begin** database command and a **commit**. The job is found by using a **select for update** command. In case of error, the functions performs a **rollback** before exiting.

Table 21. Values returned by de_q2r

Condition	Values	Description
success	row	job_id, cpp_source
failure	null or undefined	Information that can be used to distinguish between an empty queue and an error condition such as an invalid handle is provided in each of the language bindings.

Table 22. Arguments to de_q2r

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

get_run_jobs -- get all jobs currently being running

Synopsis

row-array = get_run_jobs handle

Description

An array of rows containing all jobs with state_code = 'run' is returned. Each row contains only the field: job_id. This function would typically be called by the runtime deamon upon starting up, in order to discover any jobs which were being run when the the computational sub-system was shut down, so that those jobs could be restarted.

Side Effects

None.

Transaction Processing Considerations

None.

Table 23. Values returned by get_run_jobs

Condition	Values	Description
success	row-array	In each row: job_id, cpp_source
failure	undefined	Information that can be used to differentiate between a select that returns the empty set and a database error condition is provided by the individual language bindings.

Table 24. Arguments to get_run_jobs

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

end_job -- end a job, whether successful or not

Synopsis

r = end_job handle job_id end_code report checkpoint

Description

Cause a job to make the transition to the *End* state. This can occur at the end of a successful run, at the end of a failed run, or in case of errors occurring during Aspk compilation.

Note: this function cannot be used to terminate a compilation or a run. It is used, instead, to record the fact that the job has terminated or completed.

Side Effects

The state_code field of the job is changed to 'end'.

The end_code of the job is changed to $end_code.

The report field contains a report.

The event_time field is updated to the time of this state transition.

Transaction Processing Considerations

None, assuming that the database supports updates and inserts with autocommit (eg. MySQL with type=InnoDB).

Table 25. Values returned by end_job

Condition	Values	Description
success	true	
failure	false	Failure occurs when the handle, job_id, or end_code is invalid. Additional error information is available in the individual language bindings.

Table 26. Arguments to end_job

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
job_id	positive integer	Key to a row in the *job* table.
end_code	string	Valid key to a row in the *end* table.
report	string	XML containing the final report, whether the results of successful computation, partial results after an incomplete run, or description of an error condition.
checkpoint	string	XML containing the checkpoint file, in case the job completed without errors. Otherwise, the string should be null.

job_report -- retrieve the final report for a job

Synopsis

report = job_report handle job_id

Description

The final report, in XML, is returned.

Side Effects

None.

Transaction Processing Considerations

None.

Table 27. Values returned by job_report

Condition	Values	Description
success	string	A string containing a complete XML document.
failure	null or undefined	Failure occurs if the handle or job_id is invalid. The language bindings return additional information about error conditions.

Table 28. Arguments to job_report

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
job_id	positive integer	Key to a row in the *job* table.

new_dataset -- add a dataset to the database

Synopsis

dataset_id = new_dataset handle user_id name abstract archive

Description

The *dataset* table contains the source code for scientific datasets along with identification information. The archive field is an RCS-compatible file which contains the current and all previous versions of the XML text of the dataset. This function is used to establish the initial version of a dataset in the database.

Side Effects

The event_time field is set to the time of insertion.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB), that the *dataset_id* field carries the auto_increment and that the *name* field is unique for the given *user_id* due to a **UNIQUE user_id (user_id, name)** modifier.

Table 29. Values returned by new_dataset

Condition	Values	Description
success	positive integer	Key to a row newly added to the *dataset* table.
failure	0	Failure occurs if the or the user_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 30. Arguments to new_dataset

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
user_id	positive integer	Key to a row in the *user* table.
name	string	Name of the dataset.
abstract	string	Succinct description of the dataset.
archive	string	RCS-compatible source archive.

get_dataset -- retrieve a dataset

Synopsis

list = get_dataset handle dataset_id

Description

Given the key into the *dataset* table, the corresponding row is returned.

Side Effects

None.

Transaction Processing Considerations

None.

Table 31. Values returned by get_dataset

Condition	Values	Description
success	pair list	List of all field names for the *dataset* table, along with associated values.
failure	null or undefined	Failure occurs if the handle or the dataset_id is invalid. Information that can be used to distinguish a missing row from an error is provided by the individual language bindings.

Table 32. Arguments to get_dataset

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
dataset_id	positive integer	Key to a row in the *dataset* table.

update_dataset -- update a row in the *dataset* table

Synopsis

update_dataset handle dataset_id list

Description

A set of fields in a row of the *dataset* table are updated. It is not necessary to update all rows. The list is a set of (name, value) pairs, which specifies the names of fields and the new values which are to be stored into them.

Side Effects

The event_time field is set to the time of the update.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB) and that the *name* field is unique for the given *user_id* due to a **UNIQUE user_id (user_id, name)** modifier.

Table 33. Values returned by update_dataset

Condition	Values	Description
success	true	
failure	false	Failure occurs if the handle or the dataset_id is invalid. Individual language bindings provide additional information in case of error.

Table 34. Arguments to update_dataset

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
dataset_id	positive integer	Key to a row in the *dataset* table.
list	pair list	The list consists of names of fields in the *dataset* table and the corresponding values that should replace current values.

user_datasets -- get a user's datasets

Synopsis

row-array = user_datasets handle user_id

Description

Returns an array of rows, each of which contains a description of one of a given user's datasets. Rows are sorted in order of dataset_id.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports consistent reads (as does MySQL database, with type=InnoDB).

Table 35. Values returned by user_datasets

Condition	Values	Description
success	row-array	In each row: dataset_id, name, abstract.
failure	null or undefined	Failure occurs if the handle or user_id is invalid. Individual language bindings provide information sufficient to recognize the case where no datasets for this user exist and to identify faults.

Table 36. Arguments to user_datasets

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
user_id	positive integer	Key to a row in the *user* table.

new_model -- add a model to the database

Synopsis

model_id = new_model handle user_id name abstract archive

Description

The *model* table contains the source code for scientific models along with identification information. The archive field is an RCS-compatible file which contains the current and all previous versions of the XML text of the model. This function is used to establish the initial version of a model in the database.

Side Effects

The event_time field is set to the time of insertion.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB), that the *model_id* field carries the auto_increment and that the *name* field is unique for the given *user_id* due to a **UNIQUE user_id (user_id, name)** modifier.

Table 37. Values returned by new_model

Condition	Values	Description
success	positive integer	Key to a row newly added to the *model* table.
failure	0	Failure occurs if the or the user_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 38. Arguments to new_model

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
user_id	positive integer	Key to a row in the *user* table.
name	string	Name of the model.
abstract	string	Succinct description of the model.
archive	string	RCS-compatible source archive.

get_model -- retrieve a model

Synopsis

list = get_model handle model_id

Description

Given the key into the *model* table, the corresponding row is returned.

Side Effects

None.

Transaction Processing Considerations

None.

Table 39. Values returned by get_model

Condition	Values	Description
success	pair list	List of all field names for the *model* table, along with associated values.
failure	null or undefined	Failure occurs if the handle or the model_id is invalid. Information that can be used to distinguish a missing row from an error is provided by the individual language bindings.

Table 40. Arguments to get_model

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
model_id	positive integer	Key to a row in the *model* table.

update_model -- update a row in the *model* table

Synopsis

update_model handle model_id list

Description

A set of fields in a row of the *model* table are updated. It is not necessary to update all rows. The list is a set of (name, value) pairs, which specifies the names of fields and the new values which are to be stored into them.

Side Effects

The event_time field is set to the time of the update.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB) and that the *name* field is unique for the given *user_id* due to a **UNIQUE user_id (user_id, name)** modifier.

Table 41. Values returned by update_model

Condition	Values	Description
success	true	
failure	false	Failure occurs if the handle or the model_id is invalid. Individual language bindings provide additional information in case of error.

Table 42. Arguments to update_model

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
model_id	positive integer	Key to a row in the *model* table.
list	pair list	The list consists of names of fields in the *model* table and the corresponding values that should replace current values.

user_models -- get a user's models

Synopsis

row-array = user_models handle user_id

Description

Returns an array of rows, each of which contains a description of one of a given user's models. Rows are sorted in order of model_id.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports consistent reads (as does MySQL database, with type=InnoDB).

Table 43. Values returned by user_models

Condition	Values	Description
success	row-array	In each row: model_id, name, abstract.
failure	null or undefined	Failure occurs if the handle or user_id is invalid. Individual language bindings provide information sufficient to recognize the case where no models for this user exist and to identify faults.

Table 44. Arguments to user_models

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
user_id	positive integer	Key to a row in the *user* table.

new_user -- add a new user

Synopsis

user_id = new_user handle username password ...

Description

Add a new Spk user.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB) and that the *user_id* field carries the auto_increment modifier.

Table 45. Values returned by new_user

Condition	Values	Description

Condition	Values	Description
success	positive integer	Key to a row newly added to the *user* table.
failure	0	Failure occurs if the handle is invalid, or if the list does not include at least a unique username and a password. Additional information about the nature of the failure is provided by individual language bindings.

Table 46. Arguments to new_user

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
list	name and value list	A list if names of fields in the *user* table, along with values to be stored in the corresponding fields. This list must include, at a minimum, username and password, along with their associated values.

update_user -- update a row in the *user* table

Synopsis

r = update_user handle user_id list

Description

Update fields of the row in the *user* table that corresponds to user_id.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 47. Values returned by update_user

Condition	Values	Description

Condition	Values	Description
success	true	
failure	false	Failure occurs if the handle or the user_id is invalid. Individual language bindings provide additional information in case of error.

Table 48. Arguments to update_user

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
user_id	positive integer	Key to a row in the *user* table.
list	pair list	The list consists of names of fields in the *user* table and the corresponding values that should replace current values.

get_user -- retrieve a user record by username

Synopsis

list = get_user handle username

Description

This function retrieves the entire row in the *user* table corresponding to the username. Since username is a unique alternative key, there will be no more than one row returned.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports consistent reads (as does MySQL database, with type=InnoDB).

Table 49. Values returned by get_user

Condition	Values	Description

Condition	Values	Description
success	pair list	List of all field names for the *user* table, along with their associated values.
failure	null or undefined	Failure occurs if the handle or the username is invalid. Information sufficient to distinguish the case of a nonexistent user from various error conditions is provided by the language bindings.

Table 50. Arguments to get_user

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
username	string	User name for logging in to Spk via the MDA.

get_end_table - - return the entire end table

Synopsis

row-array = get_end_table handle

Description

The end table maps end codes into their English language equivalents. The table is not very large, hence it is most efficient for the function to return the entire table.

Table 51. Values returned by get_end_table

Condition	Values	Description
success	row-array	end_code, end_name
failure	null or undefined	The database "select" that this function performs should never come up empty, hence the only basis for failure would be a database error, in which case the result would be null or undefined, depending on the language binding.

Table 52. Arguments to get_end_table

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

get_method_table - - return the entire method table

Synopsis

row-array = get_method_table handle

Description

The method table maps method codes into their English language equivalents. The table is not very large, hence it is most efficient for the function to return the entire table.

Table 53. Values returned by get_method_table

Condition	Values	Description
success	row-array	method_code, method_name
failure	null or undefined	The database "select" that this function performs should never come up empty, hence the only basis for failure would be a database error, in which case the result would be null or undefined, depending on the language binding.

Table 54. Arguments to get_method_table

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

get_state_table - - return the entire state table

Synopsis

row-array = get_state_table handle

Description

The state table maps state codes into their English language equivalents. The table is not very large, hence it is most efficient for the function to return the entire table.

Table 55. Values returned by get_state_table

Condition	Values	Description
success	row-array	state_code, state_name
failure	null or undefined	The database "select" that this function performs should never come up empty, hence the only basis for failure would be a database error, in which case the result would be null or undefined, depending on the language binding.

Table 56. Arguments to get_state_table

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

abort job -- abort a job

Synopsis

r = abort_job handle job_id

Description

Abort a job when the job is in one of the four possible states. If the job's state_code equals to 'q2c', set the job's state_code to 'end' and the job's end_code to 'abrt'. If the job's state_code equals to 'cmp', set the job's state_code to 'q2ac'- queued to abort compiling. If the job's state_code equals to 'q2r', set the job's state_code to 'end' and the job's end_code to 'abrt'. If the job's state_code equals to 'run', set the job's state_code to 'q2ar' - queued to abort running.

Side Effects

The event_time field is updated to the time of this transition.

A state transition record is appended to the *history* table.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB) and that the *job_id* field carries the auto_increment modifier.

Table 57. Values returned by abort_job

Condition	Values	Description
success	true	The job's state_code has been set to either 'end, 'q2ac' or q2ar'.
failure	false	The job's state_code has not been set to either 'end, 'q2ac' or q2ar'

Table 58. Arguments to abort_job

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
job_id	positive integer	A valid key to the job table.

get job ids -- get all job ids with a given state_code

Synopsis

list = get_job_ids handle state_code

Description

Get job_ids of all jobs as an array with a given state_code.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB) and that the *job_id* field carries the

auto_increment modifier.

Table 59. Values returned by get_job_ids

Condition	Values	Description
success	integer array	An array of job_ids of all the jobs with the given state_code, 0 if there is no job with the given state_code
failure	null or undefined	Failure occurs if the handle or job_id is invalid. Additional error information is available in the individual language bindings.

Table 60. Arguments to get_job_ids

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
state_code	string	state_code of the jobs to get.

de_q2ac -- remove highest priority job from aborting compiler queue

Synopsis

job_id = de_q2ac handle

Description

Remove the highest priority job from the aborting compiler queue, so that it can be aborted by the compiler daemon.

Side Effects

The state_code field of the job is set to acmp.

The event_time field is updated to the time of this state transition.

A state transition record is added to the *history* table.

Transaction Processing Considerations

The work of this function is performed within transaction state, between a **begin** database command and a **commit**. The job is found by using a **select for update** command. In case of error, the functions performs a **rollback** before exiting.

Table 61. Values returned by de_q2ac

Condition	Values	Description
success	integer	job_id of highest priority aborting compilation job, false if aborting compiler queue is empty
failure	null or undefined	Failure occurs if the handle is invalid. Additional error information is available in the individual language bindings.

Table 62. Arguments to de_q2ac

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

de_q2ar -- remove highest priority job from aborting run queue

Synopsis

job_id = de_q2ar handle

Description

Remove the highest priority job from the aborting run queue, so that it can be aborted by the run-time daemon.

Side Effects

The state_code field of the job is set to arun.

The event_time field is updated to the time of this state transition.

A state transition record is added to the *history* table.

Transaction Processing Considerations

The work of this function is performed within transaction state, between a **begin** database command and a **commit**. The job is found by using a **select for update** command. In case of error, the functions performs a **rollback** before exiting.

Table 63. Values returned by de_q2ar

Condition	Values	Description
success	integer	job_id of highest priority aborting run job, false if aborting run queue is empty
failure	null or undefined	Failure occurs if the handle is invalid. Additional error information is available in the individual language bindings.

Table 64. Arguments to de_q2ar

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

get_mail_notice -- get end-job email notice request option

Synopsis

r = get_mail_notice handle job_id

Description

Given a job_id, get end-job email notice request option for the job.

Side Effects

None.

Transaction Processing Considerations

None, as long as the job_id is valid.

Table 65. Values returned by get_mail_notice

Condition	Values	Description
success	true or false	true if the end-job email notice request option is set, false if otherwise
failure	null or undefined	Failure occurs if the handle is invalid. Additional error information is available in the individual language bindings.

Table 66. Arguments to get_mail_notice

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
job_id	integer	The key to a row in the job table.

get_user_by_id -- retrieve a user record by user_id

Synopsis

list = get_user_by_id handle user_id

Description

This function retrieves the entire row in the *user* table corresponding to the user_id. Since user_id is a unique key, there will be no more than one row returned.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports consistent reads (as does MySQL database, with type=InnoDB).

Table 67. Values returned by get_user_by_id

Condition	Values	Description

Condition	Values	Description
success	pair list	List of all field names for the *user* table, along with their associated values.
failure	null or undefined	Failure occurs if the handle or the user_id is invalid. Information sufficient to distinguish the case of a nonexistent user from various error conditions is provided by the language bindings.

Table 68. Arguments to get_user_by_id

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
user_id	positive integer	User ID of the user.

set_job_abstract -- set job abstract.

Synopsis

r = set_job_abstract handle job_id

Description

This function sets the job abstact *job* table corresponding to the job_id.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports consistent reads (as does MySQL database, with type=InnoDB).

Table 69. Values returned by set_job_abstract

Condition	Values	Description
success	true	

Condition	Values	Description
failure	false	Failure occurs if the handle, the user_id or the job_id is invalid.

Table 70. Arguments to get_user_by_id

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
user_id	positive integer	User ID of the user.
job_id	positive integer	Job ID of the job.
abstraction	string	The specified job abstract.

new_group -- add a new group

Synopsis

group_id = new_group handle group_name

Description

Add a new Spk user group.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB) and that the *group_id* field carries the auto_increment modifier.

Table 71. Values returned by new_group

Condition	Values	Description
success	positive integer	Key to a row newly added to the *team* table.

Condition	Values	Description
failure	0	Failure occurs if the handle is invalid, or if the group_name has already been used by an existing group. Additional information about the nature of the failure is provided by individual language bindings.

Table 72. Arguments to new_group

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
group_name	string	Specified group name.

new_group_member -- add a new group member

Synopsis

r = new_group_member handle username group_id

Description

Add a user to a group.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 73. Values returned by new_group

Condition	Values	Description
success	true	Group ID is set to the user in the *user* table.
failure	false	Failure occurs if the handle is invalid, or if the username or the group_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 74. Arguments to new_group

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
username	string	Specified username.
group_id	positive integer	Specified group_id.

get_group_users -- retrieve all users of a group

Synopsis

list = get_group_users handle group_id

Description

This function retrieves the a list of entire rows in the *user* table corresponding to the group_id.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports consistent reads (as does MySQL database, with type=InnoDB).

Table 75. Values returned by get_group_users

Condition	Values	Description
success	pair list	List of all field names for the *user* table, along with their associated values.
failure	null or undefined	Failure occurs if the handle or the group_id is invalid. Information sufficient to distinguish the case of a nonexistent user from various error conditions is provided by the language bindings.

Table 76. Arguments to get_user

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
group_id	positive integer	Specified group ID.

new_folder -- add a new folder

Synopsis

r = new_folder handle name parent user_id

Description

Add a folder to a user account.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 77. Values returned by new_foler

Condition	Values	Description
success	folder_id	A row is added to the *folder* table.
failure	0	Failure occurs if the handle is invalid, or if the name, parent or the user_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 78. Arguments to new_folder

Name of Argument	Data Type	Description

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
name	string	Specified folder name.
parent	positive integer	Specified folder_id of the parent folder.
user_id	positive integer	Specified user_id of the folser owner.

rename_folder -- rename a folder

Synopsis

r = rename_folder handle folder_id name user_id

Description

Rename a folder.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 79. Values returned by delete_foler

Condition	Values	Description
success	true	A folder name in the *folder* table is set.
failure	false	Failure occurs if the handle is invalid, or if the folder_id, name or the user_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 80. Arguments to delete_folder

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
folder_id	positive integer	Specified folder_id of the folder to rename.
name	string	Specified the folder name.
user_id	positive integer	Specified user_id of the folser owner.

delete_folder -- delete a folder

Synopsis

r = delete_folder handle folder_id user_id

Description

Delete a folder.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 81. Values returned by delete_foler

Condition	Values	Description
success	true	A row in the *folder* table is deleted.
failure	false	Failure occurs if the handle is invalid, or if the folder_id or the user_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 82. Arguments to delete_folder

Name of Argument	Data Type	Description

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
folder_id	positive integer	Specified folder_id of the folder to delete.
user_id	positive integer	Specified user_id of the folder owner.

move_folder -- move a folder

Synopsis

r = move_folder handle folder_id parent user_id

Description

Move a folder to specified parent.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 83. Values returned by move_foler

Condition	Values	Description
success	true	The parent in the *folder* table is set.
failure	false	Failure occurs if the handle is invalid, or if the folder_id, parent or the user_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 84. Arguments to move_folder

Name of Argument	Data Type	Description

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
folder_id	positive integer	Specified folder_id of the folder to move.
parent	positive integer	Specified folder_id of the parent.
user_id	positive integer	Specified user_id of the folser owner.

move_jobs -- move jobs

Synopsis

r = move_jobs handle job_ids parent

Description

Move jobs to specified folder.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 85. Values returned by move_jobs

Condition	Values	Description
success	true	The folder_id in the *job* table is set.
failure	false	Failure occurs if the handle is invalid, or if the job_ids or parent is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 86. Arguments to move_jobs

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
job_ids	list of positive integers	Specified job_ids of the jobs to move.
parent	positive integer	Specified folder_id of the jobs to move into.

get_folder_path -- get folder path

Synopsis

r = get_folder_path handle folder_id user_id

Description

Retrieves path and name of a specified folder.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 87. Values returned by get_folder_path

Condition	Values	Description
success	string	The path and name of the folder.
failure	empty string	Failure occurs if the handle is invalid, or if the folder_ids or user_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 88. Arguments to get_folder_path

Name of Argument	Data Type	Description

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
folder_id	positive integer	folder_id of the specified folder.
user_id	positive integer	Specified user_id of the folder owner.

get_sub_folders -- get sub folders

Synopsis

r = get_sub_folders handle folder_id user_id

Description

Retrieves sub folders of a folder.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 89. Values returned by get_sub_folders

Condition	Values	Description
success	row-array	The row of the *folder* table selected by folder_id and user_id.
failure	null or undefined	Failure occurs if the handle is invalid, or if the folder_id or the user_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 90. Arguments to get_sub_folders

Name of Argument	Data Type	Description

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
folder_id	positive integer	Specified folder_id of the folder to retrieve sub folders.
user_id	positive integer	Specified user_id of the folder to retrieve sub folders.

user_job_count -- count total number of jobs of a user

Synopsis

r = user_job_count handle user_id

Description

Retrieves total number of jobs of a user.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 91. Values returned by user_job_count

Condition	Values	Description
success	positive integer	The total number of jobs owned by the specified user.
failure	0	Failure occurs if the handle is invalid, or if the user_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 92. Arguments to user_job_count

Name of Argument	Data Type	Description

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
user_id	positive integer	Specified user_id of the user to retrieve job count.

get_folder_path -- get folder tree

Synopsis

r = get_folder_tree handle user_id

Description

Retrieves folder tree of a specified user.

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 93. Values returned by get_folder_tree

Condition	Values	Description
success	string	The folder tree of the user.
failure	empty string	Failure occurs if the handle is invalid, or if the user_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 94. Arguments to get_folder_tree

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.

Name of Argument	Data Type	Description
user_id	positive integer	Specified user_id of the folder tree owner.

set_parallel -- set a job to run in parallel mode

Synopsis

r = set_parallel handle job_id

Description

Specify a job to run in parallel processing mode..

Side Effects

None.

Transaction Processing Considerations

None, assuming that the database supports transactions with autocommit (eg., MySQL database with TYPE=InnoDB).

Table 95. Values returned by set_parallel

Condition	Values	Description
success	true	The total number of jobs owned by the specified user.
failure	false or undefined	Failure occurs if the handle is invalid, or if the job_id is invalid. Additional information about the nature of the failure is provided by individual language bindings.

Table 96. Arguments to set_parallel

Name of Argument	Data Type	Description
handle	database handle	Handle to an open database connection.
job_id	positive integer	Specified job_id of the job to run in parallel processing mode.

Function Availability by Language

Not all functions are implemented in each of the three language bindings.

Table 97. Functions by Language

Function	Perl	Java
connect	x	x
disconnect	x	x
new_job	x	x
job_history		x
get_job		x
job_status	x	x
user_jobs	x	x
de_q2c	x	
get_cmp_jobs	x	
en_q2r	x	
de_q2r	x	
get_run_jobs	x	
end_job	x	x
job_report	x	x
new_dataset	x	x
get_dataset	x	x
update_dataset	x	x
user_datasets	x	x
new_model	x	x
get_model	x	x
update_model	x	x
user_models	x	x
new_user	x	x
update_user	x	x
get_user	x	x
get_end_table		x
get_method_table		x
get_state_table		x
abort_job		x
get_job_ids	x	
de_q2ac	x	
de_q2ar	x	
get_mail_notice	x	
get_user_by_id		x
set_job_abstract		x
new_group		x

Function	Perl	Java
new_group_member		x
get_group_users		x
new_folder		x
rename_folder		x
delete_folder		x
move_folder		x
move_jobs		x
get_foler_path		x
get_sub_folers		x
user_job_count		x
get_foler_tree		x
set_parallel	x	

Notes

1. ../jobHistory/jobHistory.html
2. http://www.acm.org/classics/nov95/toc.html
3. ../jobHistory/jobHistory.html
4. ../erModel/erModel.html
5. ../dbSchema/dbSchema.html
6. ../dbSchema/dbSchema.html