Aktions' Design Specification

Table of Contents

Introduction

Abstract

This document describes the design of the second iteration of Aktions. It contains an overview of the system at large, and detailed specifications concerning the implementation of the functionality defined within our requirements document..

Purpose of this document

The purpose of this document is to describe the structure and implementation for the second iteration of Aktions. Readers of this document should have read our second requirements document. Readers of this document should have a thorough knowledge of computers, the Qt libraries, and the KDE Libraries.

Goals

The goal of this document is to anticipate and dictate the design of the Aktions Usability Toolkit. This document details:

- A refactored API,
- An administrative console that manages usability tests, and
- The hooks used to instrument the KDE libraries.

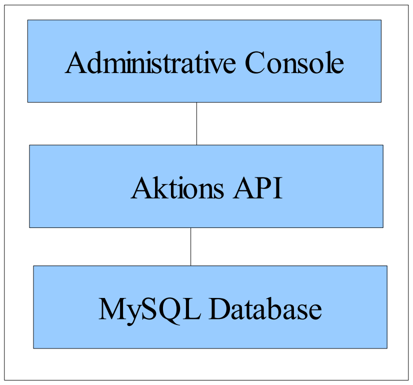
Overview of the rest of the document

The rest of this document describes the design that implements the goals for this iteration. It contains a detailed description of Aktions, its components, and the interactions between them. It enumerates

1. The definition of the functions and objects exported by Aktions API,
2. The design of the back-end of the system,
3. The data model that stores information for Aktions,
4. The details covering the implementation of the administrative console, and
5. The precise hooks used to instrument the KDE libraries.

General Description

While users interact with their system, KDE generates messages via KAction objects. Hooks within KActions collect information concerning what action occurred. This information is submitted through Aktions' Action object. Aktions adds meta-data describing the currently running study and task, then stores the information in a MySQL database.



Usability experts can query the database using the Administrative Console. The console queries the Aktions API using a layered set of objects. AktionsAPI objects generate studies and generic queries. Studies generate tasks, and tasks generate actions. Comments are generated by AktionsAPI, Studies, and Tasks. The Administrative Console displays those objects.

The Administrative Console also coordinates which study and task are currently running. The console uses the study and task objects to manage this. When a user creates a comment, the administrative console feeds the comment into the appropriate object.

Application Programming Interface (API)

The API provides a unified mechanism to store and retrieve usability information. It encapsulates data gathering, filtering, and retrieval. It exports administrative capabilities that group data into studies and tasks, as well as annotate studies and tasks with comments.

The design team decided the best manner to implement the requirements was to create a rats nest of interrelated objects. We can monitor the complaints generated by the use of our interface. Eventually, a developer will become irked enough to redesign a sane interface for Aktions.

The exposed API consists of the following objects:

- **AktionsAPI** – Provides miscellaneous and Aktions-wide functionality.
- **Action** – Most basic exposed data structure. Contains information related to what the user does with their system.
- **Study** – Container for Tasks. Used to administrate and generate usability studies.
- **Task** – Container for Actions. Indirectly used to administrate usability studies. Used as an intermediate object to generate usability studies.
- **Comment** – Storage for any annotations a user might want to record. Can be attached to a study or task.

The internal objects are:

- **Hashed<Object>s** – These objects are wrappers for Actions, Studies, Tasks, and Comments. They create unique keys so that their related objects can be stored in the database.
- **QueryResultEntries** – These objects store the results of any miscellaneous queries that may be placed through the AktionsAPI object. They are used for debugging purposes.
- **StudyTaskActivity** – Holds pause/resume times for studies and tasks.

- **InternalAPI** – Serves as a break point that encapsulates all internal behaviors. Inserted in the API as a stub class so that parallel development can occur.
- **Input** – The input object receives information coming from the KDE environment. It prepares and stores information within a MySQL database.
- **Output** – The output object retrieves information from the MySQL database, and forwards it to the external API.
- **Database** – The database object handles connecting to and querying the MySQL database. It also ensures that the proper tables are available to Aktions.
- **AktionsSQLConverter** – The AktionsSQLConverter object converts query calls generated by the API into SQL statements understood by MySQL.

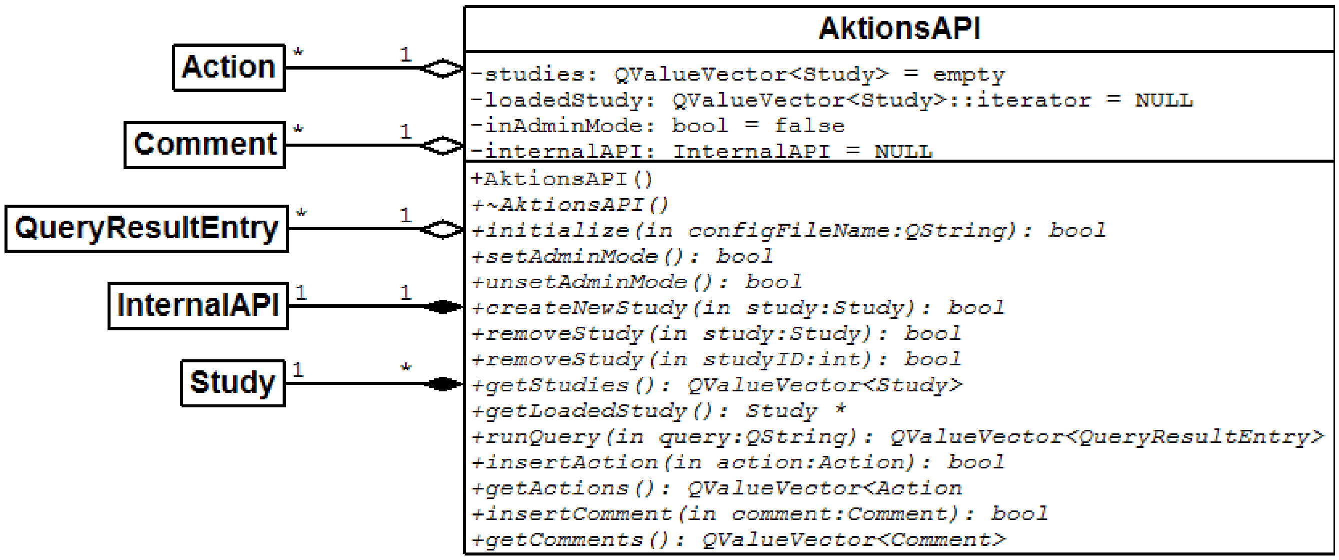
Administrative Console

The administrative console's design is automatically generated by the KDevelop IDE. Due to this behavior, its description is beyond the scope of this document.

Application Programming Interface (API) Definitions

External Definitions

AktionsAPI


AktionsAPI
-studies: QValueVector<Study> = empty
-loadedStudy: QValueVector<Study>::iterator = NULL
-inAdminMode: bool = false
-internalAPI: InternalAPI = NULL
+AktionsAPI()
+~AktionsAPI()
+initialize(in configFileName:QString): bool
+setAdminMode(): bool
+unsetAdminMode(): bool
+createNewStudy(in study:Study): bool
+removeStudy(in study:Study): bool
+removeStudy(in studyID:int): bool
+getStudies(): QValueVector<Study>
+getLoadedStudy(): Study *
+runQuery(in query:QString): QValueVector<QueryResultEntry>
+insertAction(in action:Action): bool
+getActions(): QValueVector<Action
+insertComment(in comment:Comment): bool
+getComments(): QValueVector<Comment>
Action
Comment
QueryResultEntry
InternalAPI
Study
*
1

Class Description:

This is the main entry point into the Aktions API. It provides initialization of the system, study creation and manipulation, and insertion of Actions and Comments that are not to be associated with any study.

Data Members:

Variable	*Description*
studies	A list of references to studies on the server
loadedStudy	The study which is presently loaded off of the database server.
inAdminMode	If the interface can send messages, this is true.
internalAPI	Holds the internal API that the external API is related to.

Functions:

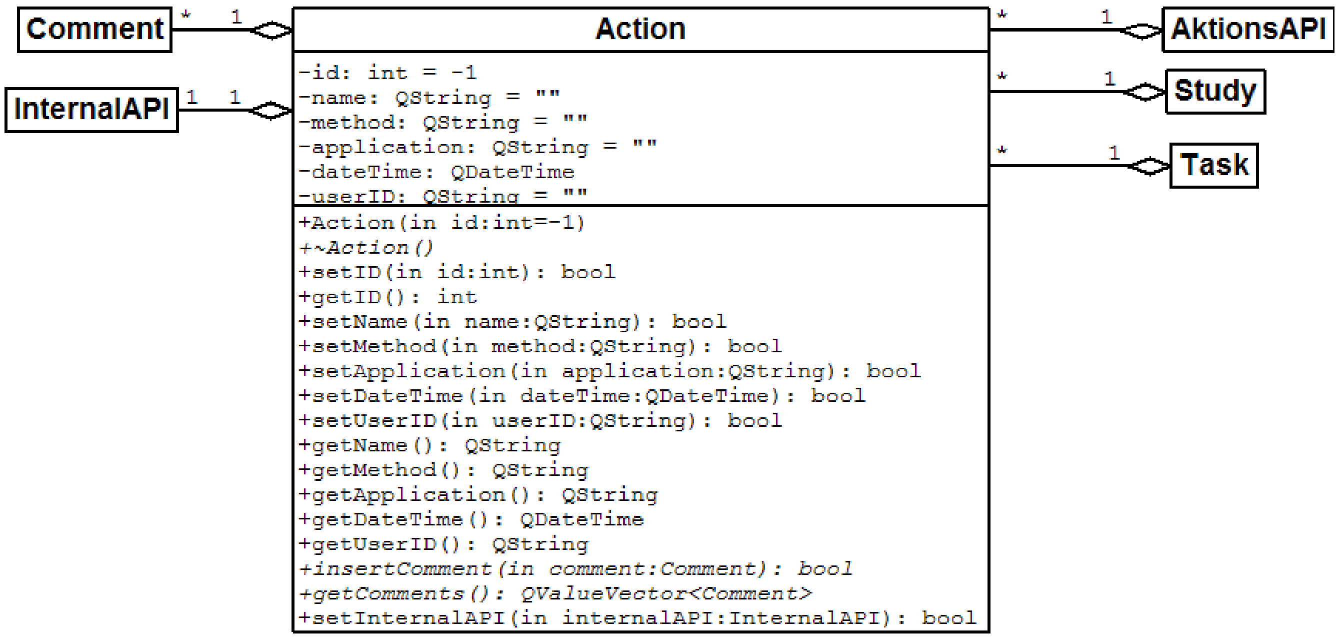
Function	*Description*
initialize(in configFileName)	Loads and applies the configuration for Aktions from file ConfigFileName. ConfigFileName is an absolute location. Returns true if Aktions initialized properly.

Function	***Description***
setAdminMode()	Attempts to activate administrator privileges for this instance of Aktions. Returns true if this action is successful.
unsetAdminMode()	Releases administrator privileges for this instance of Aktions. Returns true if this action is successful.
createNewStudy(in Study)	Inserts the provided study into the Aktions database, and sets the study as the currently loaded study. Returns false if this operation fails.
removeStudy(in Study)	Removes any study that matches Study from the Aktions database. If the study was loaded, it unloads the study. Returns false if this operation fails, or if there are no matching studies in the database.
removeStudy(in StudyID)	Removes any study whose ID value in the database is StudyID. if the study was loaded, it unloads the study. Returns false if this operation fails, or if there are no matching studies in the database.
getStudies()	Returns a list of all studies stored by the database. The list will be empty if there are no studies in the database.
getLoadedStudy()	Returns a pointer to the currently loaded study. Returns NULL if no study is loaded.
runQuery(in Query)	Query must be valid SQL. Runs Query directly through the database. Returns results of the query stored in a vector of QueryResultEntries.
insertAction(in action)	Stores the action in the database. The action stored will not be associated with any studies or tasks. Returns true if this action is successful.
getActions( )	Returns a list of all actions stored in the database that are not associated with a study or task, stored as a vector of actions.
insertComment(in comment)	Stores the comment in the database. Returns true if this action is successful.
getComments( )	Returns a list of all actions stored in the database that are not associated with a study or task.

Related Objects:

Object	***Relation***
Action	Used as a parameter to AktionsAPI
Comment	Used as a parameter to AktionsAPI
QueryResultEntry	Used as a parameter to AktionsAPI
InternalAPI	Used as a parameter to AktionsAPI
Study	Owned by and used as a parameter to AktionsAPI

Action


Comment
InternalAPI
Action
AktionsAPI
Study
Task
*
1
-id: int = -1
-name: QString = ""
-method: QString = ""
-application: QString = ""
-dateTime: QDateTime
-userID: QString = ""
+Action(in id:int=-1)
+~Action()
+setID(in id:int): bool
+getID(): int
+setName(in name:QString): bool
+setMethod(in method:QString): bool
+setApplication(in application:QString): bool
+setDateTime(in dateTime:QDateTime): bool
+setUserID(in userID:QString): bool
+getName(): QString
+getMethod(): QString
+getApplication(): QString
+getDateTime(): QDateTime
+getUserID(): QString
+insertComment(in comment:Comment): bool
+getComments(): QValueVector<Comment>
+setInternalAPI(in internalAPI:InternalAPI): bool

Class Description:

This holds the information associated with an action in the database. It is also used to insert Comments associated with a given Action.

Data Members:

Variable	***Description***
id	A unique identifier, assigned by the database. Set to -1 when the object has not been stored in the database.
name	The action performed by a user.
method	The way a user performed the action.
application	The application that the action occurred in.
dateTime	The time that an action occurred.
userID	The user that performed an action.

Functions:

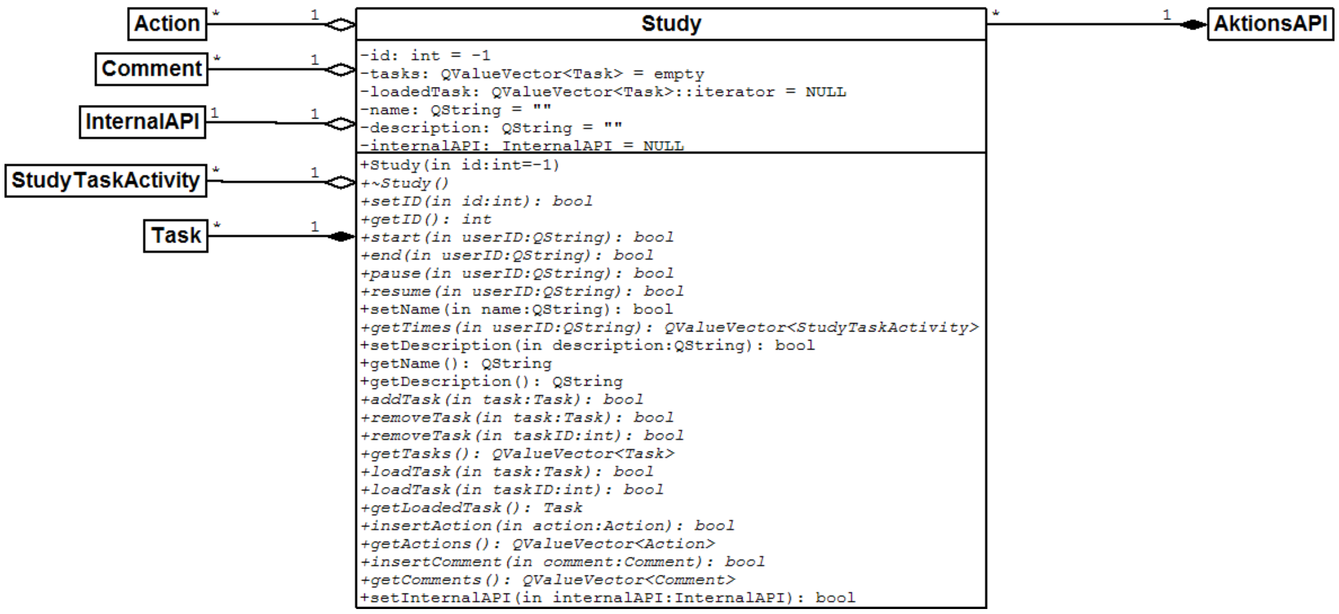
Function	***Description***
setID(in ID)	Sets the id variable to ID. Used by the InternalAPI. Returns true if this operation is sucessful.
getID( )	Returns the contents of the id variable. If the Action has not been stored in the database, this function returns -1.
setName(in name)	Sets the contents of the name variable. Returns true if this operation is sucessful.
setMethod(in method)	Sets the contents of the method variable. Returns true if this operation is sucessful.
setApplication(in application)	Sets the contents of the application variable. Returns true if this

Function	*Description*
	operation is sucessful.
setDateTime(in DateTime)	Sets the contents of the dateTime variable. Returns true if this operation is sucessful.
setUserID(in ID)	Sets the contents of the UserID variable. Returns true if this operation is sucessful.
getName( )	Returns the contents of the getName variable.
getMethod( )	Returns the contents of the method variable.
getApplication( )	Returns the contents of the application variable.
getDateTime( )	Returns the contents of the dateTime variable.
getUserID( )	Returns the contents of the userID variable.
insertComment(in comment)	Inserts a comment associated with this action into Aktions.
getComments( )	Returns a list of allcomments associated with this action.
setInternalAPI(in internalAPI)	Sets the instance of the Internal API that Aktions uses. Returns true if this operation is sucessful.

Related Objects:

Object	*Relation*
Comment	Used as a parameter to Action.
InternalAPI	Used for access to database facilities.
AktionsAPI	Action passed as parameter to AktionsAPI
Study	Action passed as parameter to Study
Task	Action passed as parameter to Task

Study


Action
*
1
Comment
*
1
InternalAPI
1
1
StudyTaskActivity
*
1
Task
*
1
Study
-id: int = -1
-tasks: QValueVector<Task> = empty
-loadedTask: QValueVector<Task>::iterator = NULL
-name: QString = ""
-description: QString = ""
-internalAPI: InternalAPI = NULL
+Study(in id:int=-1)
+~Study()
+setID(in id:int): bool
+getID(): int
+start(in userID:QString): bool
+end(in userID:QString): bool
+pause(in userID:QString): bool
+resume(in userID:QString): bool
+setName(in name:QString): bool
+getTimes(in userID:QString): QValueVector<StudyTaskActivity>
+setDescription(in description:QString): bool
+getName(): QString
+getDescription(): QString
+addTask(in task:Task): bool
+removeTask(in task:Task): bool
+removeTask(in taskID:int): bool
+getTasks(): QValueVector<Task>
+loadTask(in task:Task): bool
+loadTask(in taskID:int): bool
+getLoadedTask(): Task
+insertAction(in action:Action): bool
+getActions(): QValueVector<Action>
+insertComment(in comment:Comment): bool
+getComments(): QValueVector<Comment>
+setInternalAPI(in internalAPI:InternalAPI): bool
*
1
AktionsAPI

Class Description:

This holds the information associated with a study, including a list of the associated Tasks. It is also used to create and manage Tasks, administer a given Study (start, end, pause, resume), and insert Actions and Comments associated with a given Study.

Data Members:

Variable	*Description*
id	An identifying number, generated by the database.
tasks	A list of all tasks associated with the study
loadedTask	Holds a pointer to the task which is currently loaded from the database
name	The title of the usability study.
description	An abstract which describes what the study proposes or hopes to discover.
internalAPI	A pointer to the internal API.

Functions:

Function	*Description*
setID(in ID)	Sets the id variable to ID. Used by the InternalAPI. Returns true if this operation is sucessful.
getID( )	Returns the contents of the id variable. If the Study has not been stored in the database, this function returns -1.
start(in userID)	Begins the study. Sets the start time. Saves the userID of the user who begins the study.
end(in userID)	Ends the study. Sets the stop time. Saves the userID of the user who ends the study.
pause(in userID)	Pauses the study. Sets the pause time of the next free

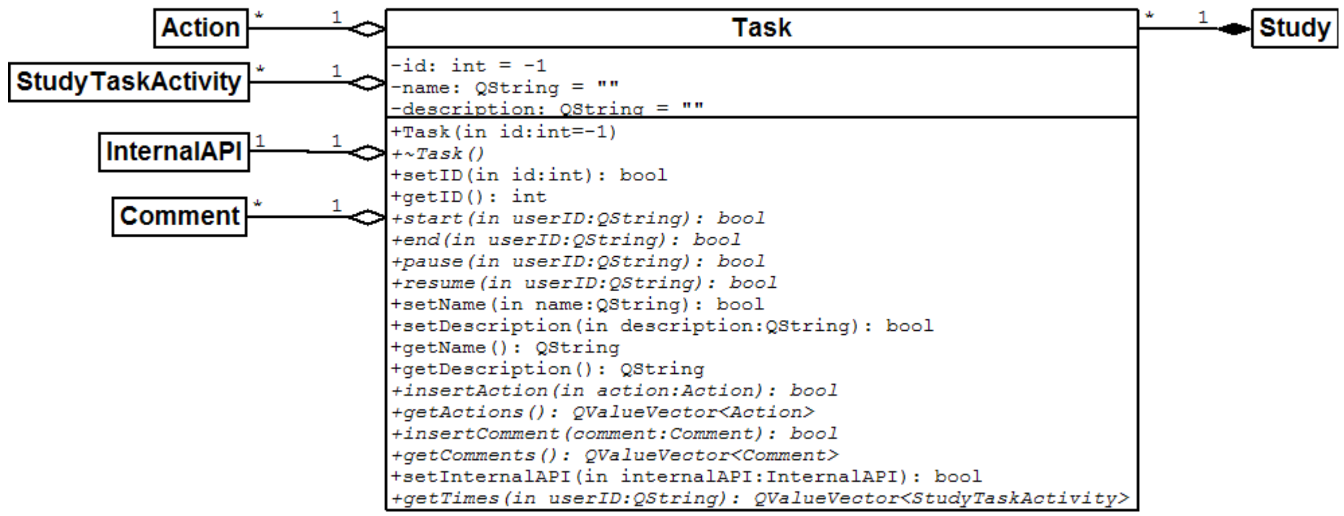
Function	***Description***
	pause/resume ordered pair. If the previous pause/resume pair lacks a resume time, sets the resume time to the pause time. Returns true.
resume(in userID)	Resumes the study. If the study is paused, sets the resume time of the last pause/resume ordered pair. Otherwise does nothing. Returns true.
setName(in name)	Sets the name of the study. Returns true if this operation is sucessful.
getTimes(in userID)	Returns a list of all pause/resume ordered pairs. The first pair in this list represents the start and stop time for the study.
setDescription(in description)	Sets descriptive text for the study. Returns true if this operation is sucessful.
getName( )	Returns the name of the study.
getDecription( )	Returns the descriptive text for the study.
addTask(in task)	Adds a task to the study, if the AktionsAPI is set to admin mode. Returns true if this operation is sucessful.
removeTask(in task)	Removes a task from the study, if the AktionsAPI is set to admin mode. Returns true if this operation is sucessful.
removeTask(in taskID)	Removes the task with TaskID from the study, if the AktionsAPI is set to admin mode. Returns true if this operation is sucessful.
getTasks( )	Returns a list of ID values for all tasks associated with the study.
loadTask(in task)	Loads a particular task from the database into a study.
loadTask(in taskID)	Loads the task with TaskID from the database into a study.
getLoadedTask( )	Returns the currently loaded task.
insertAction(in action)	Inserts an action into the database. Associates the action with the study. Returns true if this operation is sucessful.
getActions( )	Returns a list of all actions associated with the study.
insertComment(in comment)	Inserts a comment associated with this study to Aktions.
getComments( )	Gets a list of all comments associated with this study.
setInternalAPI(in internalAPI)	Sets the instance of internalAPI associated with this instance of Aktions.

Related Objects:

Object	***Relation***
Action	used as a parameter to Study
Comment	used as a parameter to Study
InternalAPI	used for access to database facilities
StudyTaskActivity	used as a parameter in Study

Object	*Relation*
Task	used as a parameter to Study
AktionsAPI	AktionsAPI owns Study

Task


Action
StudyTaskActivity
InternalAPI
Comment
Study
Task
-id: int = -1
-name: QString = ""
-description: QString = ""
+Task(in id:int=-1)
+~Task()
+setID(in id:int): bool
+getID(): int
+start(in userID:QString): bool
+end(in userID:QString): bool
+pause(in userID:QString): bool
+resume(in userID:QString): bool
+setName(in name:QString): bool
+setDescription(in description:QString): bool
+getName(): QString
+getDescription(): QString
+insertAction(in action:Action): bool
+getActions(): QValueVector<Action>
+insertComment(comment:Comment): bool
+getComments(): QValueVector<Comment>
+setInternalAPI(in internalAPI:InternalAPI): bool
+getTimes(in userID:QString): QValueVector<StudyTaskActivity>
*
1

Class Description:

This holds the information associated with a task. It is also used to administer a given Task (start, end, pause, resume) and insert Actions and Comments associated with a given Task.

Data Members:

Variable	*Description*
id	A unique identifier, assigned by the database.
name	The title of the usability study.
description	An abstract which describes what the study proposes or hopes to discover.

Functions:

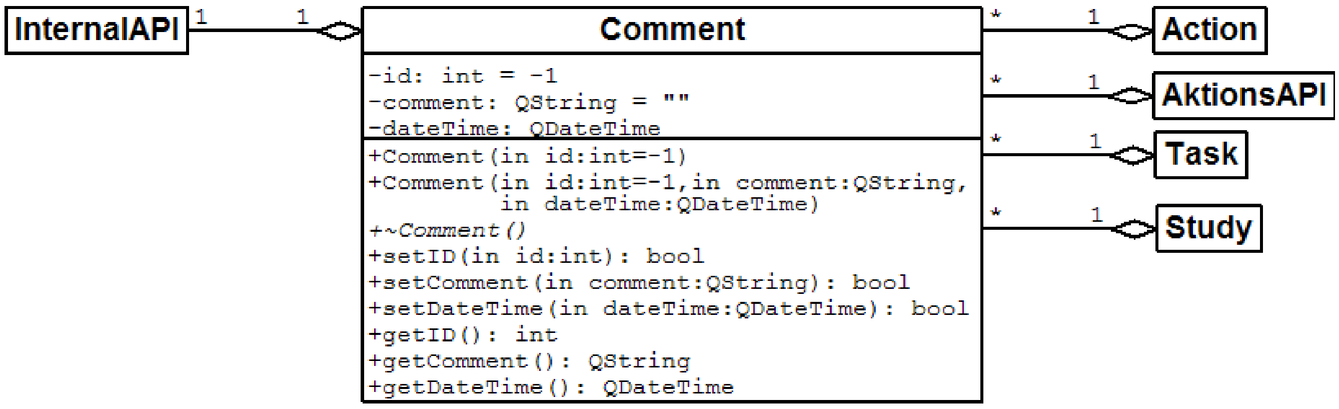
Function	*Description*
setID(in id)	Sets the id variable to ID. Used by the InternalAPI.
getID( )	Returns the contents of the id variable. If the Task has not been stored in the database, this function returns -1.
start(in userID)	Begins the study. Sets the start time. Saves the userID of the user who begins the study.
end(in userID)	Ends the study. Sets the stop time. Saves the userID of the user who ends the study.
pause(in userID)	Pauses the study. Sets the pause time of the next free pause/resume ordered pair. If the previous pause/resume pair lacks a resume time,

Function	***Description***
	sets the resume time to the pause time. Returns true.
resume(in userID)	Resumes the study. If the study is paused, sets the resume time of the last pause/resume ordered pair. Otherwise does nothing. Returns true.
setName(in name)	Sets the name of the task. Returns true if this operation is sucessful.
setDescription(in description)	Sets the descriptive text of the task. Returns true if this operation is sucessful.
getName( )	Returns the name of the task.
getDescription( )	Returns a description of the task.
insertAction(in action)	Inserts an action into the database. Associates the action with the task and the task's study. Returns true if this operation is successful.
getActions( )	Returns a list of all actions associated with the task.
insertComment(in comment)	Inserts a comment associated with the task into the database.
getComments( )	Returns a list of all comments associated with the task.
setInternalAPI(internalAPI)	Sets the task's instance of the internal API object. Returns true if the operation was successful
getTimes(in userID)	Returns a list of all pause and resume times associated with the task. The first element of the list describes the start and stop times for the task.

Related Objects:

Object	***Relation***
Action	used as a parameter to Task
StudyTaskActivity	used as a parameter in Task
InternalAPI	used for access to database facilities
Comment	used as a parameter to Task
Study	Study owns Task

Comment



Class Description:
This holds the information associated with a comment.

Data Members:

Variable	***Description***
id	A unique identifier assigned by the database.
comment	The text associated with this comment.
dateTime	The date and time the comment was created.

Functions:

Function	***Description***
setID(in id)	Sets the id variable to ID. Used by the InternalAPI. Returns true if the operation was successful.
setComment(in comment)	Sets the value of the comment field. Returns true if the operation was successful.
setDateTime(in dateTime)	Sets the value of the dateTime field. Returns true if the operation was successful.
getID( )	Returns the contents of the id variable. If the Comment has not been stored in the database, this function returns -1.
getComment( )	Returns the text of the comment
getDateTime( )	Returns the Date and Time the comment was created.

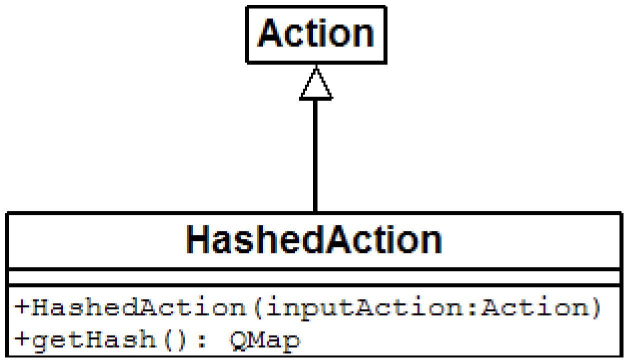
Related Objects:

Object	***Relation***
InternalAPI	used for access to database facilities
Action	Comment passed as parameter to Action
AktionsAPI	Comment passed as parameter to AktionsAPI
Task	Comment passed as parameter to Task

Object	*Relation*
Study	Comment passed as parameter to Study

Internal Definitions

Hashed<Object>s



Class Description:

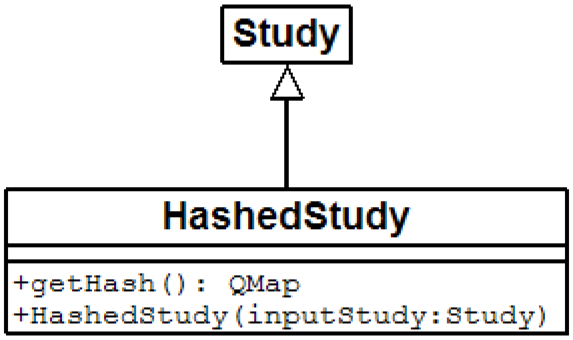
Provides an abstraction to the fields available in an Action that the database can use in a simple way. A hash accomplishes this as field names are a type of key and field values are a type of value that a key can retrieve, and it uses the field names that the database uses rather than their internal field names.

Functions:

Function	*Description*
getHash( )	Returns a hash that uniquely identifies an Action.

Related Objects:

Object	*Relation*
Action	passed as a parameter during construction of a HashedAction, and acts as a copy constructor, generating a hashed object from any actions object. It is a child of actions so it can provide copy construction and have access to protected members if needed



Class Description:

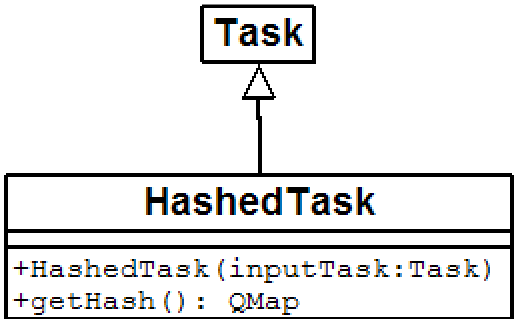
Provides an abstraction to the fields available in a Study that the database can use in a simple way. A hash accomplishes this as field names are a type of key and field values are a type of value that a key can retrieve, and it uses the field names that the database uses rather than their internal field names.

Functions:

Function	***Description***
getHash( )	Returns a hash that uniquely identifies a study.

Related Objects:

Object	***Relation***
Study	Studies are passed into HasedStudies as a type of copy constructor, and generate a hashed version of the fields on construction. It is a child of study so it can provide copy construction and have access to protected members if needed.



Class Description:

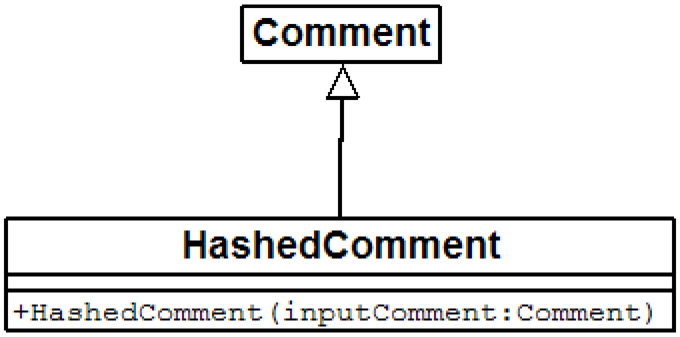
Provides an abstraction to the fields available in a Task that the database can use in a simple way. A hash accomplishes this as field names are a type of key and field values are a type of value that a key can retrieve, and it uses the field names that the database uses rather than their internal field names.

Functions:

Function	***Description***
getHash( )	Returns a hash that uniquely identifies a task.

Related Objects:

Object	*Relation*
Task	Tasks are passed into HasedTask as a type of copy constructor, and generate a hashed version of the fields on construction. It is a child of Task so it can provide copy construction and have access to protected members if needed.



Class Description:

Provides an abstraction to the fields available in a Comment that the database can use in a simple way. A hash accomplishes this as field names are a type of key and field values are a type of value that a key can retrieve, and it uses the field names that the database uses rather than their internal field names.

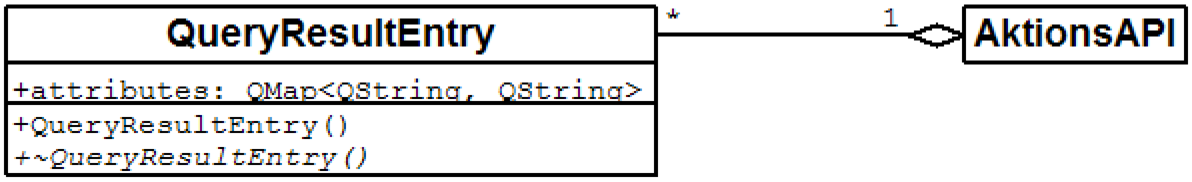
Functions:

Function	*Description*
getHash( )	Returns a hash that uniquely identifies a comment.

Related Objects:

Object	*Relation*
Comment	Comments are passed into HasedComment as a type of copy constructor, and generate a hashed version of the fields on construction. It is a child of Comment so it can provide copy construction and have access to protected members if needed.

QueryResultEntry



Class Description:

The QueryResultEntry stores the results of arbitrary SQL queries made against the Aktions database. Each QueryResultEntry stores one row of the resultant table.

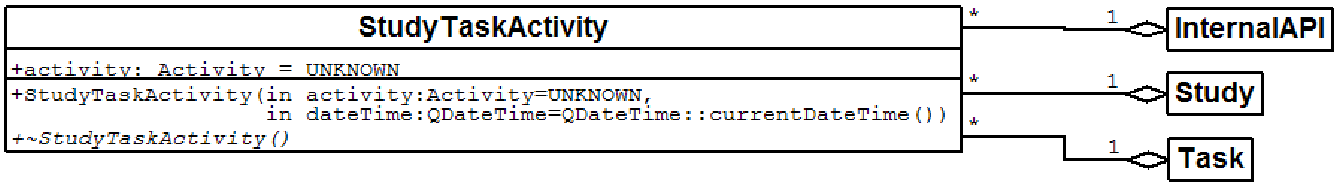
Data Members:

Variable	*Description*
attributes	A map whose keys represent the attributes of a table and whose elements represent the value stored within the table.

Related Objects:

Object	*Relation*
AktionsAPI	QueryResultEntry passed as a parameter to AktionsAPI

StudyTaskActivity


StudyTaskActivity
+activity: Activity = UNKNOWN
+StudyTaskActivity(in activity:Activity=UNKNOWN,
in dateTime:QDateTime=QDateTime::currentDateTime())
+~StudyTaskActivity()
*
1
InternalAPI
*
1
Study
*
1
Task

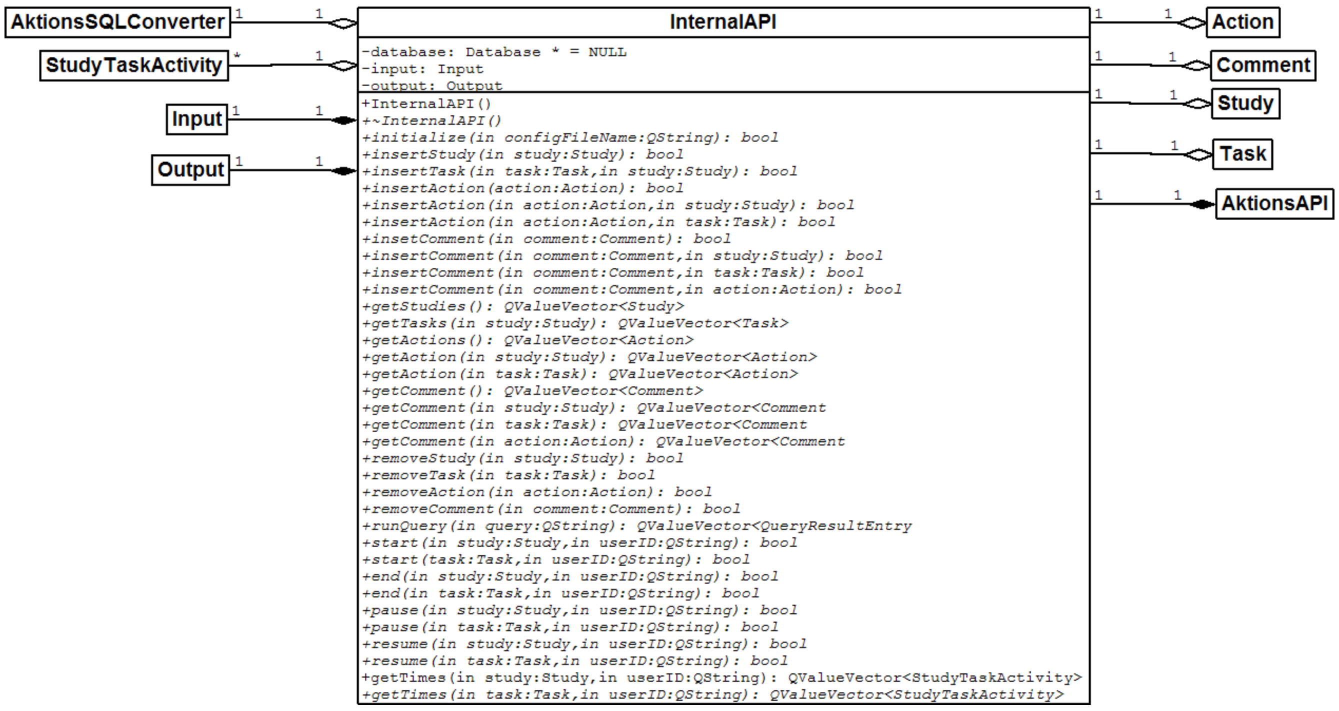
Class Description:
This holds the information associated with the administration of a Task or a Study. It groups the activity performed and a timestamp of when it was performed.
Data Members:

Variable	*Description*
activity	Stores a pair of times. Those pairs represent pause/resume or start/stop times dependent upon context.

Related Objects:

Object	*Relation*
InternalAPI	StudyTaskActivity passed as parameter to InternalAPI
Study	StudyTaskActivity passed as parameter to Study
Task	StudyTaskActivity passed as parameter to Task

InternalAPI


AktionsSQLConverter
StudyTaskActivity
Input
Output
Action
Comment
Study
Task
AktionsAPI
1
*
InternalAPI
-database: Database * = NULL
-input: Input
-output: Output
+InternalAPI()
+~InternalAPI()
+initialize(in configFileName:QString): bool
+insertStudy(in study:Study): bool
+insertTask(in task:Task,in study:Study): bool
+insertAction(action:Action): bool
+insertAction(in action:Action,in study:Study): bool
+insertAction(in action:Action,in task:Task): bool
+insetComment(in comment:Comment): bool
+insertComment(in comment:Comment,in study:Study): bool
+insertComment(in comment:Comment,in task:Task): bool
+insertComment(in comment:Comment,in action:Action): bool
+getStudies(): QValueVector<Study>
+getTasks(in study:Study): QValueVector<Task>
+getActions(): QValueVector<Action>
+getAction(in study:Study): QValueVector<Action>
+getAction(in task:Task): QValueVector<Action>
+getComment(): QValueVector<Comment>
+getComment(in study:Study): QValueVector<Comment
+getComment(in task:Task): QValueVector<Comment
+getComment(in action:Action): QValueVector<Comment
+removeStudy(in study:Study): bool
+removeTask(in task:Task): bool
+removeAction(in action:Action): bool
+removeComment(in comment:Comment): bool
+runQuery(in query:QString): QValueVector<QueryResultEntry
+start(in study:Study,in userID:QString): bool
+start(task:Task,in userID:QString): bool
+end(in study:Study,in userID:QString): bool
+end(in task:Task,in userID:QString): bool
+pause(in study:Study,in userID:QString): bool
+pause(in task:Task,in userID:QString): bool
+resume(in study:Study,in userID:QString): bool
+resume(in task:Task,in userID:QString): bool
+getTimes(in study:Study,in userID:QString): QValueVector<StudyTaskActivity>
+getTimes(in task:Task,in userID:QString): QValueVector<StudyTaskActivity>

Class Description:

This is used as a wrapper for the internal parts of the API. It was added to facilitate parallel development in the absence of functional stubs. The Aktions project plans to remove this object during spiral 3.

Data Members:

Variable	***Description***
database	An object which represents the database where Aktions stores its information.
input	An object used to insert information into the database
output	An object used to retrieve information from the database.

Functions:

Function	***Description***
initialize(in configFileName)	Loads and applies the configuration for Aktions from file ConfigFileName. ConfigFileName is an absolute location. Returns true if Aktions initialized properly.
insertStudy(in study)	Instructs the Aktions storage mechanisms to insert the given study into the database. Returns true if the operation is successful.
insertTask(in task)	Instructs the Aktions storage mechanisms to insert the

Function	*Description*
	given task into the database. Returns true if the operation is successful.
insertAction(in action)	Instructs the Akions storage mechanisms to insert the given action into the database. Returns true if the operation is successful.
insertAction(in action, in study)	Instructs the Aktions storage mechanisms to insert the given action into the database, and associate it with the given study. Returns true if the operation is successful.
insertAction(in action, in task)	Instructs the Aktions storage mechanisms to insert the given action into the database, and associate it with the given task. Returns true if the operation is successful.
insertComment(in comment)	Instructs the Aktions storage mechanisms to insert the given comment into the database. Returns true if the operation is successful.
insertComment(in comment, in study)	Instructs the Aktions storage mechanisms to insert the given comment into the database, and associate it with the given study. Returns true if the operation is successful.
insertComment(in comment, in task)	Instructs the Aktions storage mechanisms to insert the given comment into the database, and associate it with the given task. Returns true if the operation is successful.
insertComment(in comment, in action)	Instructs the Aktions storage mechanisms to insert the given comment into the database, and associate it with the given action. Returns true if the operation is successful.
getStudies( )	Instructs the Aktions storage mechanisms to retrieve a list of all studies within the database. Returns the results if the operation is successful.
getTasks(in study)	Instructs the Aktions storage mechanisms to retrieve a list of all tasks within the database that are associated with the given study. Returns the results if the operation is successful.
getActions( )	Instructs the Aktions storage mechanisms to retrieve a list of all unassociated actions within the database. Returns the results if the operation is successful.
getActions(in study)	Instructs the Aktions storage mechanisms to retrieve a list of all actions within the database that are associated with the given study. Returns the results if the operation is successful.
getActions(in task)	Instructs the Aktions storage mechanisms to retrieve a list of all actions within the database that are associated with the given task. Returns the results if the operation is successful.
getComments( )	Instructs the Aktions storage mechanisms to retrieve a list

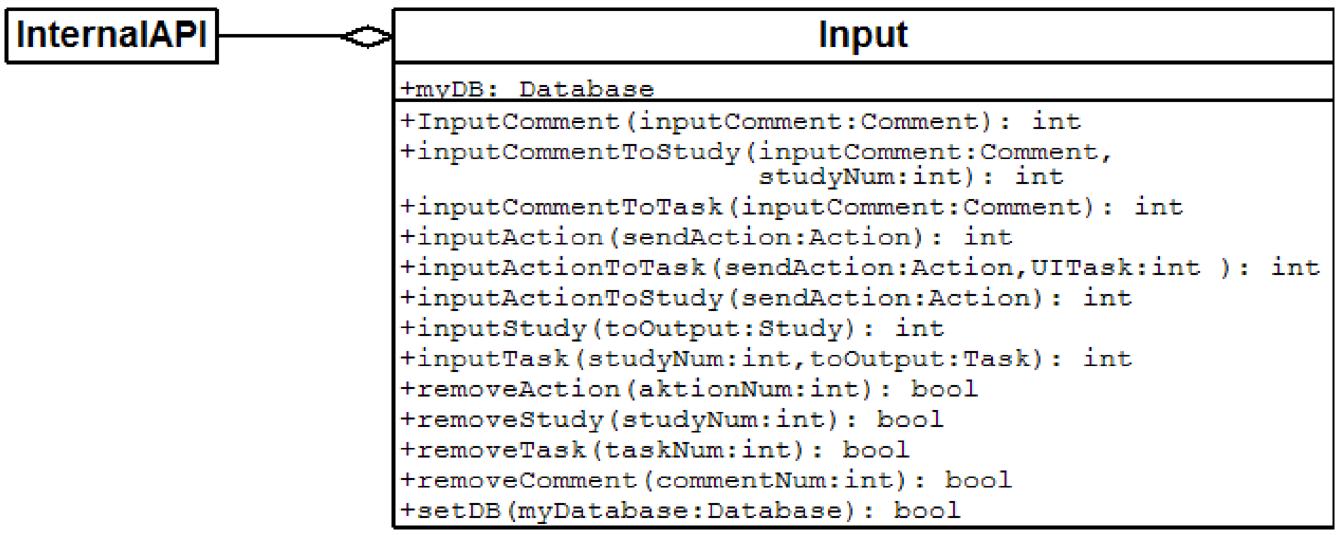
Function	***Description***
	of all unassociated comments within the database. Returns the results if the operation is successful.
getComments(in study)	Instructs the Aktions storage mechanisms to retrieve a list of all comments within the database that are associated with the given study. Returns the results if the operation is successful.
getComments(in task)	Instructs the Aktions storage mechanisms to retrieve a list of all comments within the database that are associated with the given task. Returns the results if the operation is successful.
getComments(in action)	Instructs the Aktions storage mechanisms to retrieve a list of all comments within the database that are associated with the given action.
removeStudy(in study)	Instructs the Aktions storage mechanisms to remove the given study. Returns true if the operation is successful.
removeTask(in task)	Instructs the Aktions storage mechanisms to remove the given task. Returns true if the operation is successful.
removeAction(in action)	Instructs the Aktions storage mechanisms to remove the given action. Returns true if the operation is successful.
removeComment(in comment)	Instructs the Aktions storage mechanisms to remove the given comment. Returns true if the operation is successful.
runQuery(in query)	Instructs the Aktions storage mechanisms to run the given query. Returns the results if the operation is successful.
start(in study, in userID)	Instructs the Aktions storage mechanism to begin the given study. It will store the start time in the database, along with the userID. Returns true if the operation is successful.
start(in task, in userID)	Instructs the Aktions storage mechanism to begin the given task. It will store the start time in the database, along with the userID. Returns true if the operation is successful.
end(in study, in userID)	Instructs the Aktions storage mechanism to end the given study. It will store the start time in the database, along with the userID. Returns true if the operation is successful.
end(in task, in userID)	Instructs the Aktions storage mechanism to end the given task. It will store the start time in the database, along with the userID. Returns true if the operation is successful.
pause(in study, in userID)	Instructs the Aktions storage mechanism to pause the given study. It will store the start time in the database, along with the userID. Returns true if the operation is

Function	***Description***
	successful.
pause(in task, in userID)	Instructs the Aktions storage mechanism to pause the given task. It will store the start time in the database, along with the userID. Returns true if the operation is successful.
resume(in study, in userID)	Instructs the Aktions storage mechanism to resume the given study. It will store the start time in the database, along with the userID. Returns true if the operation is successful.
resume(in task, in userID)	Instructs the Aktions storage mechanism to begin the given task. It will store the start time in the database, along with the userID. Returns true if the operation is successful.
getTimes(in study, in userID)	Instructs the Aktions storage mechanism to retrieve a list of all times associated with the given study and userID. Returns the results if the operation is successful.
getTimes(in task, in userID)	Instructs the Aktions storage mechanism to retrieve a list of all times associated with the given task and userID. Returns the results if the operation is successful.

Related Objects:

Object	***Relation***
AktionsSQLConverter	Owns this agent to accomplish the task of creating SQL that can be directly executed by the database
StudyTaskActivity	passed as a parameter to InternalAPI
Input	Owned by InternalAPI in a one-to-one relationship, as it provides the core functionality related to inputting things to the database.
Output	Owned by InternalAPI in a one-to-one relationship, as it provides the core functionality related to outputting things from the database
Action	passed as a parameter to InternalAPI
Comment	passed as a parameter to InternalAPI
Study	passed as a parameter to InternalAPI
Task	passed as a parameter to InternalAPI
AktionsAPI	InternalAPI owned by AktionsAPI

Input



```
+myDB: Database
+InputComment(inputComment:Comment): int
+inputCommentToStudy(inputComment:Comment,
                     studyNum:int): int
+inputCommentToTask(inputComment:Comment): int
+inputAction(sendAction:Action): int
+inputActionToTask(sendAction:Action,UITask:int ): int
+inputActionToStudy(sendAction:Action): int
+inputStudy(toOutput:Study): int
+inputTask(studyNum:int,toOutput:Task): int
+removeAction(aktionNum:int): bool
+removeStudy(studyNum:int): bool
+removeTask(taskNum:int): bool
+removeComment(commentNum:int): bool
+setDB(myDatabase:Database): bool
```

Class Description:
Performs all tasks related to inserting Aktions-specific objects to the database.
Data Members:

Variable	***Description***
myDB	stores a link to the database object used to store and retrieve information.

Functions:

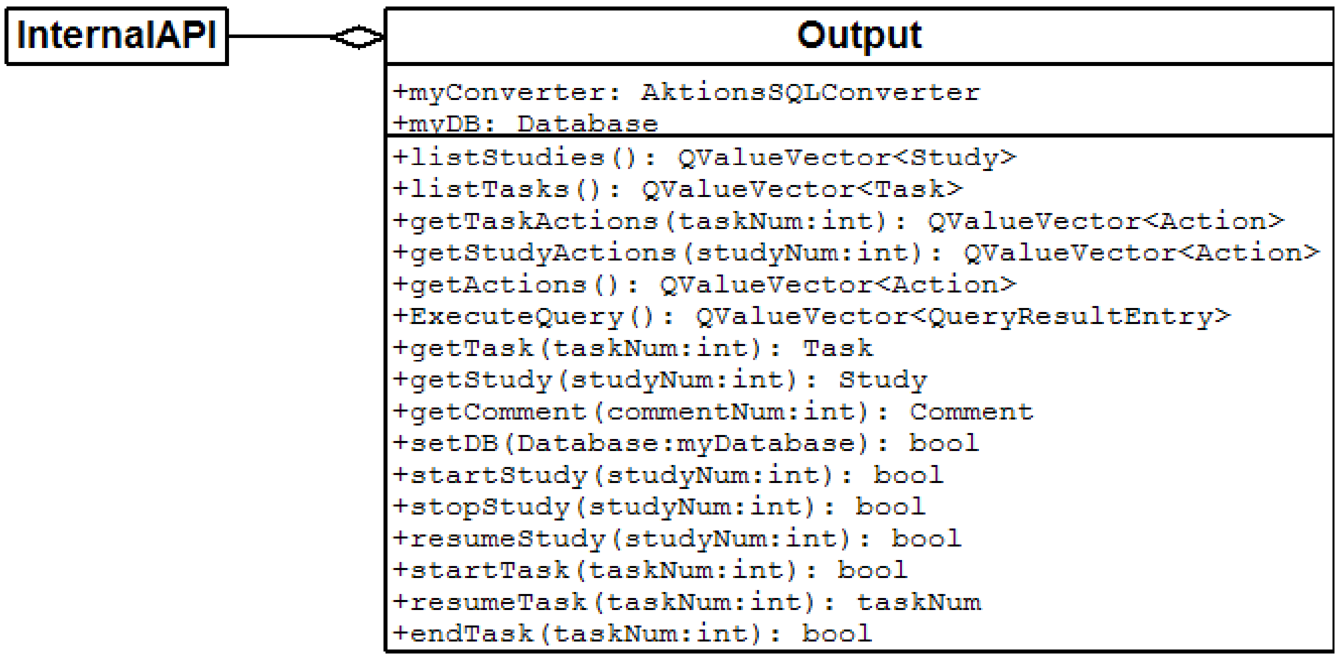
Function	***Description***
InputComment(in inputComment)	Inserts a comment into the database.
inputCommentToStudy(in inputComment, in studyNum)	Associates a comment with studyNum. Inserts it into the database.
inputCommentToTask(in inputComment)	Associates a comment with the currently running task. Inserts it into the database.
inputAction(in sendAction)	Inserts an action into the database.
inputActionToTask(in sendAction, in UITask)	Associates an action with a task. Inserts the action into the database.
inputActionToStudy(in sendAction)	Associates an action with a study. Inserts the action into the database.
inputStudy(in toInput)	Inserts a study into the database.
inputTask(in studyNum, in toInput)	Associates the task with a study. Inserts the task into the database.
removeAction(in actionNum)	Removes an action from the database. Returns true if the operation was successful.
removeStudy(in studyNum)	Removes a study from the database. Returns true if the operation was successful.

Function	***Description***
removeTask(in taskNum)	Removes a task from the database. Returns true if the operation was successful.
removeComment(in commentNum)	Removes a comment from the database. Returns true if the operation was successful.
setDB(in myDatabase)	Sets the database object for the input object.

Related Objects:

Object	***Relation***
InternalAPI	Input is used by InternalAPI to accomplish input tasks

Output



```
+myConverter: AktionsSQLConverter
+myDB: Database
+listStudies(): QValueVector<Study>
+listTasks(): QValueVector<Task>
+getTaskActions(taskNum:int): QValueVector<Action>
+getStudyActions(studyNum:int): QValueVector<Action>
+getActions(): QValueVector<Action>
+ExecuteQuery(): QValueVector<QueryResultEntry>
+getTask(taskNum:int): Task
+getStudy(studyNum:int): Study
+getComment(commentNum:int): Comment
+setDB(Database:myDatabase): bool
+startStudy(studyNum:int): bool
+stopStudy(studyNum:int): bool
+resumeStudy(studyNum:int): bool
+startTask(taskNum:int): bool
+resumeTask(taskNum:int): taskNum
+endTask(taskNum:int): bool
```

Class Description:

Performs all operations related to retrieving Aktions-specific objects from the database.

Data Members:

Variable	***Description***
myConverter	Stores a converter object to aid transitions between different database schemas.
myDB	stores the database to be queried.

Functions:

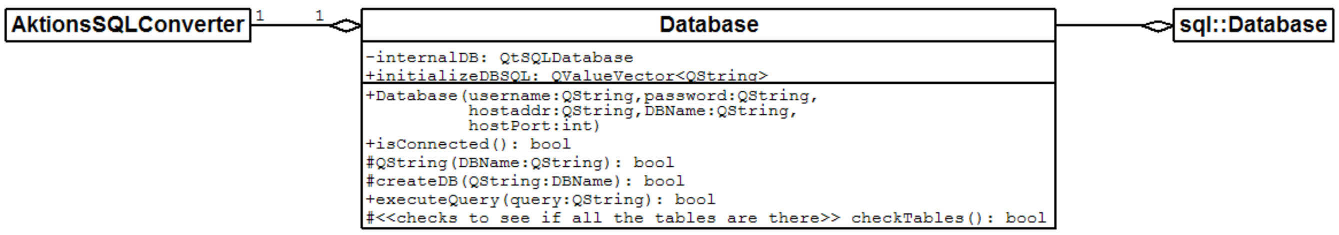
Function	***Description***
listStudies( )	Returns a list of all studies stored by the database.

Function	*Description*
listTasks( )	Returns a list of all tasks stored by the database.
getTaskActions(in taskNum)	Returns a list of all actions associated with the task that are stored in the database.
getStudyActions(in studyNum)	Returns a list of all actions associated with the study that are stored in the database.
getActions( )	Returns a list of all actions not associated with a study or task that are stored in the database.
executeQuery( )	Executes an arbitrary query on the database.
getTask(in taskNum)	Returns the details of a given task object.
getStudy(in studyNum)	Returns the details of a given study object.
getComment(in commentNum)	Returns the details of a given comment object.
setDB(in database)	Sets the database these operations will be performed on.
startStudy(in studyNum)	Sets the start time for a study within the database.
stopStudy(in studyNum)	Sets the stop time for a study within the database.
resumeStudy(in studyNum)	Sets the resume time for a study within the database.
startTask(in taskNum)	Sets the start time for a task within the database.
resumeTask(in taskNum)	Sets the resume time for a task within the database.
endTask(in taskNum)	Sets the end time for a task within the database.

Related Objects:

Object	*Relation*
InternalAPI	Output is used by InternalAPI to accomplish input tasks

Database


AktionsSQLConverter
1
1
Database
-internalDB: QtSQLDatabase
+initializeDBSQL: QValueVector<QString>
+Database(username:QString,password:QString,
hostaddr:QString,DBName:QString,
hostPort:int)
+isConnected(): bool
#QString(DBName:QString): bool
#createDB(QString:DBName): bool
+executeQuery(query:QString): bool
#<<checks to see if all the tables are there>> checkTables(): bool
sql::Database

Class Description:

Provides a facade for database interactivity, defining a simple interface for a complex Database object.

Data Members:

Variable	***Description***
internalDB	Stores the interface provided by the MySQL libraries.
initializeDBSQL	Stores the SQL statement necessary to install the Aktions database.

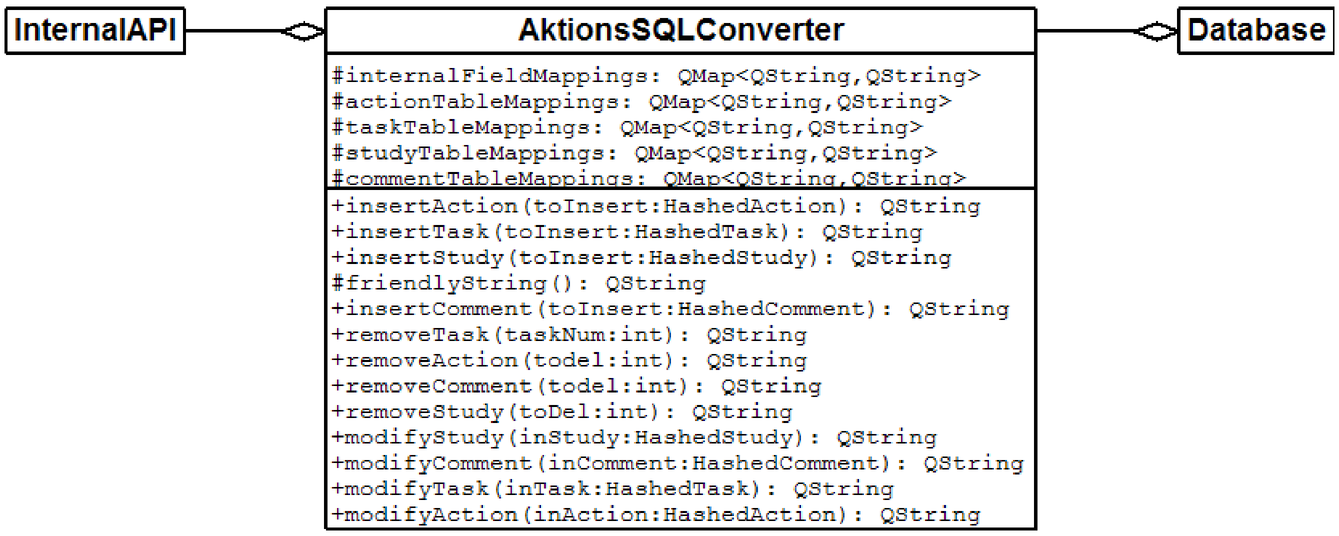
Functions:

Function	***Description***
isConnected( )	Returns true if Aktions is currently connected to a database.
setDB(in DBName)	Sets the database to connect to.
createDB(in DBName)	Creates the database if one does not exist.
executeQuery(in query)	Executes a SQL query.
checkTables( )	returns true if the database contains the tables that Aktions needs

Related Objects:

Object	***Relation***
AktionsSQLConverter	uses this to convert any raw objects into SQL appropriate functions and convert any SQL incomputable string elements when encountered.
sql::Database	the native Database object used in the native database library, mysql++

AktionsSQLConverter


InternalAPI
AktionsSQLConverter
Database
#internalFieldMappings: QMap<QString,QString>
#actionTableMappings: QMap<QString,QString>
#taskTableMappings: QMap<QString,QString>
#studyTableMappings: QMap<QString,QString>
#commentTableMappings: QMap<QString,QString>
+insertAction(toInsert:HashedAction): QString
+insertTask(toInsert:HashedTask): QString
+insertStudy(toInsert:HashedStudy): QString
#friendlyString(): QString
+insertComment(toInsert:HashedComment): QString
+removeTask(taskNum:int): QString
+removeAction(todel:int): QString
+removeComment(todel:int): QString
+removeStudy(toDel:int): QString
+modifyStudy(inStudy:HashedStudy): QString
+modifyComment(inComment:HashedComment): QString
+modifyTask(inTask:HashedTask): QString
+modifyAction(inAction:HashedAction): QString

Class Description:
Creates SQL from Aktions specific objects
Data Members:

Variable	***Description***
internalFieldMappings	Maps keys describing prorperties of an object to the appropriate table in the database.
actionTableMappings	maps HashedAction keys to the appropriate table in the database.
taskTableMappings	maps HashedTask keys to the appropriate table in the database.
studyTableMappings	maps HashedStudy keys to the appropriate table in the database.
commentTableMappings	maps HashedComment keys to the appropriate table in the database.

Functions:

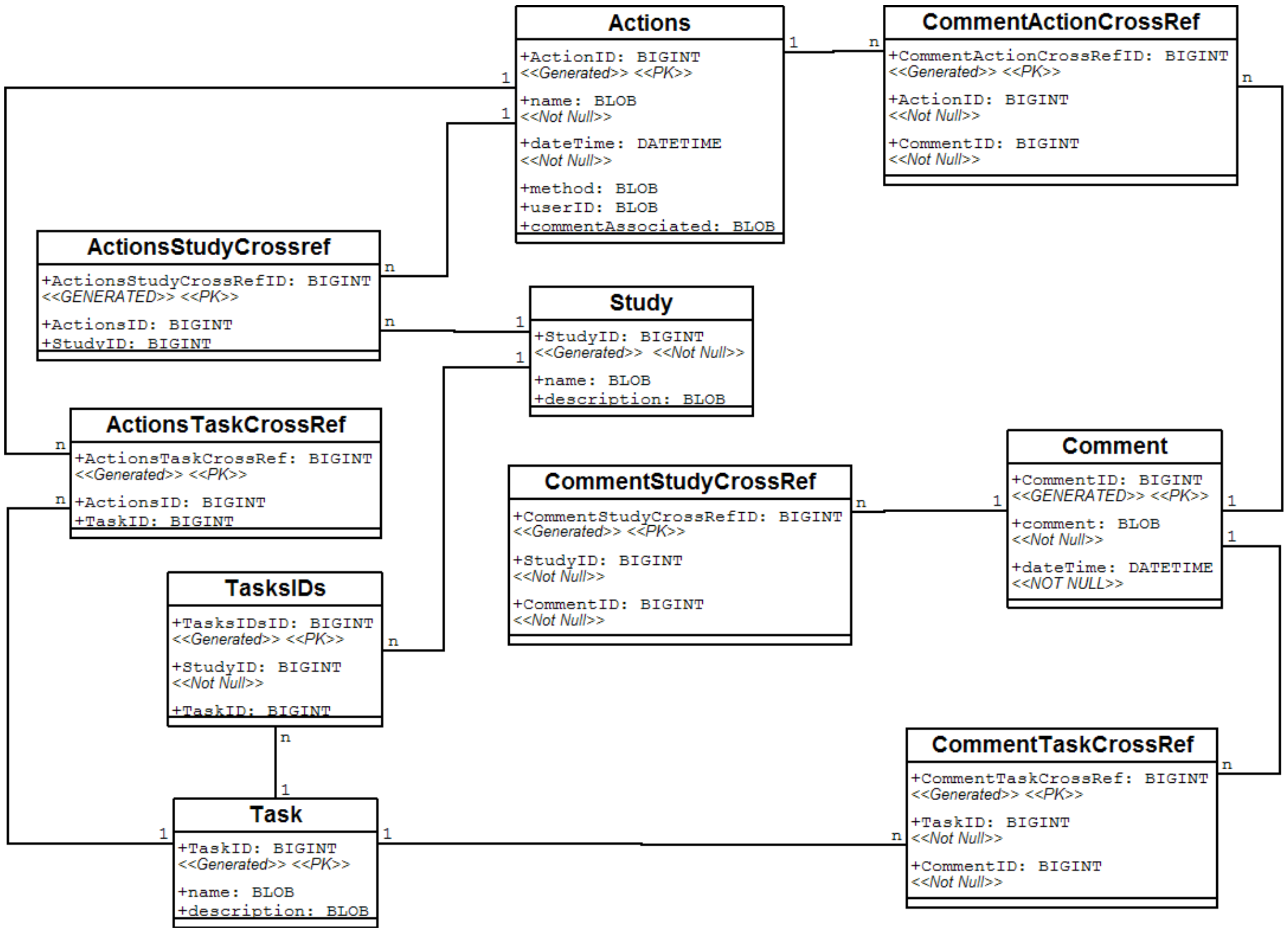
Function	***Description***
listStudies( )	Returns the SQL required to query for a list of studies.
listTasks( )	Returns the SQL required to query for a list of tasks.
getTaskActions(in taskNum)	Returns the SQL required to query for a list of actions associated with a task.
getStudyActions(in studyNum)	Returns the SQL required to query for a list of actions associated with a study.
getActions( )	Returns the SQL required to query for a list of actions unassociated with any task or study.
getTask(in TaskNum)	Returns the SQL required to query for the details of a given task.
getStudy(in studyNum)	Returns the SQL required to query for the details of a given study.
getComment(in commentNum)	Returns the SQL required to query for the details of a given

Function	***Description***
	comment.

Related Objects:

Object	***Relation***
InternalAPI	Is used by InternalAPI

Database Defintion



Actions
+ActionID: BIGINT <<Generated>> <<PK>>
+name: BLOB <<Not Null>>
+dateTime: DATETIME <<Not Null>>
+method: BLOB
+userID: BLOB
+commentAssociated: BLOB
CommentActionCrossRef
+CommentActionCrossRefID: BIGINT <<Generated>> <<PK>>
+ActionID: BIGINT <<Not Null>>
+CommentID: BIGINT <<Not Null>>
ActionsStudyCrossref
+ActionsStudyCrossRefID: BIGINT <<GENERATED>> <<PK>>
+ActionsID: BIGINT
+StudyID: BIGINT
Study
+StudyID: BIGINT <<Generated>> <<Not Null>>
+name: BLOB
+description: BLOB
ActionsTaskCrossRef
+ActionsTaskCrossRef: BIGINT <<Generated>> <<PK>>
+ActionsID: BIGINT
+TaskID: BIGINT
Comment
+CommentID: BIGINT <<GENERATED>> <<PK>>
+comment: BLOB <<Not Null>>
+dateTime: DATETIME <<NOT NULL>>
CommentStudyCrossRef
+CommentStudyCrossRefID: BIGINT <<Generated>> <<PK>>
+StudyID: BIGINT <<Not Null>>
+CommentID: BIGINT <<Not Null>>
TasksIDs
+TasksIDsID: BIGINT <<Generated>> <<PK>>
+StudyID: BIGINT <<Not Null>>
+TaskID: BIGINT
Task
+TaskID: BIGINT <<Generated>> <<PK>>
+name: BLOB
+description: BLOB
CommentTaskCrossRef
+CommentTaskCrossRef: BIGINT <<Generated>> <<PK>>
+TaskID: BIGINT <<Not Null>>
+CommentID: BIGINT <<Not Null>>
1
n

KDE Hooks Definitions

All objects contained within the kdeui library are candidates for instrumentation by Aktions. For more information on these objects, visit http://developer.kde.org/documentation/library/3.0-api/classref/kdeui/index.html. The precise set will be determined via discussions with the KDE core developers, usability experts, and the Aktions team.

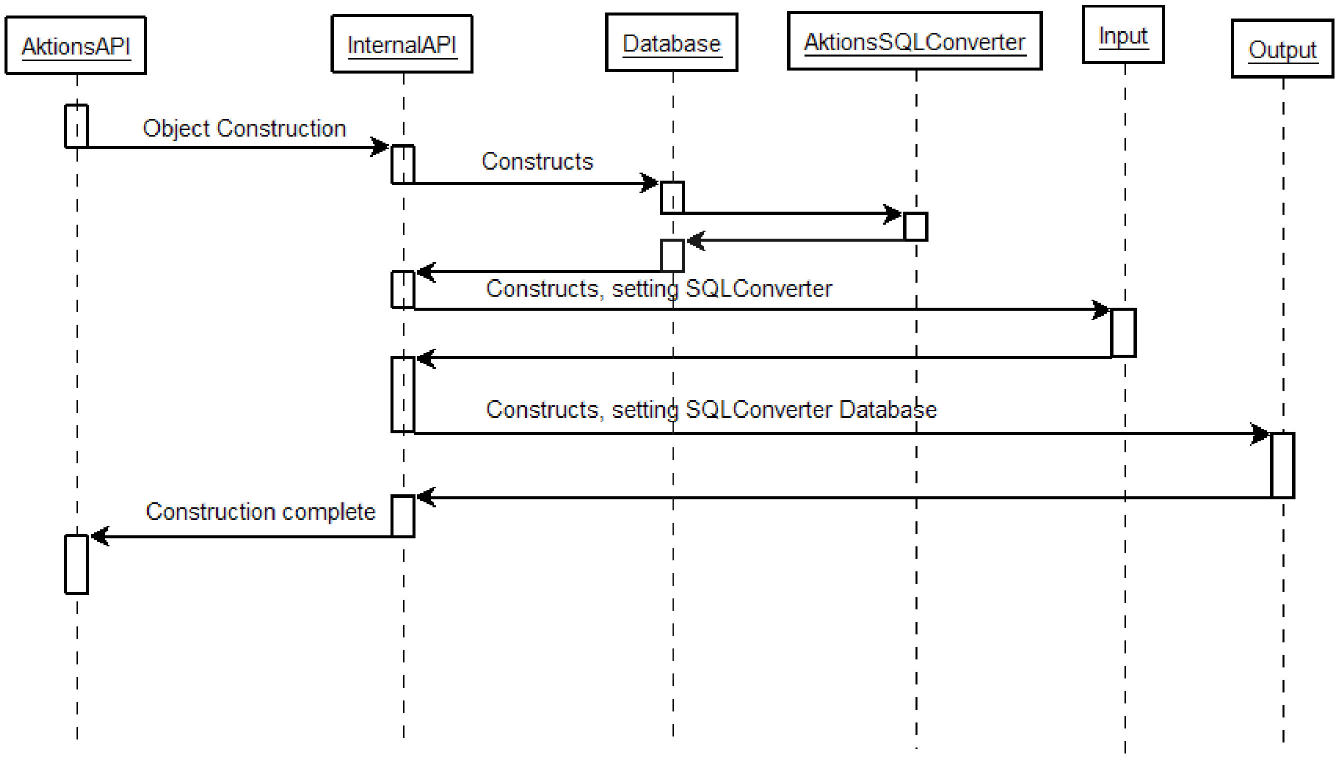
Due to the sensitive nature of the KDE core libraries, instrumentation for every object except the

KAction has been deferred to the next spiral. More information on the KAction is available at http://developer.kde.org/documentation/library/3.0-api/classref/kdeui/KAction.html.

The instrumentation of KActions occurs within the `void activate()` method. It will be necessary for developers wishing to use Aktions to call KAction::activate() if they overload this function.

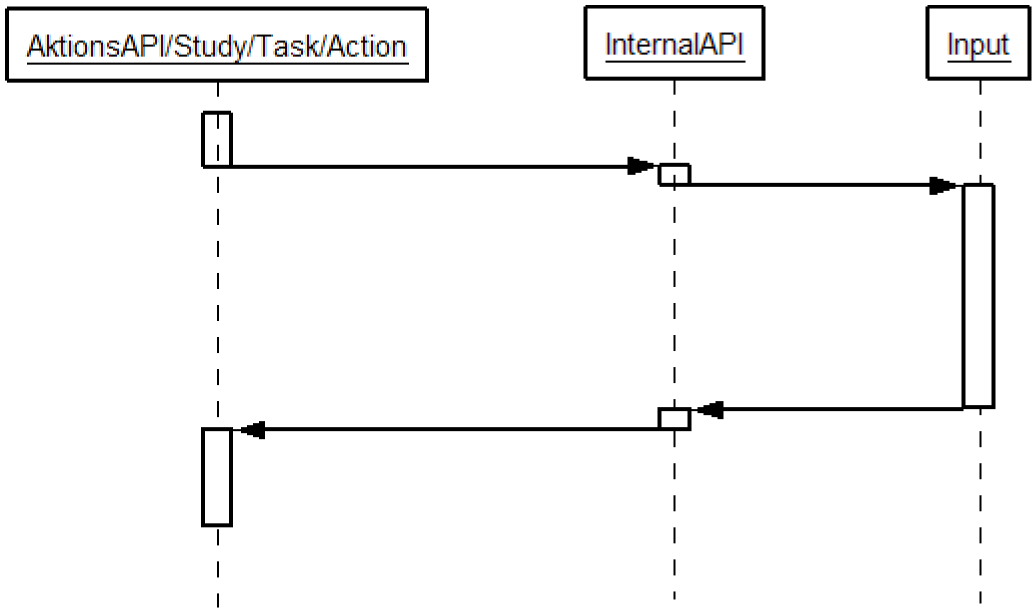
API Procedures

Initilalization



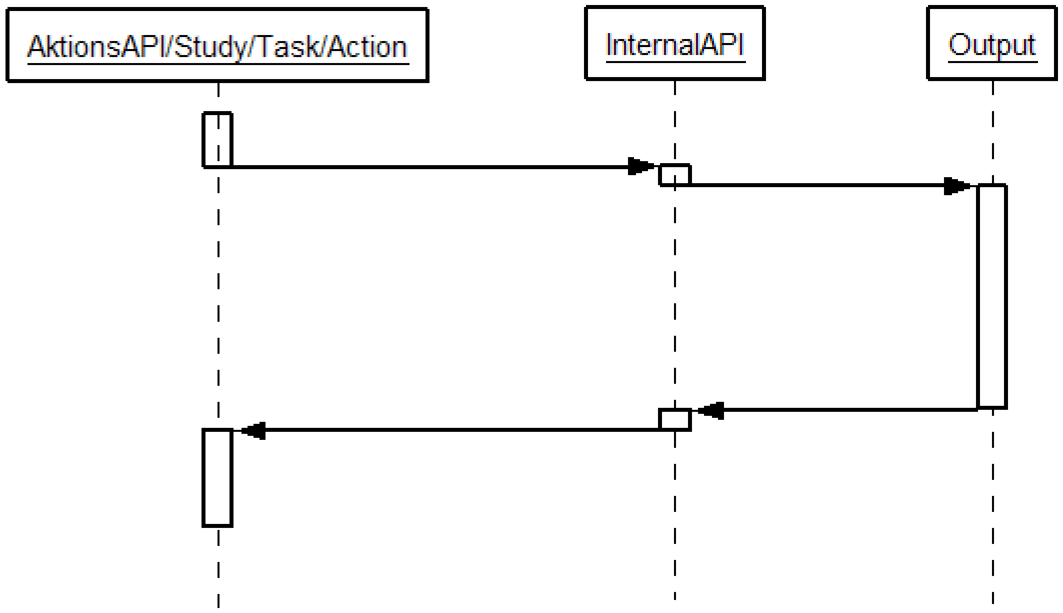
Initialization is the most complicated portion of the API. Due to the sheer volume of setup required before data can be collected, this call reaches deeply into the bowels of Aktions. Before the Aktions library is used, the Initialize() function must be called. This function creates the Internal API, which in turn verifies that the database exists, then creates objects to read and write to it. The internal API serves as a factory class, creating the Database, AktionsSQLConverter, Input, and Output helper objects.

Input Sequence



The input sequence is used to send information from KDE to the database.

Output Sequence



The input sequence is used to retrieve information from the database.