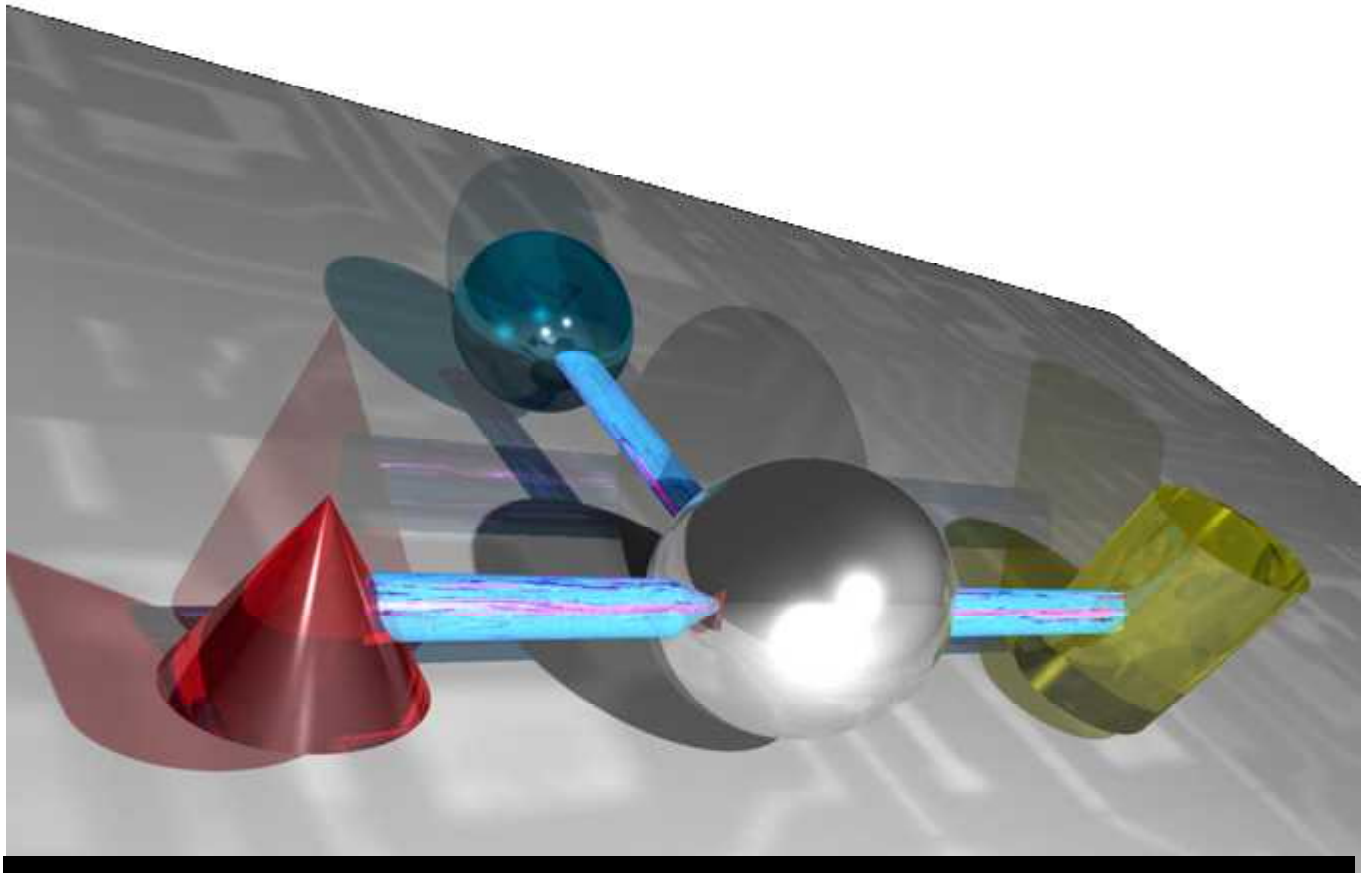


Kframework Reference Manual

Draft @50%

KFramework 1.4

KF Reference Manual 1.6 DRAFT 50%.doc

sourceforge

Contents

KFramework

Please forward all comments and errata to:

Alejandro Vazquez
avazqueznj@users.sourceforge.net
http://k-framework.sourceforge.net

Design
Alejandro Vazquez
Ke Li

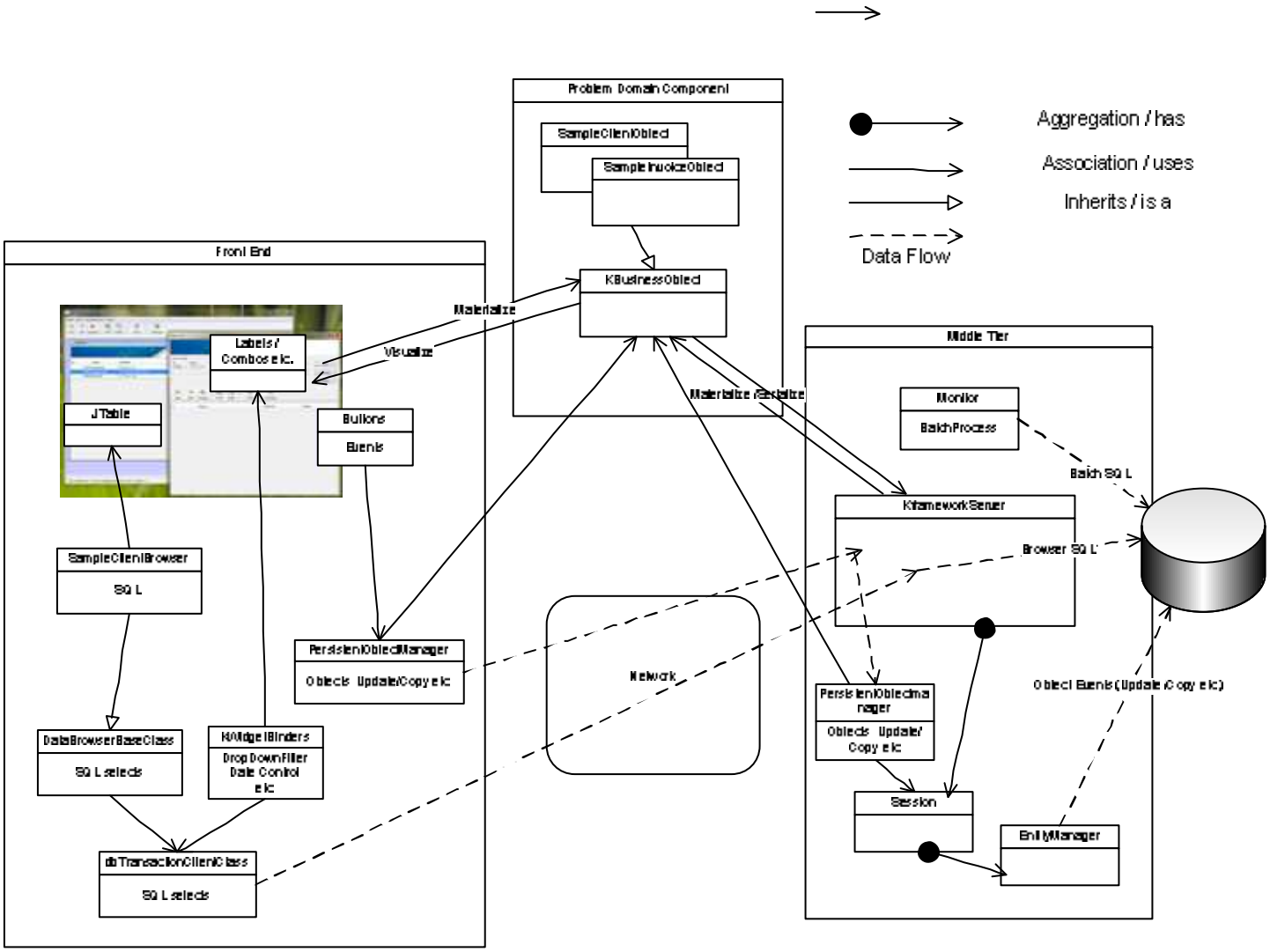
Programming
Alejandro Vazquez
Ke Li
Bahaven Patel
Chris Backas
Maricela Islas
Xavier Gonzalez

Manual Revision history:

Version 1

1. 31/ene/2011 First prerelease by Alejandro Vazquez

1. General Map



2. Back End Reference

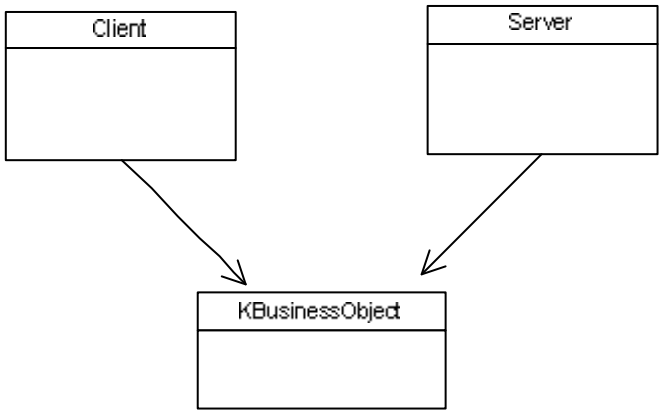
2.1. Defining the Problem Domain Component, The KBusinessObjectClass

The problem domain it the most important part of the system. This Doman Driven approach helps ensure the correctness of the system, facilitates its enhancement, and reduces costs by ensuring that the delivered solution always conforms to the business requirements.

This single key feature is what makes KFramework based projects successful.

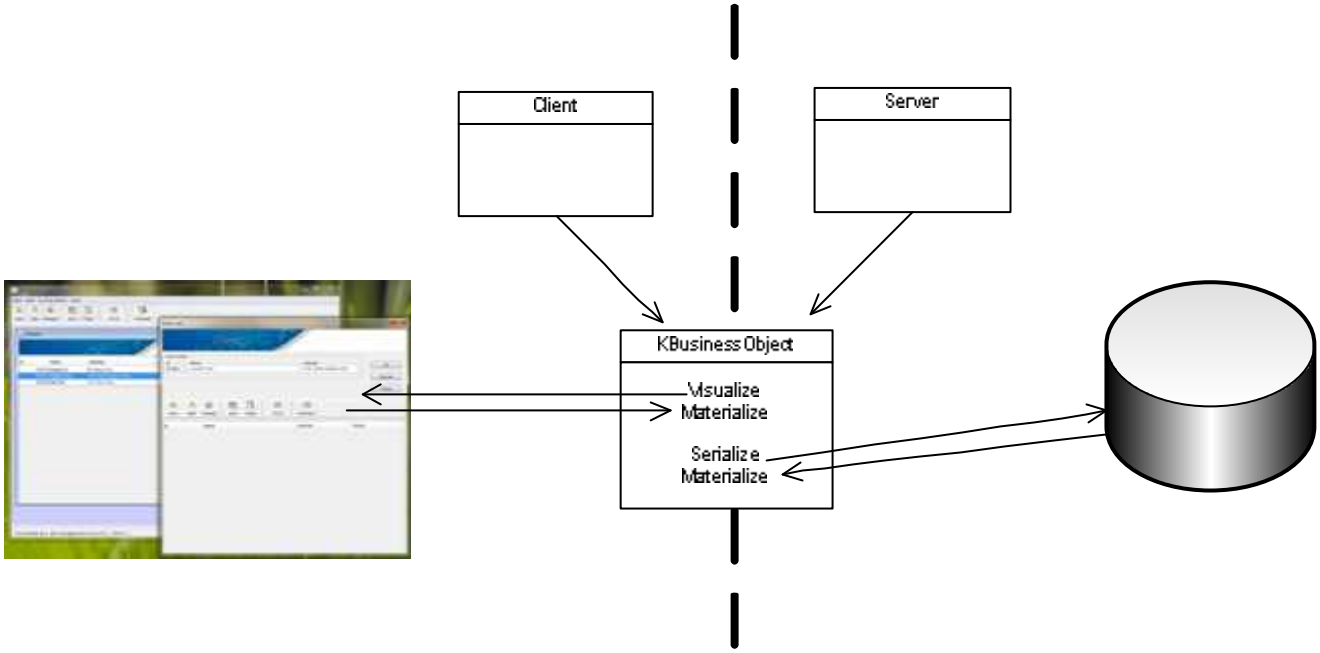
The framework is designed to help you maintain focus on business entities, relationships and data flows, instead of implementation artifacts, but discipline is required to prevent contaminating the Problem Domain Design with implementation artifacts. For more information:

- Domain-driven design (http://en.wikipedia.org/wiki/Domain-driven_design)
- Domain-model (http://en.wikipedia.org/wiki/Domain_model)
- How to create a good domain model. Top 10 advices (http://www.makinggoodsoftware.com/2010/05/17/how-to-create-a-good-domain-model-top-10-advices/)
- Martin Fowler (2003) *Patterns of Enterprise Application Architecture*, Addison



The problem domain component is delivered as a separate JAR, which is linked by the server and the client, so they are always in synch. You declare everything once, change once, it will be available in either side automatically.

All business objects inherit from KBusinessObject. This will enable the objects to be used by many tools, at the server and client.



Business objects need not be very complicated. Any method, variable, constant or enum declared is available at the server and client.

Check the tutorial to see how to make business objects easily.

2.1.1. Object Oriented Design (OOD) elements utilized

Once we discover the business objects we need to define the relationships amongst them. The relationships define the sequence in which objects are created, navigated, and represented together in views.

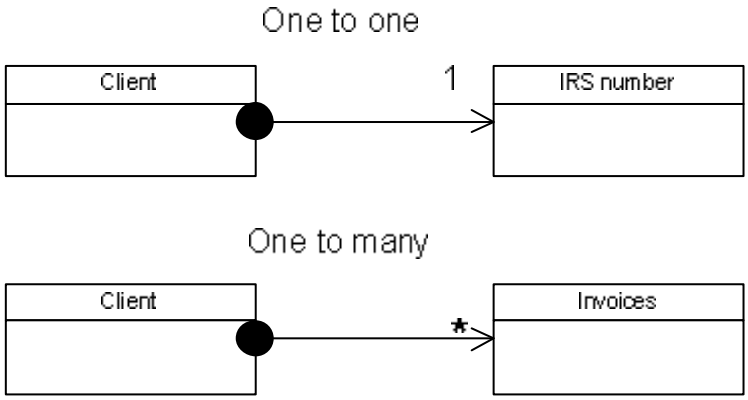
1.1.1.1. Primary Keys / Object IDs

First, all objects are identified by a key. In the K-Framework we use "dumb" keys using simple integers. There is no instance with out an ID and no ID is complex or related to business information. Keys are implementation artifacts not to be used for any business process. If ID as necessary for a business process, like account numbers or sample IDs, a separate field is to be created. This is known as a surrogate key. In case of errors, surrogate keys can be changed; primary keys shall never be changed.

1.1.1.2. Your diagram should be hierarchical

A hierarchical design should not have cycles and should conform to a directed acyclic graph. Not only are this designs easier to develop because modules can be isolated, but they normally do not map to a logical business process indicating an error.

1.1.1.3. Aggregation (Has):



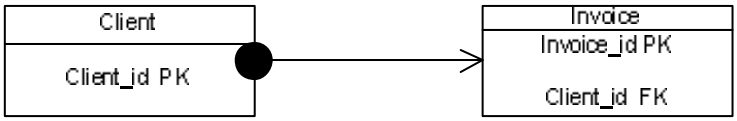
Business objects form relationships in a tree form where some objects are created and inserted directly to the system, like clients, and others are then created descending from these initial objects, like sales.

Under this relationship we say that clients "have" sales, and they are depicted using the above arrow flowing from the father to the children.

Objects participating in these kind of relationships are protected by the system so that child objects can not be created with out a father and fathers can not be deleted leaving "orphaned" children.

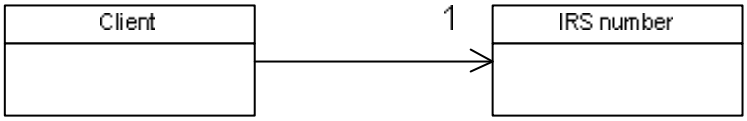
This is the most common relationship.

Aggregations are implemented by adding a field on each child object that holds the ID of the parent. For automatic integrity enforcement in the K-Framework these fields, known as Foreign Keys, must be named as the primary key they point to, therefore all primary keys must have unique names. Class constructors will require the ID of the parent for the child to be constructed.

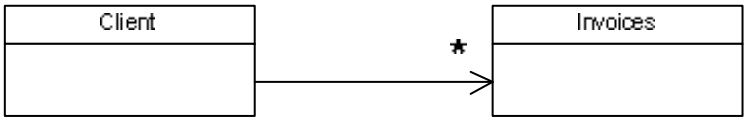


1.1.1.4. Association (Uses):

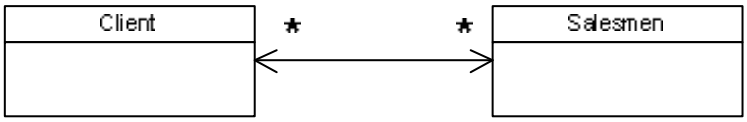
One to one



One to many



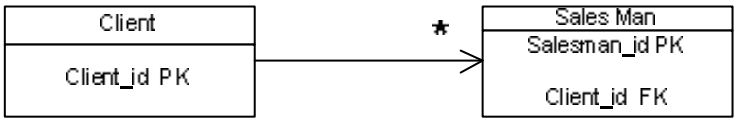
Many to many



As the system grows in complexity deferent objects from different lines need to know each other and form simple relationships. We use a simple arrow line between these objects flowing from the object that requires the information to the source of the information.

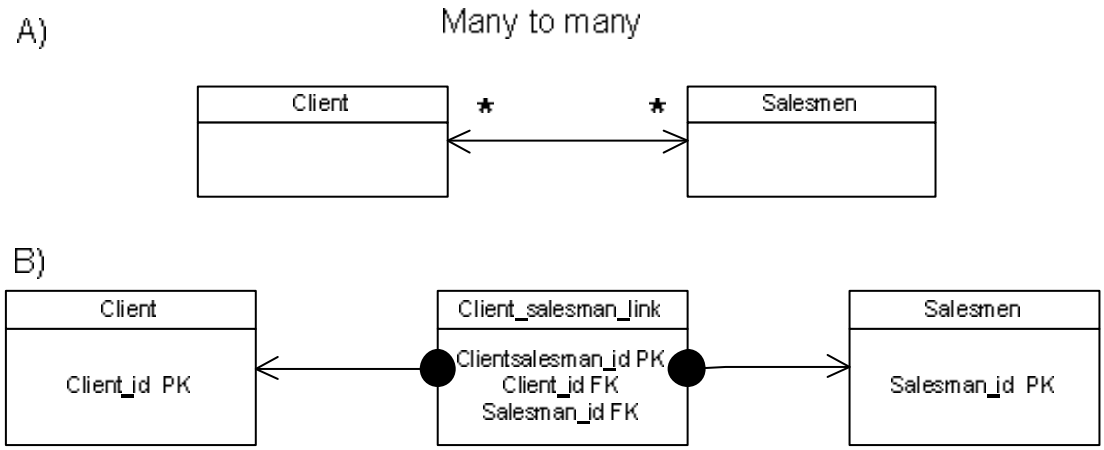
These relationships are implemented in many ways. For example a class function might receive another class as a parameter; in this case a Foreign Key might not be required.

If the relationship is to be persisted, a Foreign Key can be used, just like with aggregations.



A special case is the many to many relationship, where an instance can refer to several classes, and these refer back to several instances of the other class. To implement this relationship, and avoid violating the no cycles rule, a link object is required. In the k-framework a link object is never build using a compound key, but as a regular object with its own Primary Key, but holding two Foreing Keys, each to the objects it is relating.

For Link objects to be handled by the framework, you need to create the corresponding PDC class. It is not unusual for this links to also hold data in the link and become regular PDC objects.



If the link holds not data, you can just use A) to represent the link. If the link holds data, then use B) and treat the link like any other PDC class. In the example above you might want to save the Quota for the salesman for the corresponding client in the link, for example.

In summary, a business object can have relationships with other objects in two ways: Objects can form an aggregation, or they can have a simple relationship. Normally an object will only have a parent forming an aggregation and none or several simple relationships. These relationships can have different cardinality. The cardinality in an association refers to the number of objects that can participate on either side of the relationship.

1.1.1.5. Field / Column Naming Conventions

To prevent conflicts in the mappers, it is required for all fields to be uniquely named. This is also a good recommendation even if not using the KFramework, it's just common sense and hard learned experience.

You can get around data fields of the same name in different tables, but PK fields names must be uniquely named, and the corresponding FKs named exactly as the PK pointed to, or the automatic referential integrity will not work.

For example, if a table has a field named "name" or "Address" or "status", it is very likely that other tables will have an equally named field. This is bound to create a lot of confusion when you need to join tables or code java, since you will end up with clashing field names.

You can get around this with the following code in the PDC objects:

```
@Column(name = "ADDRESS")
private String clientAddress;
```

… but then the java field names will not match those on SQL queries, again adding confusion an making debugging much more difficult. Use it at your own discretion, you have been warned.

You can also add an alias to the FK so that it is recognized as a FK to a differently named PK, but this was designed for cases when you need more than one FK to the same PK, don't abuse it. See the KBusinessObject reference to use aliases.

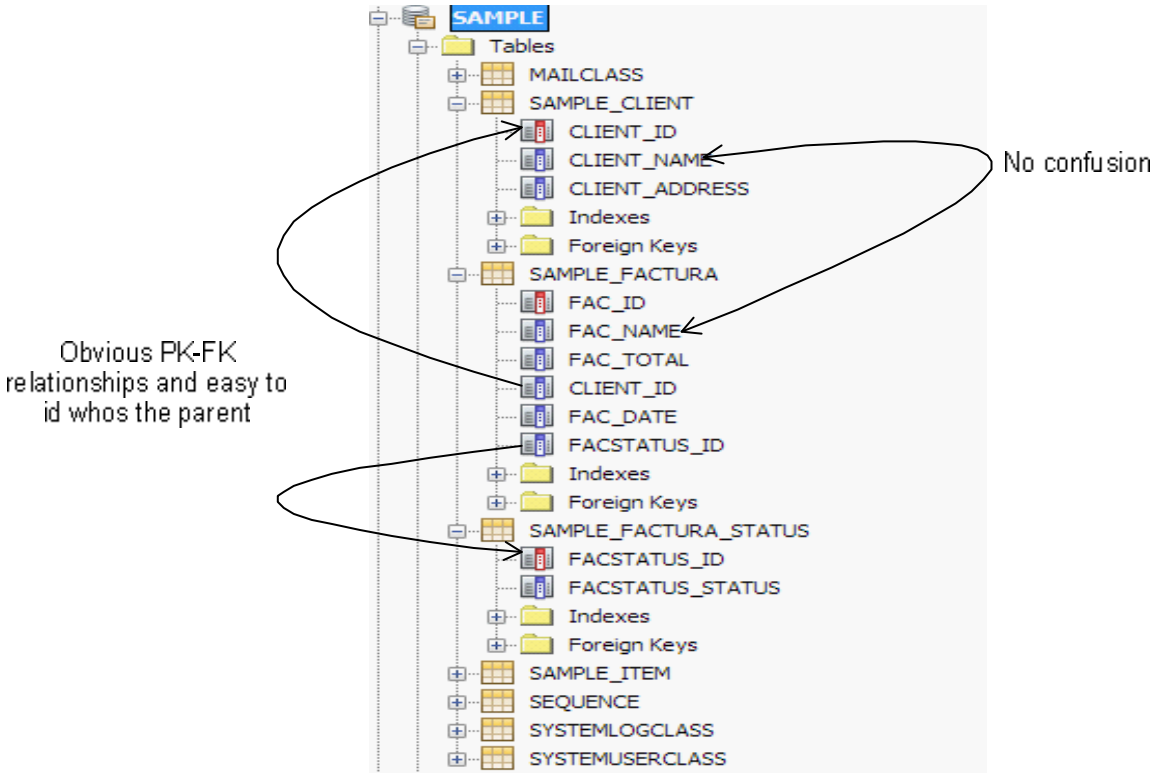
The recommended approach to avoid any of these issues in the kframework, java and SQLin general is to:

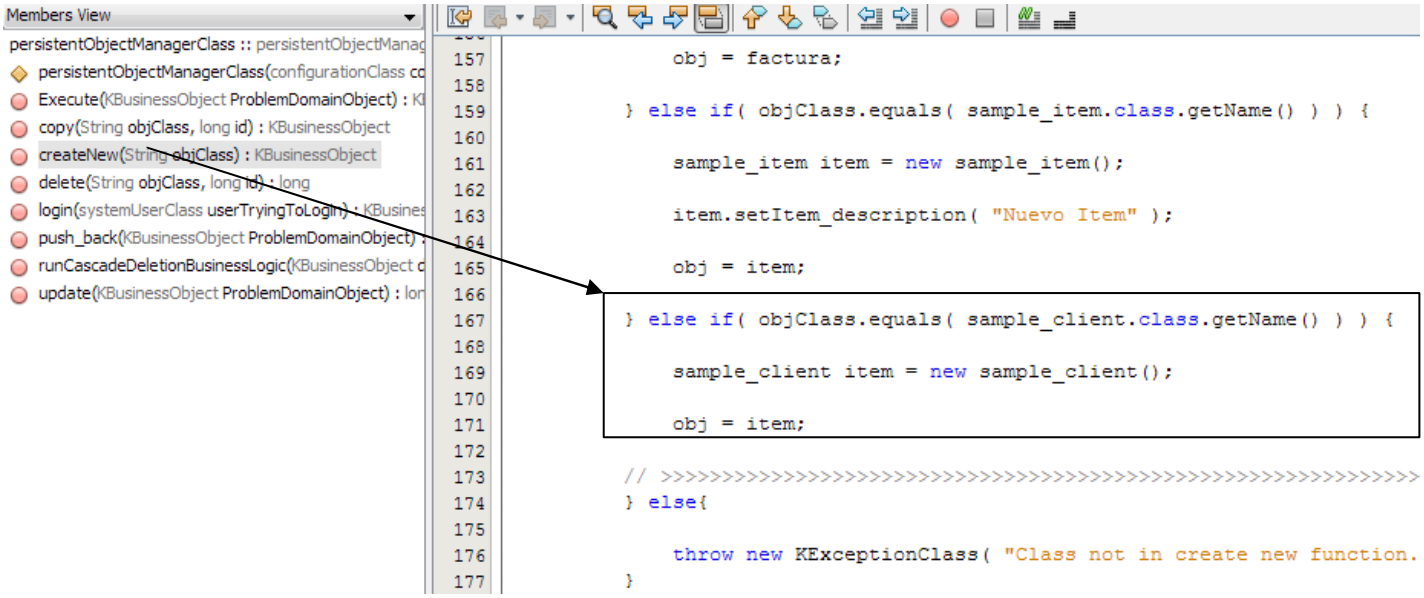
1) Name all fields as "PREFIX-FIELD", where prefix is the name of the table or its acronym, and then the field name.

Note that only simple integer keys are supported as PK-FK, and there is no plan to support complex keys or non numeric keys, since they are not deemed a recommended approach

PK values of 0 or less than 0, -1, -2 etc. are ok, but they are used for special cases and become read only. When ever you need a record protected like a basic default value for a combo or whatever like that, set the PK to 0, the framework will prevent it from being deleted or edited.

Naming Convention Example





2.1.2. Configure referential integrity

Referential integrity is automatic. At startup the framework will build a tree and on every delete will verify the affected child objects. The default is for a parent to delete its children if erased, but this can be changed at runtime. If you are ok with the default no coding is necessary.

To prevent conflicts in the PK-FK handling, it is recommended for all fields to be uniquely named. This is also a good recommendation even if not using the KFramework, it's just common sense and hard learned experience.

You can get around data fields of the same name in different tables, but PK fields names must be uniquely named, and the corresponding FKs named exactly as the PK pointed to, or the automatic referential integrity will not work. See the previous section for more details.

1) At the createNewObject function add any code to initialize the object. Here you can code any rules or queries needed to default any fields to necessary initial states.

```
        obj = factura;

    } else if( objClass.equals( sample_item.class.getName() ) ) {

        sample_item item = new sample_item();

        item.setItem_description( "Nuevo Item" );

        obj = item;

    } else if( objClass.equals( sample_client.class.getName() ) ) {

        sample_client item = new sample_client();

        obj = item;

    // >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    } else{

        throw new KExceptionClass( "Class not in create new function.
    }
```

2) Declare for integrity management to the POM in the constructor. Classes declared here are used by the POM to build the hierarchy tree and enforce referential integrity. If you do not subscribe the class, it will be available to use, but will not be checked for dependant objects or as dependant of another class.

```
 /** Creates new persistentObjectManager */
public persistentObjectManagerClass(
            configurationClass configurationParam,
            logClass logParam )
throws KExceptionClass{

    // inherits
    super( configurationParam, logParam );

    // uses

    // ----------------------------------------


    subscribePDCClass( systemUserClass.class.getName() );
    subscribePDCClass( mailClass.class.getName()  );

    subscribePDCClass( sample_client.class.getName() );
    subscribePDCClass( sample_factura.class.getName()  );
    subscribePDCClass( sample_item.class.getName()  );


    log.log( this, "Constructed successfully." );
```

3) Finally, if case cascade delete is not appropriate for a relationship, change the rule for that object. Referential integrity rules are dynamic and not harcoded. Every time an object is deleted, its children are looked for and integrity maintained. The function called is runCascadeDeletionBusinessLogic. This function is called for every object to be deleted, either by the user or by the automatic referential integrity, for the system to make a decision at runtime. For example, some items might be allowed deletion as long as some status has not been updated, etc.

```
@Override
public void runCascadeDeletionBusinessLogic(
        KBusinessObject  deletee, String OIDname, long OID,
        KBusinessObject child
        ) throws integrityExceptionClass, KExceptionClass{



    try{


        if( child.getClass() == itemClass.class ){

            throw new integrityExceptionClass(
                    " No allowed to delete " + deletee.className() + " if " + child.className() +
                    " exist for this object. "
                    );
        }
```

In this crude example, itemClass are not allowed to be deleted, from any parent. This will prevent the items to be deleted AND the parent to be deleted.

2.1.3. KBusinessObjectClass reference

Field Summary	

java.util.HashMap	aliasFieldNames
	Used to add an alias to a PK. This is, where the parent has pk named X, use the current field Y as the Foreign Key. Normally the framework will automatically match parent and children by looking for children where a field is named X, but sometimes you have children with more than one parent and you need to name the PKs uniquely, say X, Y, Z. In this case add alias X to Y and Z for them to be recognized as foreign keys to the corresponding parent.
	For example:
	`aliasFieldNames.put("X", "Y");`
	`aliasFieldNames.put("X", "Z");`

boolean	Editable Make the Object read Only.
protected java.util.Map<java.lang.String,java.lang.Integer>	fieldMaxSize A pair fieldName / Size to set the max sizes of fields. The client will automatically set any widget (TextBox, Area) on visualize to match the max for the corresponding field.
java.util.Vector<java.lang.String>	fieldNames Listo fieldnames of the object
protected java.util.Map<java.lang.String,java.lang.Integer>	fieldTypes Set the field types by adding fieldName / Types pairs. Used by client on visualize to automatically set the format and validation rules of text boxes. **CURRENCY_TYPE to currency** **DATE_TYPE to date** **NUMERIC_TYPE numeric free form** **NUMERIC2_TYPE numeric 2 decimals**

	`NUMERIC5_TYPE` numeric 5 decimals `TIME_TYPE` hour minute
java.lang.String	Oldfield Returns the name od the OID field (PK)
protected java.util.Vector<java.lang.String>	readOnlyFields Set the fields to read only by adding fieldNames. Used by client on visualize to automatically disable the corresponding widgets.

Method Summary	
boolean	**compareByFields**(KBusinessObject other) Compares two objects and returns true/false. False if they do not share all the fields.
boolean	**compareCommonFields**(KBusinessObject other) Compares two objects and true / false. You can also use it. Fields not shared are ignored on either side.
java.lang.String	**diff**(KBusinessObject comparator) Returns a report on the difference between two objects. Used by the log to save the object differences.
void	displayVisualize(java.awt.Container visualDisplay) This method will automatically display a business object to an edit dialog. It will look into the given component and find all widgets where the property name is set to a field, and set the value of the widget, whether it is a text box, label, combo, date picker etc. This will fail if a component is missing for a field. USE THIS.

void	displayVisualize(java.awt.Container visualDisplay, int missingFieldsBehavior)
	This method will automatically display a business object to an edit dialog. It will look into the given component and find all widgets where the property name is set to a field, and set the value of the widget, whether it is a text box, label, combo, date picker etc.
	The missingFieldsBehavior indicates what to do if a widget is not found for a field:
	`DONOT_IGNORE_MISSING_FIELDS`
	`IGNORE_MISSING_FIELDS`
	This is useful for information boxes inside windows, where we do not expect to edit the object.

void	displayVisualize(java.awt.Container visualDisplay, java.util.List extraList) This method will automatically display a business object to an edit dialog. It will look into the given component and find all widgets where the property name is set to a field, and set the value of the widget, whether it is a text box, label, combo, date picker etc. This will fail if a component is missing for a field. USE THIS. The extraList component is used to include dropDownFillerClasses which control a widget indirectly. That is, for bound combos using dropDownFillerClasses, you need to pass the controller: dropDownFillerClasses, not the widget. Add as many as required.
void	displayVisualize(java.awt.Container visualDisplay, java.util.List extraList, int missingFieldsBehavior) This method will automatically display a business object to an edit dialog. It will look into the given component and find all widgets where the property name is set to a field, and set the value of the widget, whether it is a text box, label, combo, date picker etc. The missingFieldsBehavior indicates what to do if a widget is not found for a field: **DONOT_IGNORE_MISSING_FIELDS** **IGNORE_MISSING_FIELDS** The extraList component is used to include dropDownFillerClasses which control a widget indirectly. That is, for bound combos using dropDownFillerClasses, you need to pass the controller: dropDownFillerClasses, not the widget. Add as many as required.
java.util.Iterator	getFieldNamesIterator () Returns a list with all the field names.
long	getOID() Returns the PK value of the object.
void	materializeField(java.lang.String fieldName, double newValue) Set a value for a field.
void	materializeField(java.lang.String fieldName, float newValue) Set a value for a field.
void	materializeField(java.lang.String fieldName, int newValue) Set a value for a field.
void	materializeField(java.lang.String fieldName, long newValue) Set a value for a field.
void	materializeField(java.lang.String fieldName, java.lang.String newValue) Set a value for a field.
void	materializeFromDisplay(java.awt.Container visualDisplay) Takes an object and sets it from a window. This function will look for all the widgets named as a field, and set the fields to the found value. Used after an OK is click in a window to obtain the changes to the business object.

void	materializeFromDisplay(java.awt.Container visualDisplay, int missingFieldsBehavior) Takes an object and sets it from a window. This function will look for all the widgets named as a field, and set the fields to the found value. Used after an OK is click in a window to obtain the changes to the business object. The missingFieldsBehavior indicates what to do if a widget is not found for a field: **DONOT_IGNORE_MISSING_FIELDS** **IGNORE_MISSING_FIELDS**
void	materializeFromDisplay(java.awt.Container visualDisplay, java.util.List extraList) Takes an object and sets it from a window. This function will look for all the widgets named as a field, and set the fields to the found value. Used after an OK is click in a window to obtain the changes to the business object. The extraList component is used to include dropDownFillerClasses which control a widget indirectly. That is, for bound combos using dropDownFillerClasses, you need to pass the controller: dropDownFillerClasses, not the widget. Add as many as required
void	materializeFromDisplay(java.awt.Container visualDisplay, java.util.List extraList, int missingFieldsBehavior) Takes an object and sets it from a window. This function will look for all the widgets named as a field, and set the fields to the found value. Used after an OK is click in a window to obtain the changes to the business object. The extraList component is used to include dropDownFillerClasses which control a widget indirectly. That is, for bound combos using dropDownFillerClasses, you need to pass the controller: dropDownFillerClasses, not the widget. Add as many as required The missingFieldsBehavior indicates what to do if a widget is not found for a field: **DONOT_IGNORE_MISSING_FIELDS** **IGNORE_MISSING_FIELDS**
void	materializeObject(KFrameWork.Base.logClass log, java.util.Map<java.lang.String,java.lang.String> attrMap) Reads and sets an object from a map of attributes.
java.lang.String	objectString() Returns a nice report of all the fields and the current values.
java.lang.String	serializeField(java.lang.String fieldName) Returns a field in text form, no matter the type.

java.util.Map<java.lang.String,java.lang.String>	serializeObject(KFrameWork.Base.logClass log) Returns a map of field name / value pair from any object..

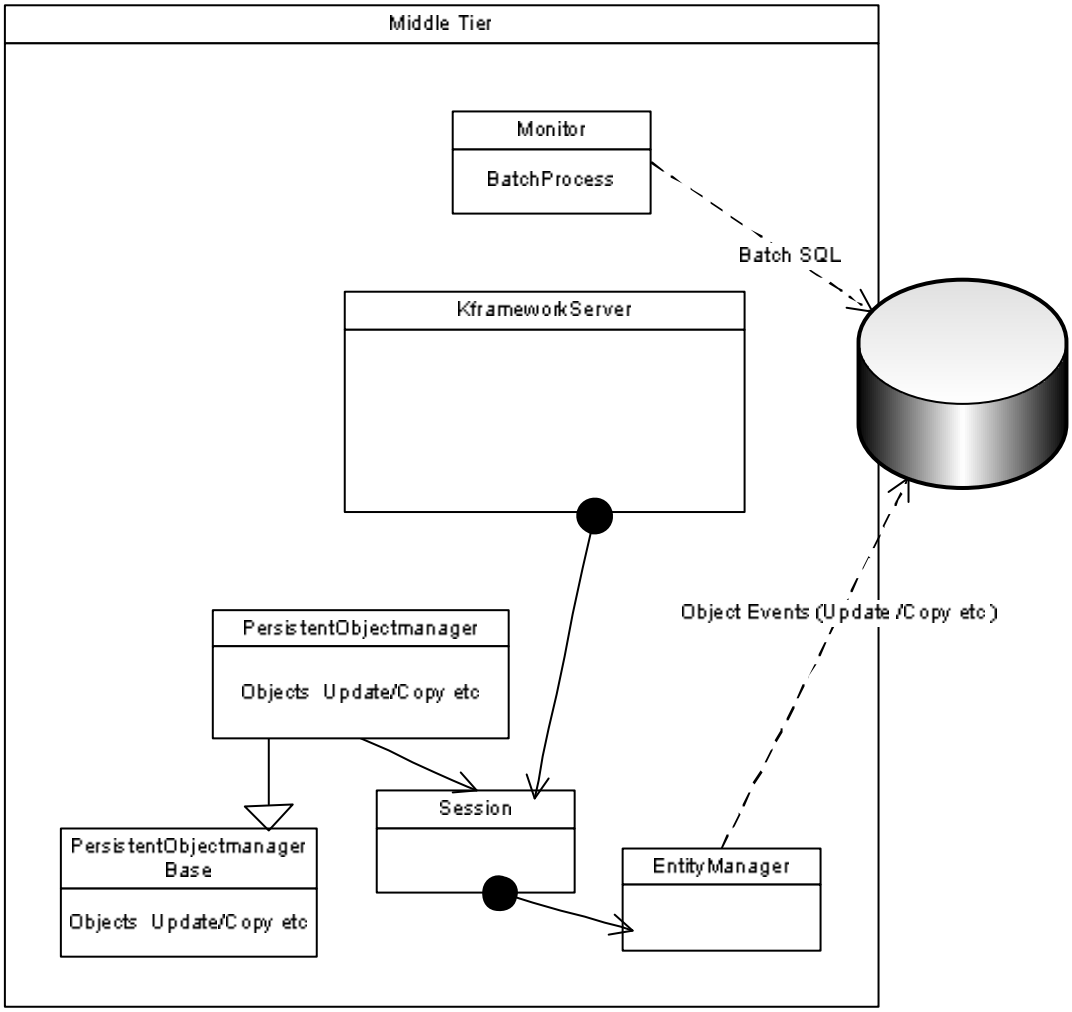
2.2. Implementing Business Logic: The Persistent Object Manager

The POM is the server implementation. The POM, as delivered, works for all basic functions called by the client, that is: insert, copy, update and delete an object. You only need to add business logic, everything else is provided.

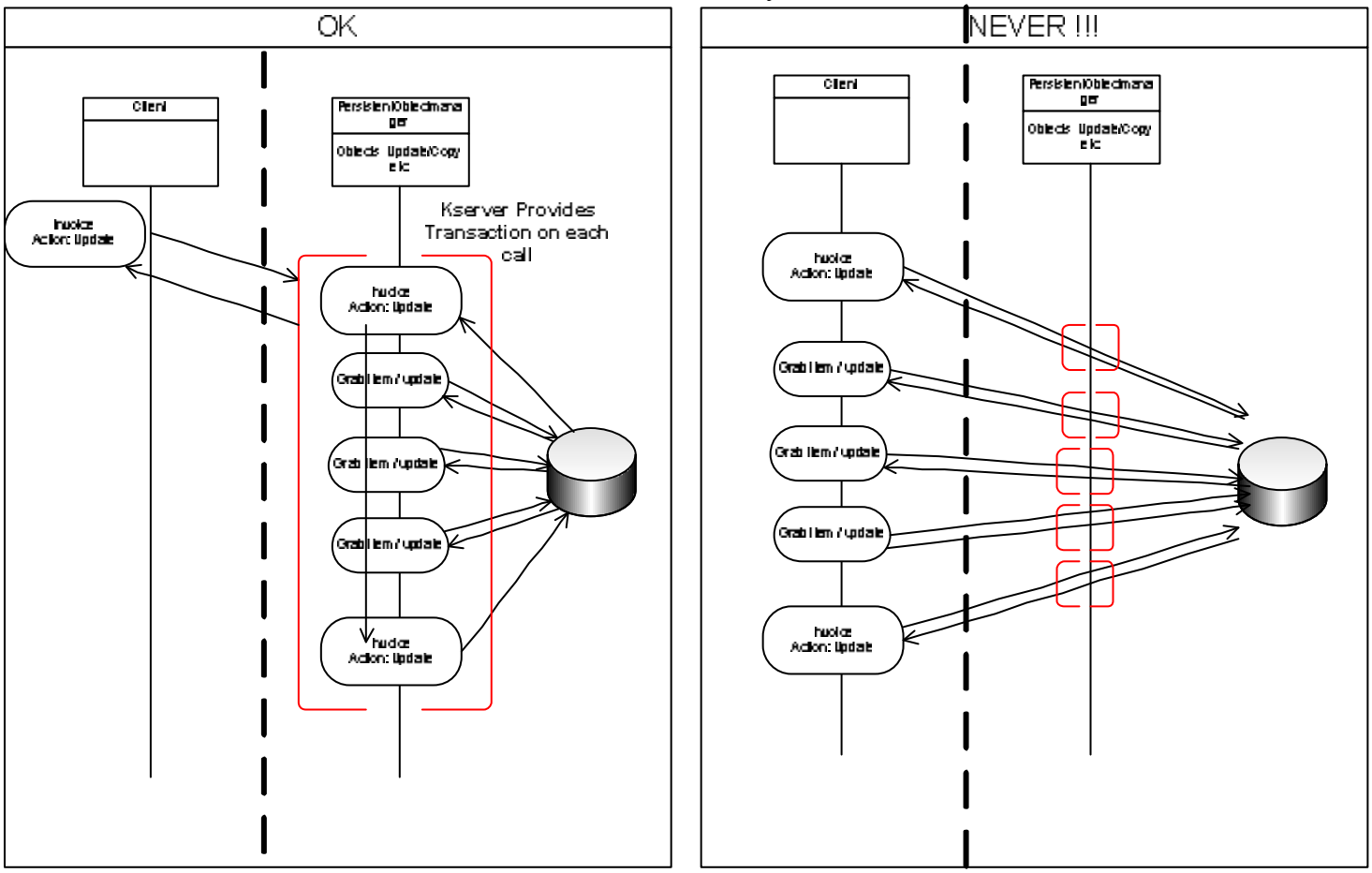
So the POM has a collection of functions that, as delivered, simply call the super class function.

The POM is designed to handle the business logic. If you handle too much in the client side, sooner or later you will run into problems of having to synch a lot of things and having to hold transactions for long time. A well designed system will not need to do that.

Make a clear distinction between a client managing an object and the server resolving dependencies and multiple objects.



Correct and incorrect flows are:



To achieve this, the POM lets the user implement each action, apply any logic, and then call the base action.

POM needs to inherit from persistentObjectManagerBase.

Note that referential integrity is automatic. At startup the framework will build a tree and on every delete will verify the affected child objects. The default is for a parent to delete its children if erased, but this can be changed at runtime. If you are ok with the default no coding is necessary. The way it works is that the frameworks checks all objects containing a field (The FK) names the same as the PK of the object currently to be deleted. This is cascaded automatically with no limit on depth. The default is to delete children on father deletion, but you can override this, because on every single delete, a callback is called to verify the logic so that you can decide at runtime and depending on data.

A transaction is always started when the POM is called, and committed on successful exit, including the batch process. You can obtain the transaction from the session object. It something goes wrong, just throw and exception. The server will rollback and send the exception's message to the server.

Method Summary	
KFrameWork.Base.KBusinessObject	copy(java.lang.String objClass, long id) This function is called when a client requests an object. You can add logic here before or/and after the action.
abstract KFrameWork.Base.KBusinessObject	createNew(java.lang.String objClass) This function is called every time a new object is required. Don't confuse with push_back which is called by the clients. You have a chance here to initialize objects.
long	delete(java.lang.String objClass, long id) This function is called by the client to start a delete operation. You can add logic. Remember that once you call the base delete, the server will search for children and call the cascadeDeletion for them. If you do not code anything there, the default is to remove the objects. By overriding you can block the deletion, change some other status, or any other logic.
abstract KFrameWork.Base.KBusinessObject	Execute(KFrameWork.Base.KBusinessObject ProblemDomainObject) Some times you need to tell the server an event has occurred for an object, which is not a push_back, copy, update or delete. In this case, in orther to trigger some logic at the server, you call execute with a specific message. For example, in an client's edit dialog window:

```
private void concilia(){

    if( !metaUtilsClass.showConfirmationMessage( this, "Sure recalc?" ).equals( " OK " ) ){
        return;
    }

    try {

        ///define parametros de la funcion
        HashMap functionParameters = new HashMap();

        /// Ejecuta funcion
        amexClass amex = new amexClass();

        persistentObjectManagerClass persistentObjectManager =
                new persistentObjectManagerClass( configuration, log );

        persistentObjectManager.execute(amexClass.Functions.RECALC.type, amex, functionParameters);

        amexBrowser.refresh();

    } catch (customExceptionClass ex) {
        metaUtilsClass.showErrorMessage( this, "Cant recalc: " + ex.toString() );
        log.log(this, ex.toString());
    }

}
```

Then in the server:

```
/** Get the object in DB */
public serializableObjectInterface Execute( sessionClass session, EntityManager manager,
        String objClass,serializableObjectInterface ProblemDomainObject )

throws Exception {

    String message = "Executing function from object: [" + objClass + "] ";
    log.log( this, message );

    String fuctionName = ProblemDomainObject.serializeField("functionName");


        //  ----------------------------------------------
        // BUSINESS RULES update OBJECT -------------------


        if( objClass.equals( ObjClassEnum.AMEX.type ) ) {

            // Elimina los registros encontrados entre las fechas recibidas
            if ( fuctionName.equals(amexClass.Functions.RECALC.type) ) {
```

Note the constants are part of the business object, which is automatically shared by client and server.

long	getDefaultId(java.lang.String IDfield, java.lang.String Table) Use normally when creating new objects, this will return the minimum value of the PK.
long	getDefaultId(java.lang.String IDfield, java.lang.String Table, java.lang.String SQLcommand) Use normally when creating new objects, this will return the minimum value of the PK constrained with a where clause.
void	initialize(sessionClass sessionParam) This initializes the POM. Basically, sets the session which will be used.

	You can not call any function, except login, before a session is set. The delivered sample project handles this. But you could use a POM outside of the framework or j2ee server and handle the session manually. A session just holds an entity manager, and db connection. Though you can add some application's specific data.
void	logAction(java.lang.String action, KFrameWork.Base.KBusinessObject target, java.lang.String message) This is called to log something. Internally it is called an every change.
KFrameWork.Base.KBusinessObject	login(ProblemDomainComponent.systemUserClass userTryingToLogin) Here is the authentication. A simple table based authentication is provided, but you might want to change it. See the code for more details. A successful login returns a valid session.
long	push_back(KFrameWork.Base.KBusinessObject ProblemDomainObject) This function is called when a client requests to insert a new object. You can add logic here before or/and after the action.
void	subscribePDCClass(java.lang.String ClassName) This is used in the constructor to add a class to the integrity validation. A class not subscribed will not be checked for children on deletion, nor will be verified in case it has a parent. If you know a class is stand alone, don't include it here, but do not forget to add any class that has a "has" relationship.
long	update(KFrameWork.Base.KBusinessObject ProblemDomainObject) This function is called when a client sends an object for update. You can add logic here before or/and after the action.

2.3. Using the session

Each POM must run under a session, which is passed in the initializer function. Either you can make sessions by hand, or use the POM own login function. Using the Kserver, you have one session for each connected user. Sessions cache dbconnections and other things. Sessions are completely destroy after they expire.

Field Summary	
KFrameWork.dataBase.dataBaseClass	dataBase Current DB connection
long	lastTime Las time is was used
java.util.Map	userItems Hang here what ever you need. Now you should not need it, since all is managed at the client. It is not even constructed; you need to connect it to some Map instance, before using it.

Constructor Summary
sessionClass(KFrameWork.Base.configurationClass configurationParam, KFrameWork.Base.logClass logParam)
sessionClass(KFrameWork.Base.configurationClass configurationParam, KFrameWork.Base.logClass logParam, java.lang.String driver, java.lang.String url, java.lang.String username, java.lang.String password, java.lang.String PersistenceUnitName)

Method Summary	
void	commitEntityTransaction() Used by Server, this commits the transaction upon event handler completion
void	finalize() Closes resources in case we forget
javax.persistence.EntityManager	getEntityTransaction() Gets a JPA ET
java.lang.String	getID() Get my ID
long	getLastTime() Last time accesed
javax.persistence.Query	getQuery(java.lang.String SQLcommand) Returns a JPA query, ready to bind or execute
void	kill() Close everythying, and destroy
javax.persistence.EntityManager	openEntityTransaction() To be used bt Server to start a transaction
void	resetTime() Touch time
void	rollbackEntityTransaction() To be used by Server on error returned in an event handler

The time to live for an idle session is configured in the main configuration file:

```
session_idle_max_mins=30
```

2.4. Using the Audit Trial

The system can log ALL actions to an audit trail. Every update, delete or insert will trigger the function. The before and after are compared and a log entry saved.

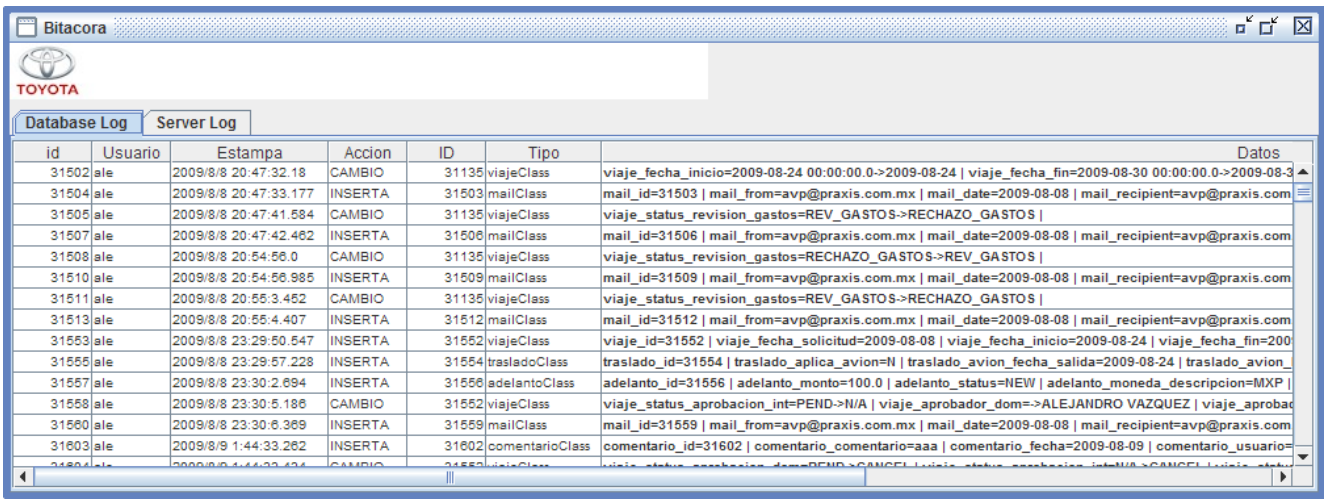
To enable the audit trial, make sure you have the table system_log in your database. To enable the audit trial:

1) Enable the audit trail in the server's magnus.conf, don't confuse with the normal log.

```
audittrail=yes
```

You might notice that the framework provides a systemLogClass to read audit trial entries, like any other business object.

Check the sample client for the provided audit trial browser and editor.



id	Usuario	Estampa	Accion	ID	Tipo	Datos
31502	ale	2009/8/8 20:47:32.18	CAMBIO	31135	viajeClass	viaje_fecha_inicio=2009-08-24 00:00:00.0->2009-08-24 | viaje_fecha_fin=2009-08-30 00:00:00.0->2009-08-3
31504	ale	2009/8/8 20:47:33.177	INSERTA	31503	mailClass	mail_id=31503 | mail_from=avp@praxis.com.mx | mail_date=2009-08-08 | mail_recipient=avp@praxis.com
31505	ale	2009/8/8 20:47:41.584	CAMBIO	31135	viajeClass	viaje_status_revision_gastos=REV_GASTOS->RECHAZO_GASTOS |
31507	ale	2009/8/8 20:47:42.462	INSERTA	31506	mailClass	mail_id=31506 | mail_from=avp@praxis.com.mx | mail_date=2009-08-08 | mail_recipient=avp@praxis.com
31508	ale	2009/8/8 20:54:56.0	CAMBIO	31135	viajeClass	viaje_status_revision_gastos=RECHAZO_GASTOS->REV_GASTOS |
31510	ale	2009/8/8 20:54:56.985	INSERTA	31509	mailClass	mail_id=31509 | mail_from=avp@praxis.com.mx | mail_date=2009-08-08 | mail_recipient=avp@praxis.com
31511	ale	2009/8/8 20:55:3.452	CAMBIO	31135	viajeClass	viaje_status_revision_gastos=REV_GASTOS->RECHAZO_GASTOS |
31513	ale	2009/8/8 20:55:4.407	INSERTA	31512	mailClass	mail_id=31512 | mail_from=avp@praxis.com.mx | mail_date=2009-08-08 | mail_recipient=avp@praxis.com
31553	ale	2009/8/8 23:29:50.547	INSERTA	31552	viajeClass	viaje_id=31552 | viaje_fecha_solicitud=2009-08-08 | viaje_fecha_inicio=2009-08-24 | viaje_fecha_fin=200
31555	ale	2009/8/8 23:29:57.228	INSERTA	31554	trasladoClass	traslado_id=31554 | traslado_aplica_avion=N | traslado_avion_fecha_salida=2009-08-24 | traslado_avion_
31557	ale	2009/8/8 23:30:2.694	INSERTA	31556	adelantoClass	adelanto_id=31556 | adelanto_monto=100.0 | adelanto_status=NEW | adelanto_moneda_descripcion=MXP |
31558	ale	2009/8/8 23:30:5.186	CAMBIO	31552	viajeClass	viaje_status_aprobacion_int=PEND->N/A | viaje_aprobador_dom=->ALEJANDRO VAZQUEZ | viaje_aprobac
31560	ale	2009/8/8 23:30:6.369	INSERTA	31559	mailClass	mail_id=31559 | mail_from=avp@praxis.com.mx | mail_date=2009-08-08 | mail_recipient=avp@praxis.com
31603	ale	2009/8/9 1:44:33.262	INSERTA	31602	comentarioClass	comentario_id=31602 | comentario_comentario=aaa | comentario_fecha=2009-08-09 | comentario_usuario=

2.5. Accessing the database directly in the server with SQL

You might need to access the database directly while in an event handler of the POM (copy, update, execute etc…).

The preferred option is to access through the JPA's entity manager; by getting the entity manager form the session.

Check the JPA documentation on how to query using the EntityManager.

2.5.1. Connecting to a database and setting up transactions using JPA

The preferred option is to access through the JPA's entity manager; by getting the entity manager form the session. The POM always has an active session you can access.

Check the JPA documentation on how to query using the EntityManager.

Example:

```
String SQL =  "SELECT x FROM adelantoClass x WHERE x.viaje_id = ?1" +" AND
x.adelanto_moneda_descripcion = ?2";

Query q = session.getEntityTransaction().createQuery( SQL );
q.setParameter(1, viaje_id);
q.setParameter(2, moneda);
List< adelantoClass > results =  ( List< adelantoClass > ) q.getResultList();

if(!results.isEmpty()){ …
```

2.5.2. Connecting to a database and setting up transactions using JDBC: The DataBaseClass

First thing to connect to the database is to get a database connection. Then you create SQL commands from that connection. The connection controls the transaction across all its commands and result sets. In the Kramework connections are created by the dataBaseClass. The database classes take their configuration from a configuration object, and they require a log object.

Example connect to a database:

In the magnus.conf, or the configuration file you choose:

```
// MSSQL
//db_username=sample
//db_password=sample
//db_url=jdbc:sqlserver://127.0.0.1:1433;databaseName=SAMPLE1
//db_driver=com.microsoft.sqlserver.jdbc.SQLServerDriver

// Oracle
db_username=sample
db_password=sample
db_url=jdbc:oracle:thin:@localhost:1521:XE
db_driver=oracle.jdbc.OracleDriver
```

The, in the server code:

```
dataBaseClass dataBase = new dataBaseClass( configuration, log  );
dataBase.connect();

dbTransactionClass query =
               new dbTransactionClass( configuration, log, dataBase );

query.prepare( " select mail_id from mailClass where mail_status = '" +
               mailClass.STATUS_NEW + "' order by mail_id " );
```

```
query.executeQuery();

while( query.fetch() ){

                query.getField( "mail_id" );
}
```

Constructor Summary	
dataBaseClass(configurationClass configParam, logClass logParam)	Contructor. A transaction is started automatically. So don't forget to commit!

Method Summary	
void	close() Diconnect,and close
void	commit() Commit the current transaction
void	connect() create a DB connection.
void	connect(java.lang.String driver, java.lang.String url, java.lang.String username, java.lang.String password) create a DB connection.
void	finalize() Disconnects, in case you forget to close explisitly.
java.sql.Connection	getConnection() Get the JSBC handle form manual control
void	rollback() Rollback all changes

2.5.3. Using DB cursors: The dbTransactionClass

A dbTransactionClass is used to create a DB cursor or executing other, non databaset, SQL commands:

```
dataBaseClass dataBase = new dataBaseClass( configuration, log  );
dataBase.connect();

dbTransactionClass query =
              new dbTransactionClass( configuration, log, dataBase );

query.prepare( " select mail_id from mailClass where mail_status = ? " );

query.bind( "1", "SENT" );

query.executeQuery();

while( query.fetch() ){

              query.getField( "mail_id" );
}
```

For non dataset commands:

```
dbTransactionClass query =
              new dbTransactionClass( configuration, log, dataBase );

query.prepare( " delete from systemLogClass where log_id < ( select max( log_id
) - 5000 from systemLogClass ) " );

query.executeModify();
dataBase.commit();
```

Constructor Summary	
dbTransactionClass(configurationClass configParam, logClass logParam, dataBaseClass dataBaseParam)	Creates a new instance of dbTransactionClass

Method Summary	
void	bind(int paramIndex, java.lang.Object parameterValue) BINDING Parameters can be reassigned at any moment and may be reset only partially
int	columnCount() Result set columna count
void	executeModify()

	Execute non dataset SQL
void	executeQuery() Execute Dataset SQL
boolean	fetch() Retrieve rows
java.lang.String[]	getColumnNames() Listo f columns fetchable
java.lang.String	getField(java.lang.String fieldName) Get a row cell
java.sql.ResultSetMetaData	getMetaData() Set metadata
void	prepare(java.lang.String SQLstring) Prepare Command
java.sql.ResultSet	getResultSet() Access the ResultSet Directly
java.lang.String	toHTMLTable(java.lang.String headingColor) Output the results as an HTML table, with the column names as headings and the rest of the results filling regular data cells.

2.6. Using the Mail Engine

Mails are coded at the client, but always sent by the server. So you might have a client running through the internet in china, he can still send mails. You need to insert mailClass objects into the database, like using any other regular object.

The server implements standard SMTP and it can be authenticated. This will run from windows or UNIX.

For example, in the client do:

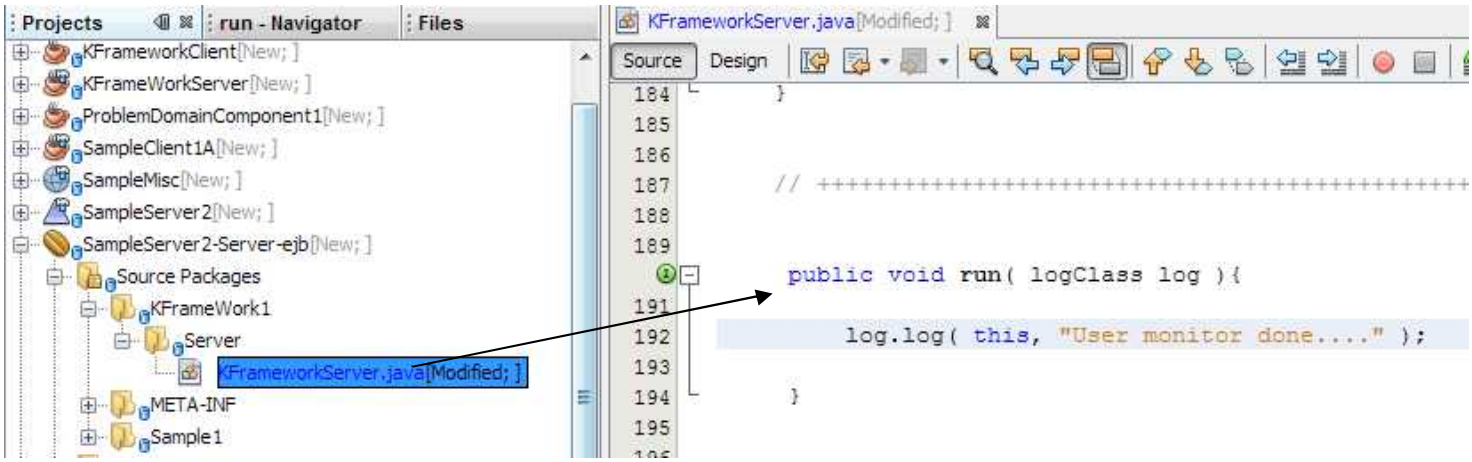
```
persistentObjectManagerClass persistentObjectManager
        = new persistentObjectManagerClass( configuration, log );

mailClass mail = new mailClass();
persistentObjectManager.createNew( mail );

mail.mail_from = from;
mail.mail_recipient = to;
mail.mail_subject = subject;
mail.mail_data = message;
mail.mail_data_type = message_type;  // text_plain
mail.mail_data += comments;

persistentObjectManager.push_back( mail );
```

You need to configure the server, in the main configuration fix:

```
// mailer
mail_enabled=true
mail_server_address=smtp.xxxxx.com
mail_server_user=xxxx
mail_server_password=xxxxx
mail.smtp.auth=false
```

2.7. Using the Batch Process Engine

The server fires a function periodically, which you can use to fire tasks on a schedule. This function is implemented by the KServer. Open the Kserver and find the function "run":



Note that you do not have a Session or POM, because this function is not fired by a user. If you need them, you need to make a session and a POM by hand. Example:

```
// access gwy data
sessionClass GWYsession = new sessionClass(
        configuration, log,
        configuration.getField("gwy_driver"),
        configuration.getField("gwy_url"),
        configuration.getField("gwy_user"),
        configuration.getField("gwy_password"),
        configuration.getField("gwy_pu") );
GWYsession.openEntityTransaction();

persistentObjectManagerClass GWYPOM = new
persistentObjectManagerClass(configuration, log);
GWYPOM.initialize( GWYsession  );
```

.. this will start a transaction, don't forget to commit and rollback after finished. Check the session reference on how to do that.

To configure how often the service runs, you need to code to check the time. The services will fire several times per hour, depending on the configuration file. The server also schedules internal tasks, so don't change the frequency time. Again, your code needs to check when it needs to run, every time the function is fired.

Configuration:

```
clearer_delay_mins=1
```

By default the functions is called every minute. This is also used to check for idle sessions to expire and send mails.

2.8. Doing Authentication and Authorization

Authentication is customizable to what ever you require. The framework provides a simple schema to authenticate using passwords and users in a table in the database, but you can change this to fit your needs.

The POM executes the authentication in function login.

If you wish to change this, all you are required to do is to return any user object.

For authorization, just add any properties to the configuration object, and check the rights every where you might need from the configuration object.

Example POM.login:

```
String user_name = userTryingToLogin.serializeField("system_user_name");

// if OK-> get the user from the DB

EntityManager entityTransaction = session.getEntityTransaction();
List< systemUserClass > results;
String SQL =
        " SELECT x FROM systemUserClass x WHERE " +
        " x.system_user_name = ?1"
Query q = entityTransaction.createQuery( SQL );
q.setParameter(1, user_name);
results =  ( List< systemUserClass > ) q.getResultList();

if(results.isEmpty()){

    // NOOK

    //the user is not registered
    log.log( this, " User [" + user_name + "] is not authenticated!!!" );
```

```
    throw new KExceptionClass( " Invalid user/password for user [" + user_name +
"] ", null );

}
else {


    log.log( this, " User [" + user_name + "] [" + results.get( 0 ).getOID() +
"]is authenticated!!!" );

    session.put( "system_user_name", "" + results.get( 0 ).getOID()  );
    session.put( "system_user_id", "" + results.get( 0 ).getOID()  );
    session.put( "system_user_role", "" + results.get( 0 ).getSystem_user_role()
);
    session.put( "system_user_isadmin", "" + results.get( 0
).getSystem_user_isadmin()  );

    return results.get( 0 );
}

}
```

3. Front-End Reference

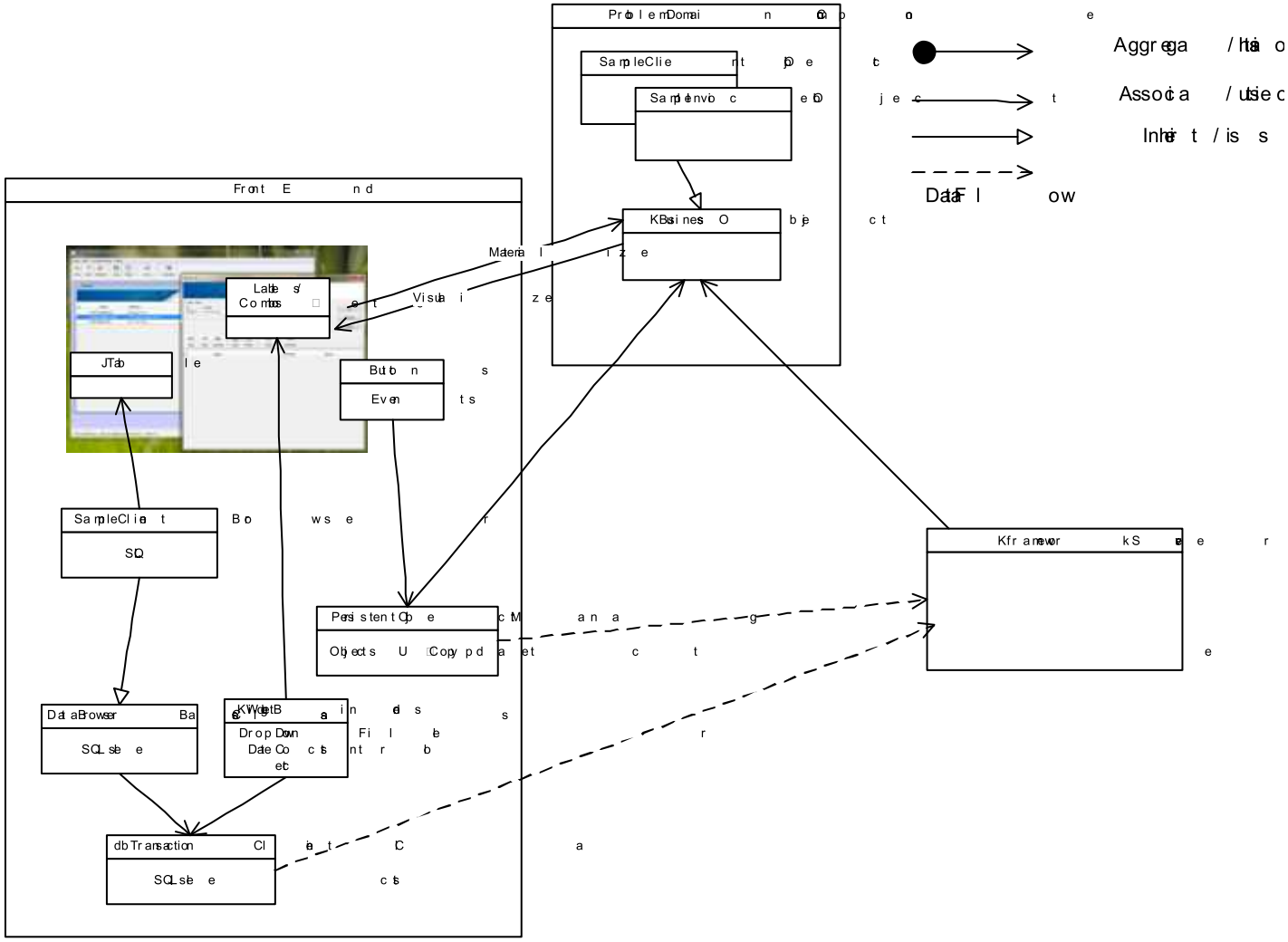
3.1. Using Business Objects in the front end. The KBusinessObjectClass

The KBusinessObject of the client are the same as the server, since they are coded in a separate package that is shared and linked at compile time.

For the reference see the server side reference for the KBusinessObject.

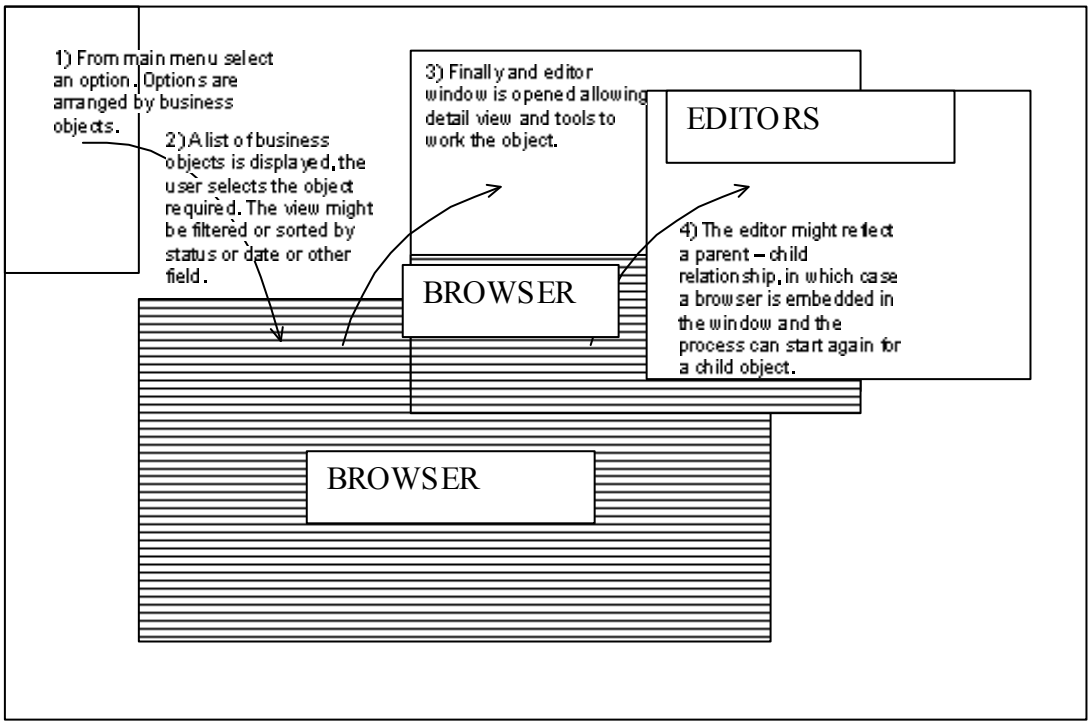
3.2. Using SWING Dialogs: javax.swing.JDialog

The KFramework does not require special code to handle dialogs. You can use a standard JDialog to start with, but the recommended approach is to copy an existing one as a template.



Following the convention an edit dialog will handle editing of one business object. You could edit more than one, but your interface might become too complex to users. Doing this normally indicates a poorly designed UI. See the tutorial for a quick guide on designing your UIs.

The framework is built around a multiple document interface; the general guideline is as follows:



The application's menu should represent the father objects of the PDC design, plus some child objects that can be accessed directly. Inside an editor window a browser can be included to browse the corresponding child objects. A good design will reflect the PDC diagram always. Any deviation can indicate an illogical view or a poorly designed PDC. The PDC is the map to guide all designs for all the developers. See the tutorial for a full example and more details on designing objects and interfaces.

3.2.1. Event Handling

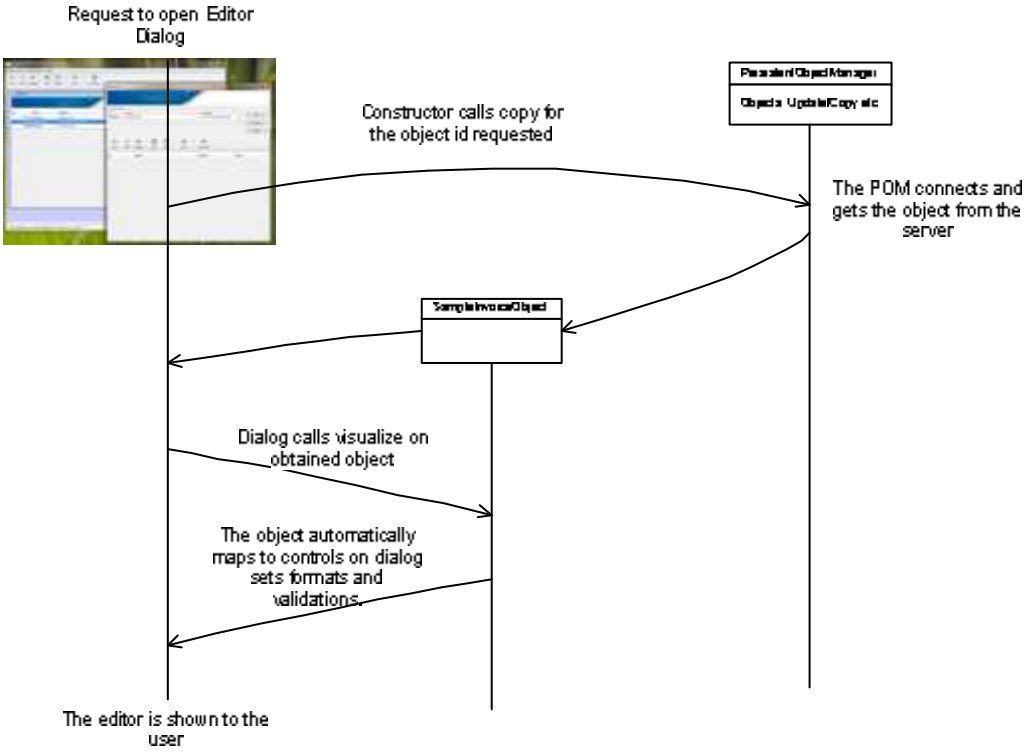
Following this approach you just need to handle the corresponding events for a business object as the user clicks OK or Apply in the Edit Dialogs. Events are handled by the Client's version of the Persistent Object Manager, in the corresponding sub package of the KServerClient. You just need to pass the business object to the corresponding function and the server will take care of sending it to the server, executing the action plus any derived business logic and returning it to the client, when applicable or an error message send by the server.

Example: Normally you will only have 4 event handling functions in a Dialog. Note that you don't code delete here. Objects are normally deleted from the browser:

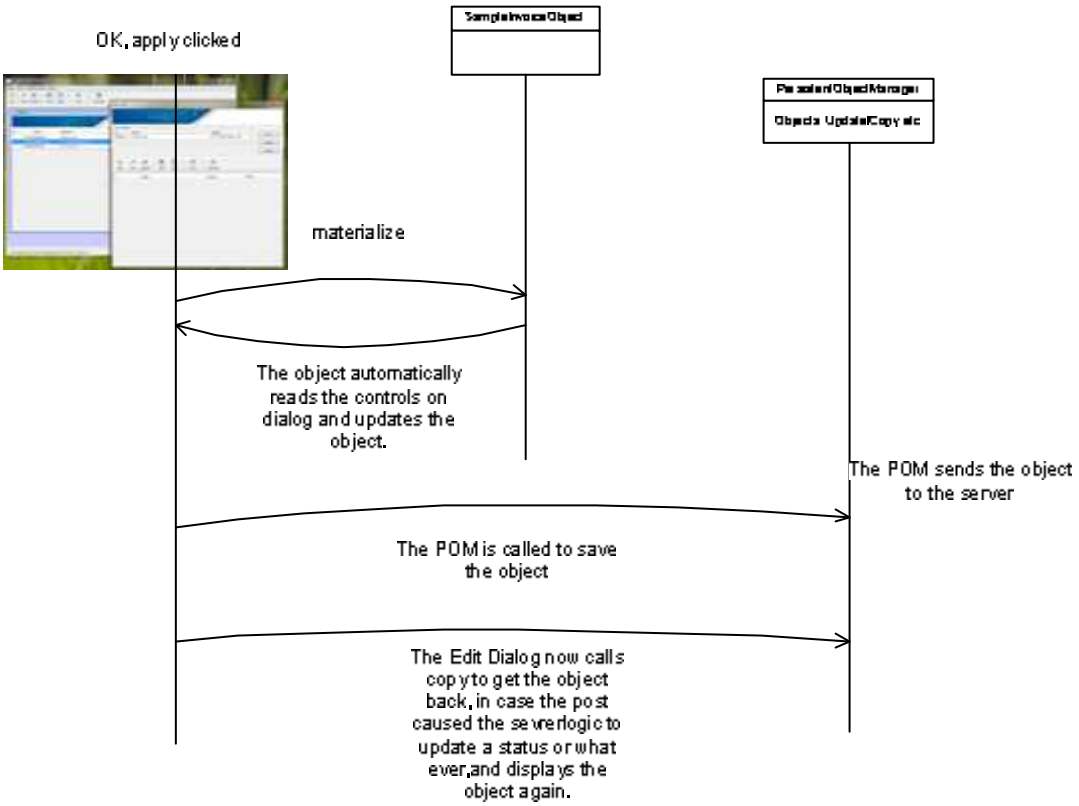
```
public void newObject() // get a new object from server - not posted to DB yet
throws KExceptionClass
{
    sample_client client = new sample_client();
    persistentObjectManagerClass persistentObjectManager =
            new persistentObjectManagerClass( configuration, log );
    persistentObjectManager.createNew( client );
    client.displayVisualize( getContentPane() );
}


public long pushBack() // after newObject, when ready, push:back called to
insert in DB

throws KExceptionClass
{
    sample_client client = new sample_client();
    client.materializeFromDisplay( getContentPane() );
    persistentObjectManagerClass persistentObjectManager =
            new persistentObjectManagerClass( configuration, log );
    long id = persistentObjectManager.push_back( client );

    return id;

}

public void edit( long id ) // Get an object for editing
throws KExceptionClass
{
    sample_client client = new sample_client();
    persistentObjectManagerClass persistentObjectManager =
            new persistentObjectManagerClass( configuration, log );
    persistentObjectManager.copy( id, client );
    client.displayVisualize( getContentPane() );
    setupTables( id );

}

public void updateUser( ) // Save the changes to object
throws KExceptionClass
{
    sample_client client = new sample_client();
    client.materializeFromDisplay( getContentPane() );
    persistentObjectManagerClass persistentObjectManager =
            new persistentObjectManagerClass( configuration, log );
    persistentObjectManager.update_with( client.getClientId(), client );

}
```

These methods handle the following flows to send or retrieve from the server, and to display to the screen automatically. To allow the automatic mapping to work, name the SWING widgets according to the corresponding PDC object field. In other words, for example, paint a edit box in the screen where you want the clientName to be edited, and just name it "clientName". That way the framework will now to map it and read it.

1) To get an object from the server and display:



2) To send updates in an object back to the server:



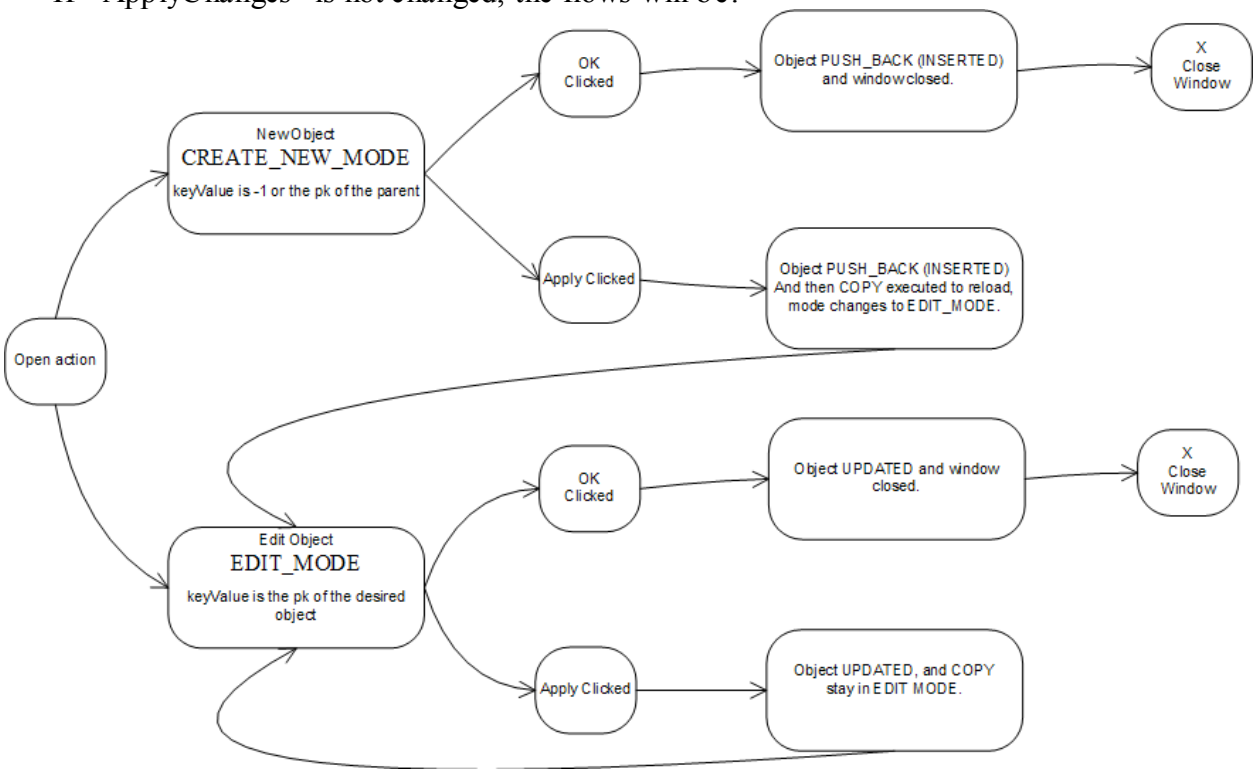
Note how the objects are mapped automatically to widgets via the visualize / materialize functions of the PDC object, which all PDC objects inherit from kBusinessObject. Override this function for any extra validations or formatting, do not put this code in the window, so that this will apply on any window and it is easy to locate.

Also note that the business objects include tools to define the validations per field, so that you seldomly require to complement the visualize or materialize. *Using Dialog's Business Object Auto Field Validation and Formatting*, for more information.

These functions are called when the user clicks OK or Apply, and depend on the "MODE" of the window. Normally you only need to change the class names of the business objects and leave everything else.

The samples include the "ApplyChanges" function that handles modes. You normally do not need to change it, just make sure that any OK or APPLY buttons are connected to it.

If "ApplyChanges" is not changed, the flows will be:



Finally, note that "ApplyChanges" requires two functions: checkSecurity and Setup Tables. You need to code these functions.

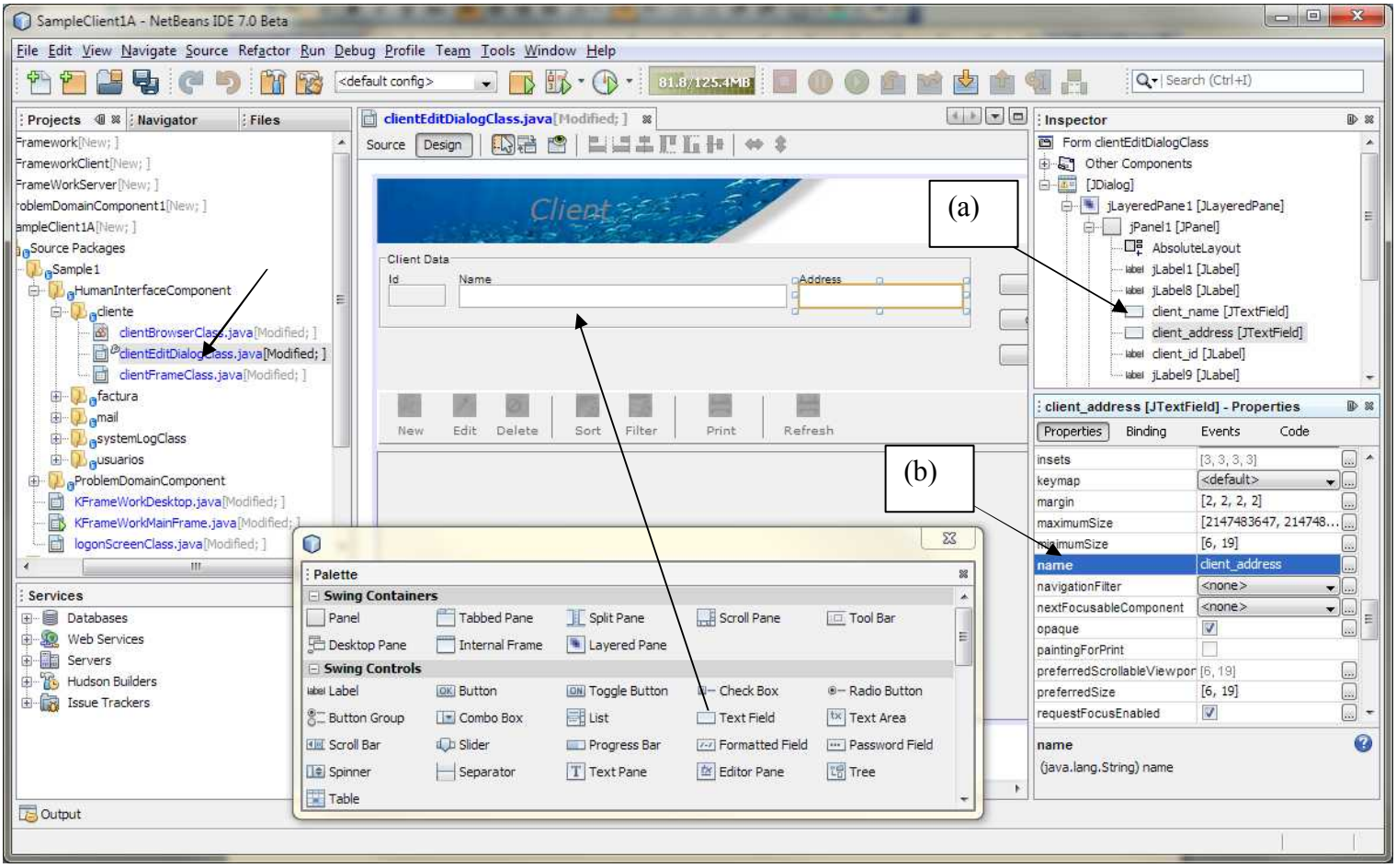
1) **checkSecurity** is used at startup, and after the objects are loaded for you to verify the user permissions and disable any tool or block the window all together.
2) **setupTables** is usted to load sub browsers that display child objects. Note that this can exist if the object has not been posted to the DB, so the function is not called until the EDITOR switches to EDIT_MODE.

Examples:

```
     private void ApplyChanges(boolean closeEditor) {


      try{



          if( operationMode == CREATE_NEW_MODE ){
              long newId = pushBack();
              if (!closeEditor) {
                 operationMode = EDIT_MODE;
                 edit( newId );
              }

          }else if( operationMode == EDIT_MODE ){
              updateUser();
          }

          if (closeEditor) {
             setVisible (false);
             dispose ();
          }

      }
      catch( Exception error ){
          // log error
          log.log( this, metaUtilsClass.getStackTrace( error ) );
          // show error message
          metaUtilsClass.showErrorMessage( this, error.toString() );
      }
    }



private void checkSecurity()
throws KExceptionClass
{

    long securityMask = configuration.getLongField( "system_user_mask" );

    if( (  securityMask & securitySwitchClass.OPEN_ALLOWED_SWITCH ) == 0 )
        throw new securityExceptionClass( "Access denied" );
```

```
    if( ( securityMask & securitySwitchClass.EDIT_ALLOWED_SWITCH ) == 0 )  {
        if( operationMode == CREATE_NEW_MODE ) throw new securityExceptionClass(
"Access denied" );

    }

}


private void setupTables( long id )
throws KExceptionClass
{


    // setup subbrowser, show child objects
    browser = new facturaBrowserClass(
            configuration, log, FacturasBrowserJTable,
            facturaBrowserClass.INVOICES_BY_CLIENT, id, this );

    browser.initializeTable();

    //setup sub toolbar
    newButton.addActionListener( browser );
    deleteButton.addActionListener( browser );
    editButton.addActionListener( browser );
    sortButton.addActionListener( browser );
    filterButton.addActionListener( browser );
    printButton.addActionListener( browser );
    refreshButton.addActionListener( browser );


    newButton.setEnabled( true);
    deleteButton.setEnabled( true);
    editButton.setEnabled( true);
    sortButton.setEnabled( true);
    filterButton.setEnabled( true);
    printButton.setEnabled( true);
    refreshButton.setEnabled( true);
}
```

Designing a Dialog using NetBeans



The important thing to remember here is to set the name property (b) of the different SWING widgets to the corresponding field in the corresponding PDC class, the sample_client in this case. The name of the variable (a), is not important. But don't confuse them.

If you need to display more information for more than one object, just change the newObject() and edit(). For example, let's say that, for informational purposes you want to show the client name in the invoice screen like this:

In this case, just paint the label, call it clientName, and change the newObject() and edit() as follows:

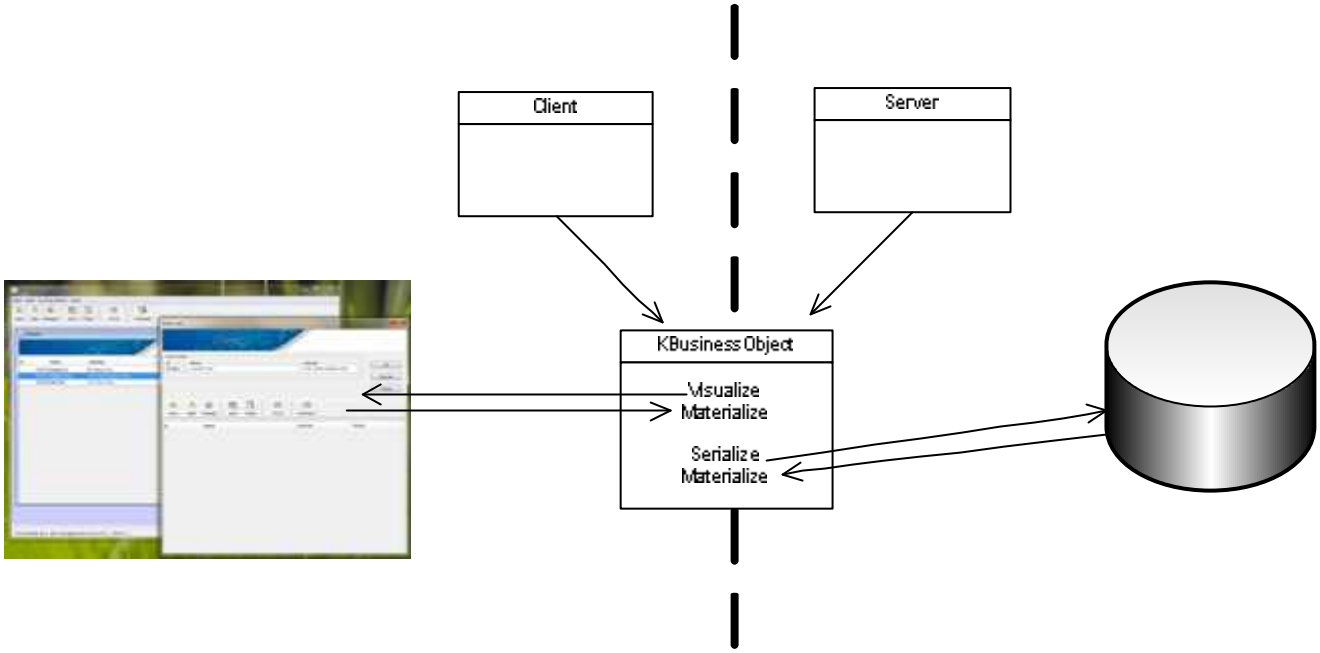
```
public void newObject( long keyValue )
throws KExceptionClass
{
    sample_factura factura = new sample_factura();

    persistentObjectManagerClass persistentObjectManager =
            new persistentObjectManagerClass( configuration, log );

        // ---------------------------------------------------------------------
        // display the client name
        sample_client client = new sample_client();
        persistentObjectManager.copy( keyValue, client );
        client.displayVisualize( getContentPane(),
sample_client.IGNORE_MISSING_FIELDS );
        // ---------------------------------------------------------------------

    persistentObjectManager.createNew( factura );

    factura.setClient_id( keyValue );

    factura.displayVisualize( getContentPane() );
}
```

```
public void edit( long id )
throws KExceptionClass
{
    sample_factura factura = new sample_factura();

    persistentObjectManagerClass persistentObjectManager =
            new persistentObjectManagerClass( configuration, log );

    persistentObjectManager.copy( id, factura );
    factura.displayVisualize( getContentPane() );

        // ---------------------------------------------------------------------
        // display the client name
        sample_client client = new sample_client();
        persistentObjectManager.copy( factura.getClient_id(), client );
        client.displayVisualize( getContentPane(),
sample_client.IGNORE_MISSING_FIELDS );
        // ---------------------------------------------------------------------


    setupTables( id );

}
```

As you can see, we simple load the client, according to the info already here, and just call visualize. The framework will get the object from the server and find the label to fill.

3.2.2. Using Dialog's Business Object Auto Field Validation and Formatting

The business objects have tools to automatically set the validation and some other parameters, like fields sizes to widgets. You just need to set some flags in the constructor of any business object.



Field Summary	
boolean	Editable Make the Whole Object read Only.
protected java.util.Map<java.lang.String,java.lang.Integer>	fieldMaxSize A pair fieldName / Size to set the max sizes of fields. The client will automatically set any widget (TextBox, Area) on visualize to match the max for the corresponding field.
protected java.util.Map<java.lang.String,java.lang.Integer>	fieldTypes Set the field types by adding fieldName / Types pairs. Used by client on visualize to automatically set the format and validation rules of text boxes.

	CURRENCY_TYPE to currency DATE_TYPE to date NUMERIC_TYPE numeric free form NUMERIC2_TYPE numeric 2 decimals NUMERIC5_TYPE numeric 5 decimals TIME_TYPE hour minute
protected java.util.Vector<java.lang.String>	readOnlyFields Set the fields to read only by adding fieldNames. Used by client on visualize to automatically disable the corresponding widgets.

Example, in the constructor of a business object:

```
public sample_factura()
throws KExceptionClass{

    super();

    // initialize fields

    // initialize fields
     fac_id = -1;
     fac_name = "";
     fac_status = FAC_OPEN;
     fac_total = 0;
     client_id = 0;


    //set max size
    fieldTypes.put( "fac_total" ,CURRENCY_TYPE  );
    fieldMaxSize.put( "fac_name" ,30  );
    readOnlyFields.add( "fac_status" );

    editable = true;


}
```

3.3. Using Tables / DataBrowsers: The DataBrowserBaseClass

Browsers are one of the most useful tools of the framework. There are some applications that use the framework to show only browsers and reports.

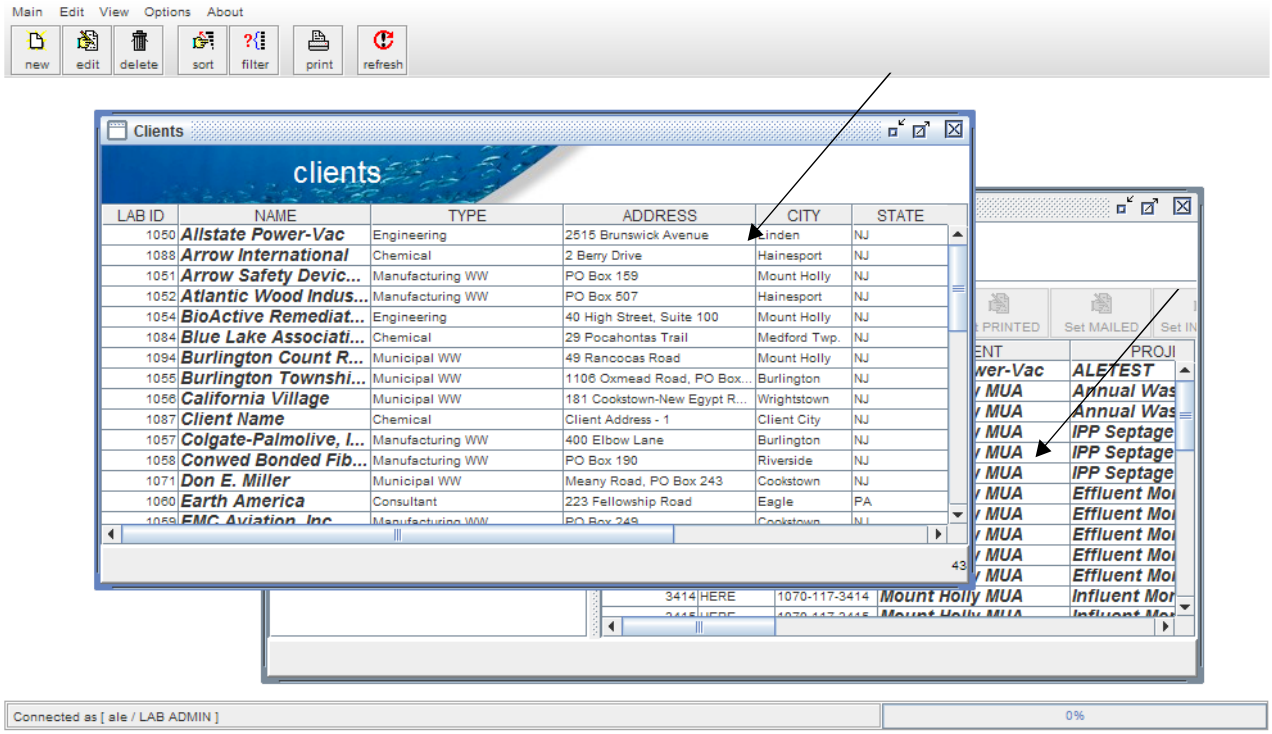
Browsers are dynamic, the user can change sort and filters, there can be totaling columns, colors fonts and this can be changed at runtime based on actual data, etc.

From the design of UI point of view, we always recommend to start from a browser, and then navigating to an edit dialog. A consistent UI using this flow will be very easy to navigate for complex applications. If you add tools to open edit dialogs directly here and there, and browsers which mix several objects with no dominant PDC object is very difficult to navigate and clearly indicates that the Domain design was not right. Check the tutorial on an example on how to do this and for some references. This framework, as well as for any other development, relies on a solid business object analysis.

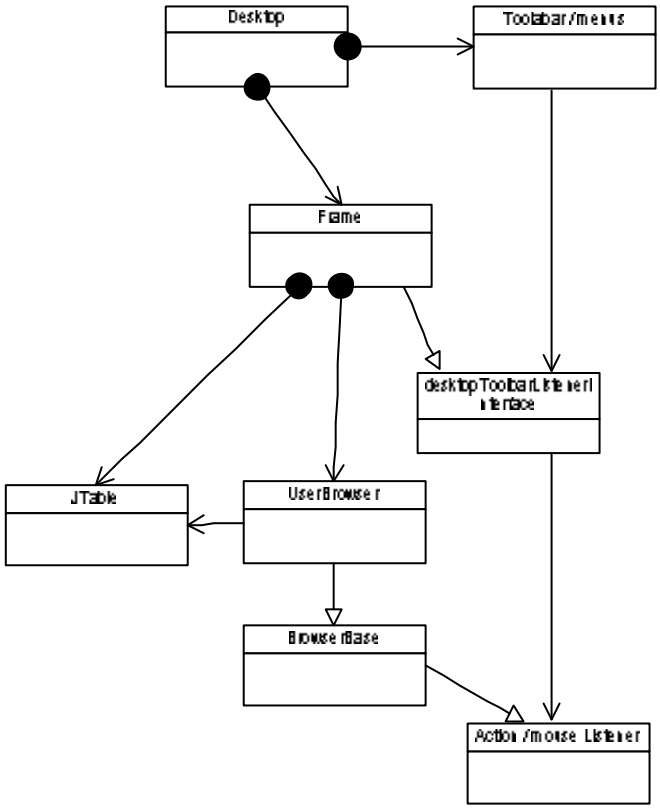
In general a browser can be inside a frame or a dialog.

A framed browser could be:

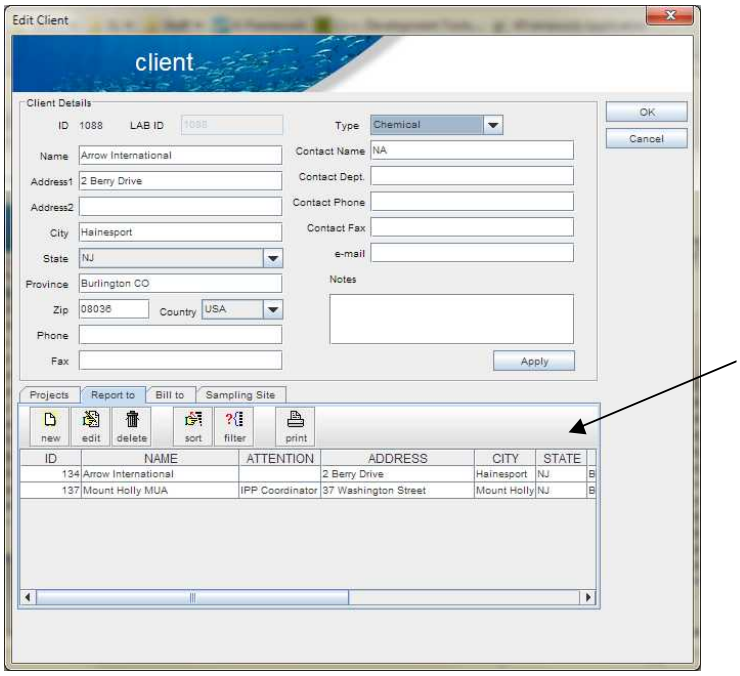
Framed Browsers



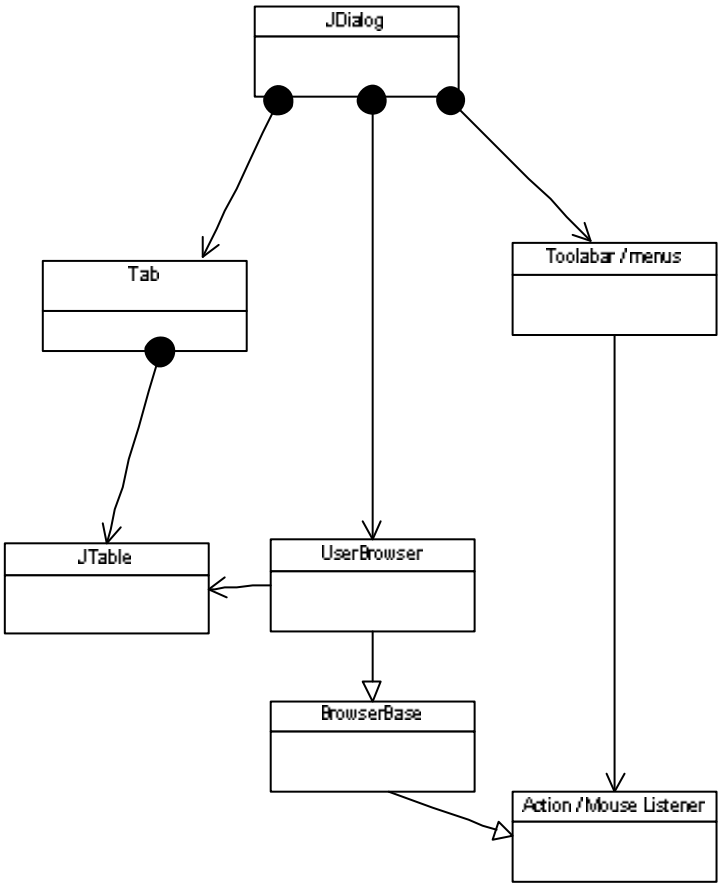
Framed Browser Hierarchy



A browser inside a dialog could be:



Internal Browser Hierarchy



Note that a browser will not load all records. It grabs only the necessary records depending on the view range, plus a small cache for smooth forward and backward scrolling. The user will "feel" he is traversing the whole set, even for millions of records, but in reality the browser is fetching only the required records in the background. You can jump to the end, middle and the browser will show the corresponding records, but it needs not load the whole set to go to the end of the table, for example.

3.4. DataBrowserBaseClass Reference

Constructor Summary	
DataBrowserBaseClass(KFrameWork.Base.configurationClass configurationParam, KFrameWork.Base.logClass logParam, boolean showKeyFieldParam, javax.swing.JTable tableParam, java.awt.Component componentParam) This constructor uses the standard configuration, log an optional key parameter used for dialog based browsers, the JTable to control and a visual parameter, normally (this), to be used as parent of dialog boxes.	

Method Summary	
void	actionPerformed(java.awt.event.ActionEvent event) This is fired for all actions handled. New, Edit, Delete, Filter, Sort and Print, plus any extra you define be registering this browser as listener in a control.
void	addCustomCriteria(java.util.List filters) Used to add a fixed and not changeable filters.
void	addCustomCriteria(java.lang.String filter) Used to add a fixed and not changeable filters.
void	adjustColumnBackgroundColor(java.lang.String columnName, java.awt.Color bgColor) Sets the background color for the column This method should be called after initializeTable method called
void	adjustColumnEditor(int column_index, javax.swing.table.TableCellEditor CellEditor) Using a JTextField, JCheckBox, or JComboBox as an Editor If you are setting the editor for a single column of cells (using the TableColumn setCellEditor method), you specify the editor using an argument that adheres to the TableCellEditor interface.
void	adjustColumnFont(java.lang.String columnName, java.awt.Font font) Set the Font for the column This method should be called after initializeTable method called
void	adjustColumnForegroundColor(java.lang.String columnName, java.awt.Color fgColor) Sets the foreground color for the column This method should be called after initializeTable method called
void	adjustColumnJustification(java.lang.String columnName, int alignment)

	Sets the justification for the column This method should be called after initializeTable method called. Aligments are: * LEFT SwingConstants.LEFT * CENTER SwingConstants.CENTER * RIGHT SwingConstants.RIGHT * LEADING SwingConstants.LEADING * TRAILING SwingConstants.TRAILING
void	adjustColumnType(java.lang.String columnName, int type)
	Set the data type for the column This method should be called after initializeTable method called. Types are used to format the column. Valid types are: public static final int CHARACTER = 0; public static final int NUMERIC = 1; public static final int NUMERIC2 = 2; public static final int CURRENCY = 3; public static final int DATE = 4; public static final int TIME = 5;
void	adjustColumnWidth(java.lang.String columnName, int width)
	Sets the width for the column This method should be called after initializeTable method called
void	**adjustHeaderRenderer**(tableHeaderRendererClass renderer) Save renderers and subscribe operations. Se below for a tutorial on headers and calculated fields.
void	bindCustomParameter(java.lang.String parameterName, long parameterValue) Adds a bind parameter for the SQL used for the dynamic SQL at runtime.
void	bindCustomParameter(java.lang.String parameterName, java.lang.String parameterValue) Adds a bind parameter for the SQL used for the dynamic SQL at runtime.

void	bindDefaultParameter(java.lang.String parameterName, java.lang.String parameterValue) Adds a bind parameter for the SQL used for the fixed filter that will not change at runtime.
void	clearCustomCriteria() Release all custom filters and bound arguments.
void	clearDefaultOrder() Clears the order by clause.
void	copyButtonActionPerformed() Called to handle actions. User can override to handle on his own.
long	dataBaseRowCount(boolean applyCustomFilters) Executes a count on the recordset. You can decide to apply the current filters, or execute over the default SQL.
void	deleteButtonActionPerformed() Called to handle actions. User can override to handle on his own.
void	displayRefresh() Reload the view
void	editButtonActionPerformed() Called to handle actions. User can override to handle on his own.
java.util.AbstractMap	evaluateOperation(java.lang.String SQLformula, boolean applyCustomFilters) Used to execute a SQL operation, like "sum(total)". The browser adds the rest. You can operate over the default SQL, or over the current filters defined by the user at runtime.
void	filterButtonActionPerformed() Called to handle actions. User can override to handle on his own.
java.util.Vector	getCheckSelectedRowKeys(int column_index) Return the key field values of current multi selected rows marked by Check Box.
void	getColumnNames(java.util.List nameList) Gets the table column names This method should be called after initializeTable.
int	getColumnType(java.lang.String columnName) Gets the table column type via column name This method should be called after initializeTable.
java.lang.String[][]	GetCustomCriteriaRowData()

	Returns the current where clauses as configured by user at runtime.
java.util.List	GetCustomOrderData()
	Returns the current sort as configured by user at runtime.
java.lang.String	getDefaultCriteria() Returns the current where clauses that are fixed.
java.util.List	getDefaultParameters()
	Get the bound parameters used by the default / fixed criteria.
javax.swing.JTable	getJTable()
java.util.Vector	getMultiSelectedRowKeys()
	Return the key field values of current multi selected table rows.
java.util.Vector	getMultiSelectedRowKeysAsStrings()
	Return the key field values of current multi selected table rows.
recordClass	getRecord(long OID)
	Returns in a recordClass the record corresponding to the given OID. Only for displaying records.
java.lang.String	getSelectedColumnVisualHeader()
	Return the visual header of current selected table column.
java.lang.String	getSelectedFieldValue(java.lang.String ColumnName)
	Return the table value at the selected row under the column name
long	getSelectedRowKey()
	Return the key field value of current selected table row.
java.lang.String	getTableDataAsHtmlTable()
	Get an HTM view of the current table
java.lang.String	getTableDataAsString() Get an String view of the current table
java.util.List	getTableDataHeaders()
	Get the columna headers

void	initializeSQLQuery(java.lang.String SQLSelect, java.lang.String DBTable, java.lang.String keyFieldParam)
	Initialize SQL statement This method is called only once after constructor, and before any other method.
void	initializeTable()
	Load the data into table
boolean	isLoaded()
	Flag indicating whether the initializeTable() has been called.
void	markChanged(long OID, java.lang.String changedParam)
	Called to handle actions. User can override to handle on his own.
void	mouseClicked(java.awt.event.MouseEvent event)
	Called to handle actions. User can override to handle on his own.
void	mouseClickPerformed(java.awt.event.MouseEvent event)
	Called to handle actions. User can override to handle on his own.
void	mouseDoubleClickPerformed(java.awt.event.MouseEvent event)
	Called to handle actions. User can override to handle on his own.
void	mouseEntered(java.awt.event.MouseEvent event)
	Called to handle actions. User can override to handle on his own.
void	mouseEnteredPerformed(java.awt.event.MouseEvent event)
	Called to handle actions. User can override to handle on his own.
void	mouseExited(java.awt.event.MouseEvent event)
	Called to handle actions. User can override to handle on his own.
void	mouseExitedPerformed(java.awt.event.MouseEvent event)
	Called to handle actions. User can override to handle on his own.
void	mousePressed(java.awt.event.MouseEvent event)
	Called to handle actions. User can override to handle on his own.
void	mousePressedPerformed(java.awt.event.MouseEvent event)

	Called to handle actions. User can override to handle on his own.
void	mouseReleased(java.awt.event.MouseEvent event)
	Called to handle actions. User can override to handle on his own.
void	mouseReleasedPerformed(java.awt.event.MouseEvent event)
	Called to handle actions. User can override to handle on his own.
void	newButtonActionPerformed()
	Called to handle actions. User can override to handle on his own.
void	notifyListeners(java.lang.String actionParam)
	Used to fire events manually.
void	**prepareCustomFieldsDBTransaction**(java.lang.String customFields, dbTransactionClientClass dbTransaction, boolean reflectCustomFilter)
	Gets the current SQL and sets a dbTransaction for query. This method should be called after initializeTable.
void	**prepareDefaultDBTransactionForTable**(dbTransactionClientClass dbTransaction)
	Gets the current SQL and sets a dbTransaction for query. This method should be called after initializeTable.
void	**prepareDefaultDBTransactionForTable**(dbTransactionClientClass dbTransaction, java.lang.String orderBy)
	Gets the current SQL and sets a dbTransaction for query. This method should be called after initializeTable.
void	**prepareTransactionWithBrowserSQL**(dbTransactionClientClass dbTransaction)
	Gets the current SQL and sets a dbTransaction for query. This method should be called after initializeTable.
void	print(java.lang.String report_name, java.lang.String report_owner)
void	printButtonActionPerformed()
	Called to handle actions. User can override to handle on his own.
void	refresh()

	Reload the table as the new setting applied.
void	refreshButtonActionPerformed()
	Called to handle actions. User can override to handle on his own.
void	**registerListener**(DataBrowserBaseClass.tableFillerListenerInterface listenerParam)
	Register to by notified for: NEW_ACTION EDIT_ACTION DELETE_ACTION SAVE_ACTION COPY_ACTION SORT_ACTION FILTER_ACTION REFRESH_ACTION PRINT_ACTION MOUSE_DBL_CLICK Used to fire a reload or a calculation.
void	resetToDefaults()
	Reload the table, clearing all custom filters.

void	**saveBrowserChanges**(tableFillerDataWriterInterface dataWriter)
	Called for Read/Write Browsers. A read/write browser has some columns setup with controls that allow writing. This returns the changes to an implementor of:
	`import KFrameWork.Base.*;`
	`public interface tableFillerDataWriterInterface {`
	`public void save( java.util.List fieldNames, java.util.List data ) throws KExceptionClass;`
	`}`

void	saveButtonActionPerformed()
	Action handler for the user. Not implemented by browser.

void	saveSQLOperation(java.lang.Object visualComponent, java.lang.String sqlOperation, boolean reflectCustomFilter)
	Used to BIND widgets. Like a total under the table. Or a count used for another process. On every reload the browser will execute the SQL on the browser and update the control. Valid controls are JLabel or JTextField.
void	saveSQLOperation(java.lang.Object visualComponent, java.lang.String sqlOperation, int dataType, boolean reflectCustomFilter)
	Used to BIND widgets. Like a total under the table. Or a count used for another process. On every reload the browser will execute the SQL on the browser and update the control. Valid controls are JLabel or JTextField. Use browser datatypes for formatting.
void	setBrowserReadWrite(boolean flag, java.lang.String tableAlias)
	Set the read write flag to true
void	setCacheSize(int size)
	Set the size of the record cache in records.
void	**setCellDisplayHook**(cellRenderingHookInterface cellDisplayHookParam) Register a cell render hook for rendering customization cell by cell and at runtime.
void	**setCellWriter**(cellWriterInterface cellWriterParam)
	Receive the writing events for the cell, validate and store the data. The browser does not cache changes.
void	setColumnNames(java.lang.String aliasName, java.lang.String fieldName, java.lang.String headerName) Set the visual column name from a column. Note that you need to pass the corresponding table ALIAS, to avoid ambiguity with to tables having a cell named the same.
void	setColumnNames(java.lang.String aliasName, java.lang.String fieldName, java.lang.String headerName, boolean colEditable)
	Set the visual column name from a column. Note that you need to pass the corresponding table ALIAS, to avoid ambiguity with to tables having a cell named the same. Also indicate if the cell is editable.

void	setColumnNames(java.lang.String aliasName, java.lang.String fieldName, java.lang.String headerName, boolean colEditable, boolean defaultRender)
	Set the visual column name from a column. Note that you need to pass the corresponding table ALIAS, to avoid ambiguity with to tables having a cell named the same. Also indicate if the cell is editable.
void	setCustomCriteriaRowData(java.lang.String[][] data)
	Add custom SQL where criteria.
void	setCustomOrder(java.util.List orderList)
	Set the order of the browser
void	setDefaultCriteria(java.lang.String criteria)
	Add custom SQL where criteria.
void	setDefaultOrder(java.lang.String order)
	Set the initial order of the browser
void	setDefaultParameters(java.util.List parameters)
	Set the parameters to bind to the default criteria.
void	setDoubleClickEnabled(boolean doubleClickEnabledParam)
	Indicate whether double click event is enabled. Usefull when browsers are read only.
void	setTableFont(java.awt.Font font)
	Sets the Font for the table This method can be called before or after initializeTable method called
void	setVisibleColumnCount(int visibleColumnCountParam)
	To hide columns, add them at the end and define here how many are visible.
void	softRefresh()
	Reload with current cache data, no access to server.
void	sortButtonActionPerformed()
	Action handler, the user can override.

3.4.1. Setting up browser's data access

Browsers are based on plain SQL, and use JDBC to access the database. To setup this access *the public void initializeTable()* function is provided.

For example:

```
public class facturaBrowserClass
extends DataBrowserBaseClass {


        // modo
        static public final int ALL_INVOICES = 0;
        static public final int INVOICES_BY_CLIENT = 1;
```

…

```
public void initializeTable()
throws KExceptionClass
{

        super.initializeSQLQuery(
            // 1 fields
            " fac.fac_id , fac.fac_name, fac.fac_status, fac.fac_total ",
            // 2 tables and joins
            " sample_factura fac ",
            // 3 key of primary PDC object
            "FAC_ID"
                );

        // mayusculas
        setColumnNames( "FAC", "FAC_ID", "id" );
        setColumnNames( "FAC", "FAC_NAME", "Name" );
        setColumnNames( "FAC", "FAC_STATUS", "STATUS" );
        setColumnNames( "FAC", "FAC_TOTAL", "TOTAL" );
```

```
        setDefaultOrder( "FAC_ID" );

        setDefaultCriteria(
            // grab name
            " fac.fac_status <> 'CANCELLED'  "
            );

        if( mode == INVOICES_BY_CLIENT ){

            // replace criteria
            setDefaultCriteria( " client_id = ? " );
            bindDefaultParameter(":client_id", parentID);
        }

        super.initializeTable();

        adjustColumnWidth( "id", 60 );
        adjustColumnWidth( "Name", 200 );
        adjustColumnWidth( "STATUS", 100 );
        adjustColumnWidth( "TOTAL", 100 );

        adjustColumnType( "TOTAL", CURRENCY );

        adjustColumnFont(  "id",  new Font( "arial", Font.PLAIN, 10  ) );
        adjustColumnFont(  "Name",  new Font( "arial", Font.PLAIN, 10 )  );
        adjustColumnFont(  "STATUS",  new Font( "arial", Font.PLAIN, 10 )  );
        adjustColumnFont(  "TOTAL",  new Font( "arial", Font.BOLD, 10 )  );

}
```

This is a basic minim setup for a browser's init function, with some customization but no totals, calculated fields or dynamic shading and colors.

3.4.2. Joining Tables

A browser shall be designed to manage one primary object, but you can join tables to add referenced fields and to complement the view. Joining tables is very easy.Follow these three steps in the browser:

```
public void initializeTable()
throws KExceptionClass
{

        super.initializeSQLQuery(

            // 1 fields
            " fac.fac_id , fac.fac_name, status.facstatus_status, fac.fac_total ",

            // 2 tables and joins
            " sample_factura fac " +
            " left join sample_factura_status status on status.facstatus_id = fac.facstatus_id  ",

            // 3 key of primary PDC object
            "FAC_ID"
                );

        // mayusculas
        setColumnNames( "FAC", "FAC_ID", "id" );
        setColumnNames( "FAC", "FAC_NAME", "Name" );
        setColumnNames( "STATUS", "FACSTATUS_STATUS", "STATUS" );
        setColumnNames( "FAC", "FAC_TOTAL", "TOTAL" );
```

3.4.3. Customizing the initial display of data

- To set the width of the columns use:

```
setColumnWidth( "Column Display Name", 150 );
```

Use this function in initializeTable(), after the call to super.initializeTable();

- To change a column's default colors

```
adjustColumnBackgroundColor( String columnNameDisplayName, Color bgColor)
adjustColumnForegroundColor( String columnNameDisplayName, Color fgColor )
```

Use these functions in initializeTable(), after the call to super.initializeTable();

To change the behavior of the table to fill all available space or force fixed column widths, change the corresponding JTable properties. See the swing JTable reference for more information.

➲ To change a column's font, font style and sizes:

```
setColumnFont(  "Asociado",  new Font( "arial", Font.ITALIC | Font.BOLD, 10 )
);
```

Use this function in initializeTable(), after the call to super.initializeTable();

➲ To change column's types for formatting:

```
setColumnType("Date", DATE);
setColumnType("SubTotal", CURRENCY );
```

This method should be called after initializeTable methodi s called. Types are used to format the column.

Valid types are:

public static final int CHARACTER = 0;

public static final int NUMERIC = 1;

public static final int NUMERIC2 = 2;

public static final int CURRENCY = 3;

public static final int DATE = 4;

public static final int TIME = 5;

Use this function in initializeTable(), after the call to super.initializeTable();

➲ To change a column's alignment

```
adjustColumnJustification( String columnName, int alignment )}
```

Use this function in initializeTable(), after the call to super.initializeTable();

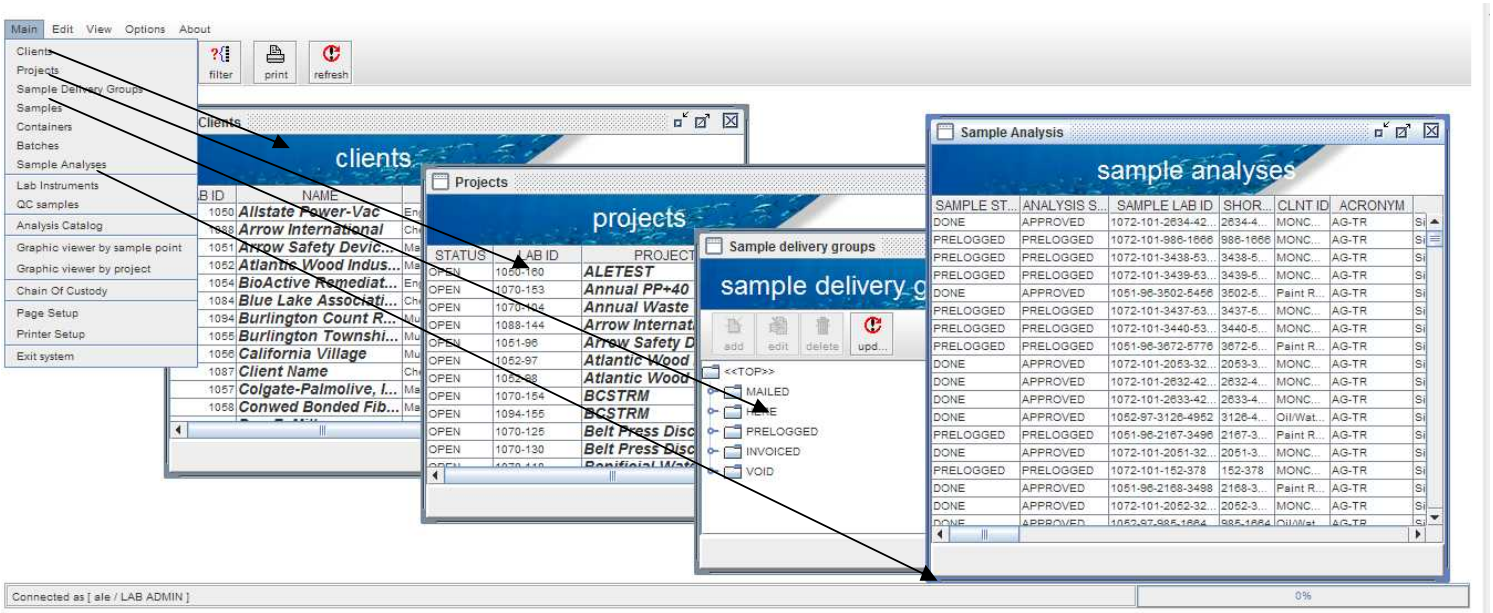
* LEFT SwingConstants.LEFT
* CENTER SwingConstants.CENTER
* RIGHT SwingConstants.RIGHT
* LEADING SwingConstants.LEADING
* TRAILING SwingConstants.TRAILING

Example setting up initial browser options

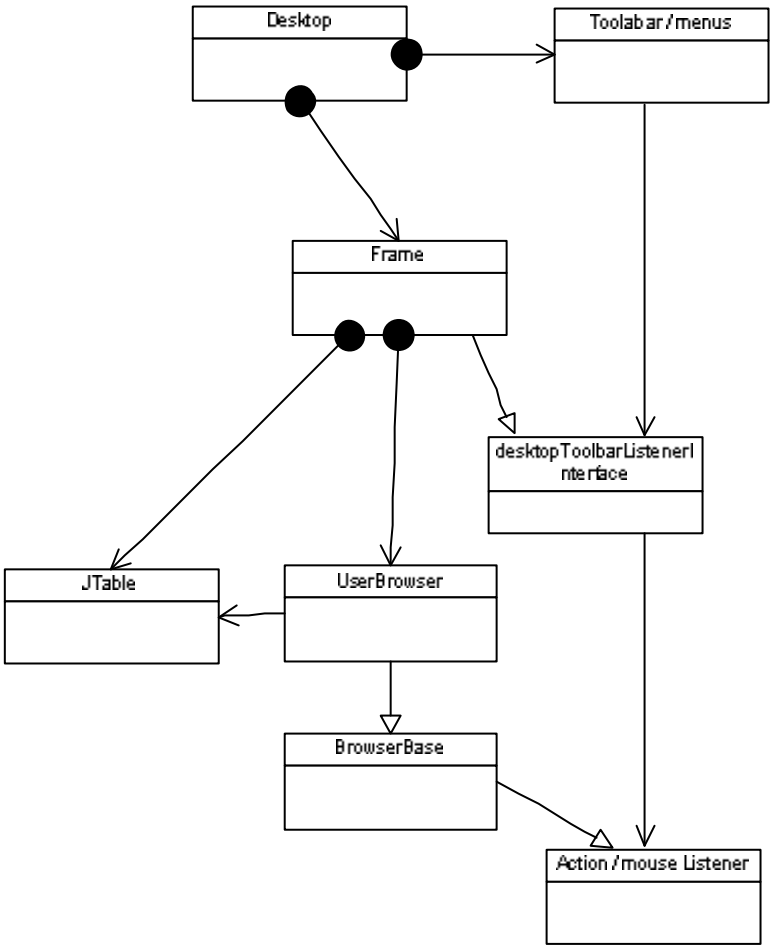
```
super.initializeTable();

adjustColumnWidth( "id", 60 );
adjustColumnWidth( "Name", 200 );
adjustColumnWidth( "STATUS", 100 );
adjustColumnWidth( "TOTAL", 100 );

adjustColumnType( "TOTAL", CURRENCY );

adjustColumnFont(  "id",  new Font( "arial", Font.PLAIN, 10  ) );
adjustColumnFont(  "Name",  new Font( "arial", Font.PLAIN, 10 )  );
adjustColumnFont(  "STATUS",  new Font( "arial", Font.PLAIN, 10 )  );
adjustColumnFont(  "TOTAL",  new Font( "arial", Font.BOLD, 10 )  );

}
```

3.4.4. Setting up a browser in a MDI frame

Following the recommended UI navigation design (See the tutorial) a browser is always the starting point of any action. Normally, you will have a main menu with different options, which reflect the top level business objects of your Problem Domain. Each option will open a browser for the corresponding business object, and from there the user selects an object, edits, and begins the navigation through the application. These initial frames are contained inside frames as follows:



A frame forms part of this hierarchy:



A frame holds a browser, optionally a tree view, and implements a DesktopToolBar Listener, which only forwards events to the browser. The Desktop handles the dispatching of the generic toolbar to the corresponding active frame.

To setup a framed browser (Assumes you already have a browser, if not see previous section):

1) Copy a frame from another package.
2) Fix:

    a) Setup the actions to take when focused, which buttons to enable and disable from the tool bar and the corresponding edit menu.

```
private void doFocusGained(){

        systemDesktop.enableDelete( true );
        systemDesktop.enableNew( true );
        systemDesktop.enableEdit( true );

        systemDesktop.enableSort( true );
        systemDesktop.enableFilter( true );
        systemDesktop.enableRefresh( true );
        systemDesktop.enablePrint( true );
    }


    private void doFocusLost(){
        systemDesktop.enableDelete( false );
        systemDesktop.enableNew( false );
        systemDesktop.enableEdit( false );
        systemDesktop.enableSort( false );
        systemDesktop.enableFilter( false );
        systemDesktop.enableRefresh( false );
        systemDesktop.enablePrint( false );
    }
```

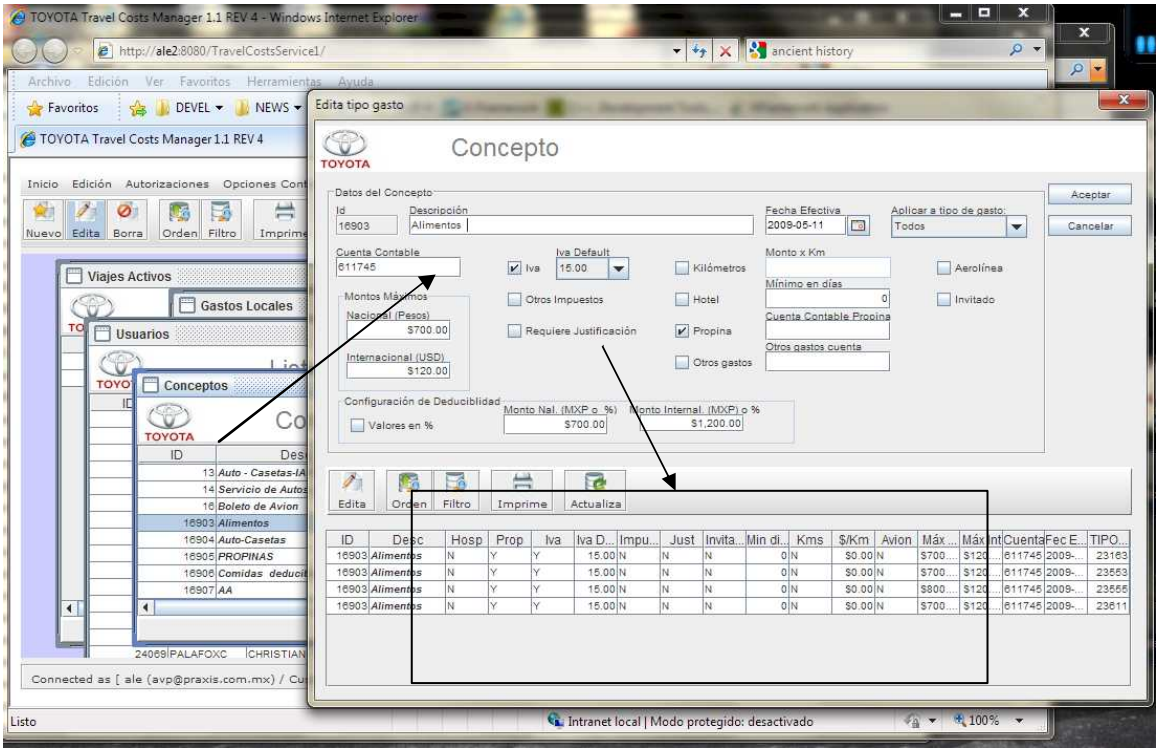
b) Define the browser to be loaded in the constructor:

```
/** Creates new form justificacionFrameClass */
public clientFrameClass(configurationClass configurationParam,logClass logParam,
{
    initComponents ();

    // uses
    configuration = configurationParam;
    log = logParam;
    systemDesktop = systemDesktopParam;

    // has defaulted
    browser =  new clientBrowserClass( configuration, log, justificationTable, t

    addInternalFrameListener( new InternalFrameListenerClass() );

    browser.initializeTable();


    log.log( this, "justificacion frame constructed successfully." );
}
```

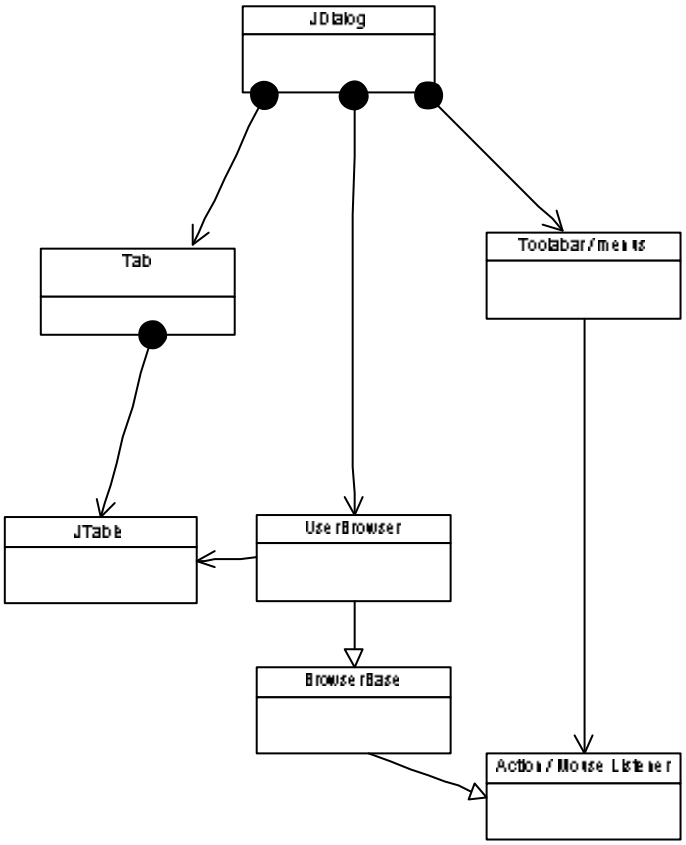
3) Ready, now we just need to add a menu item in the desktop to launch the frame and browser and test.

3.4.5. Hooking up inside an edit dialog

Following the recommended UI navigation design (See the tutorial) a browser is always the starting point of any action. Normally, you will have a main menu with different options, which reflect the top level business objects of your Problem Domain. Each option will open a browser for the corresponding business object, and from there the user selects an object and edits. The editor windows will allow the user to edit a Business Object, but this object might be a parent object of more child objects. Invoice to items, Samples to Analysis, Trips to expenses, for example. In this case, the edit window, and following the recommended design, will again present a browser to manage the next level in the business hierarchy. For example:



An internal browser follows this hierarchy:



Add an internal browser to an editor:

For each PDC object there is only one browser. This browser will be used for frames, as the previous section explained, and for internal browsers inside edit dialogs. A dialog might contain multiple internal browsers. So, for an internal browser, we just take the existing browsers for the PDC object and add some changes.

For this example we assume an invoice browsers was already created following the procedure explained before.

To add an internal view of invoices to a client editor we should:

1) Open the selected browser and add a new mode, since the browser shows all elements by default, we need now a restricted view, though we want to reuse the rest.

```
public class facturaBrowserClass
extends DataBrowserBaseClass {

        // modo
        static public final int ALL_INVOICES = 0;
        static public final int INVOICES_BY_CLIENT = 1;
```

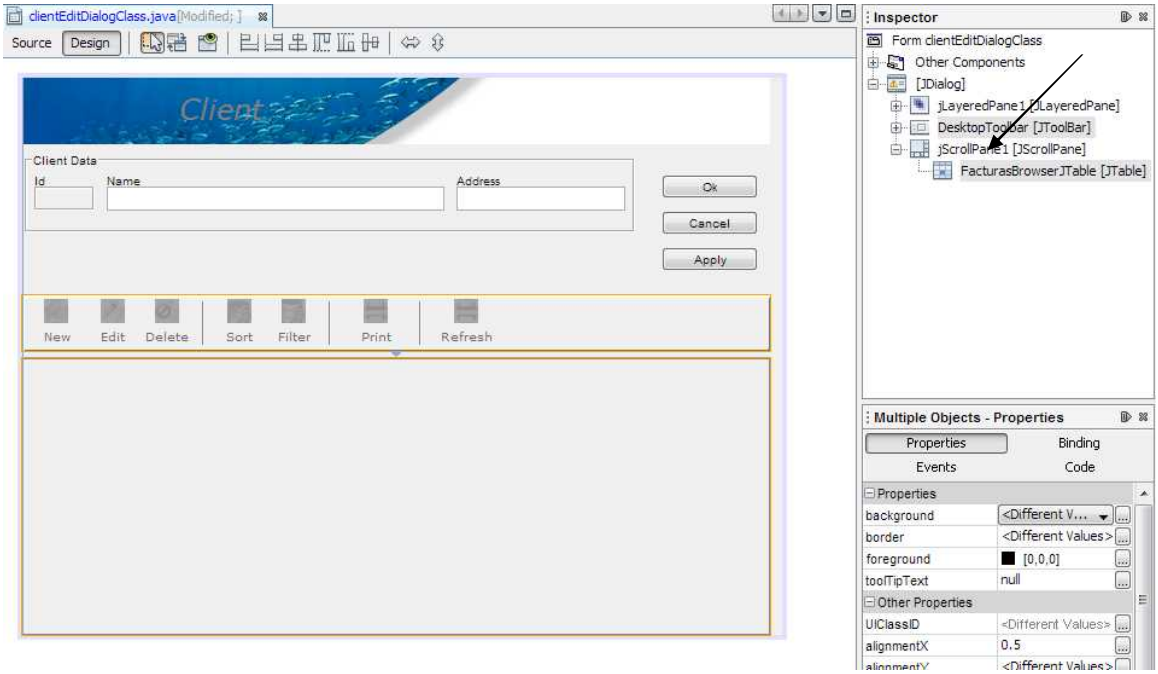
2) Add the corresponding where clause as a DefaultCriteria in the initializeFunction:

```
96
97          //-----------------------------------------------------------------
98          //Get daily category
99
100         super.initializeSQLQuery(
101
102             // 1 campos
103             " fac.fac_id , fac.fac_name, fac.fac_status, fac.fac_total ",
104             " sample_factura fac ",
105             "FAC_ID"    );
106
107         // mayusculas
108         setColumnNames( "FAC", "FAC_ID", "id" );
109         setColumnNames( "FAC", "FAC_NAME", "Name" );
110         setColumnNames( "FAC", "FAC_STATUS", "STATUS" );
111         setColumnNames( "FAC", "FAC_TOTAL", "TOTAL" );
112
113         setDefaultOrder( "FAC_ID" );
114
115
116         if( mode == INVOICES_BY_CLIENT ){
117
118             addDefaultCriteria( " client_id = ? " );
119             bindDefaultParameter(":client_id", parentID);
120         }
121
122         super.initializeTable();
123
124
125         adjustColumnWidth( "id", 60 );
126         adjustColumnWidth( "Name", 200 );
127         adjustColumnWidth( "STATUS", 100 );
128         adjustColumnWidth( "TOTAL", 100 );
129
130         adjustColumnType( "TOTAL", CURRENCY );
131
132         adjustColumnFont(  "id",  new Font( "arial", Font.PLAIN, 10  ) );
```

That's it for the browser. Everything else is reused, no changes. But you can change the scope of the "if" to change the fonts or the whole SQL. In this case we want to reuse everything, but only show the invoices for the current client's.

Note how the previously unused parameter parentID is now used. The frame version passed -1, but inside the client editor we will pass the PK of the parent client being displayed.

3) Now let's go back to the client edit window and visually add a toolbar and empty JTABLE. Copy them from an example dialog.



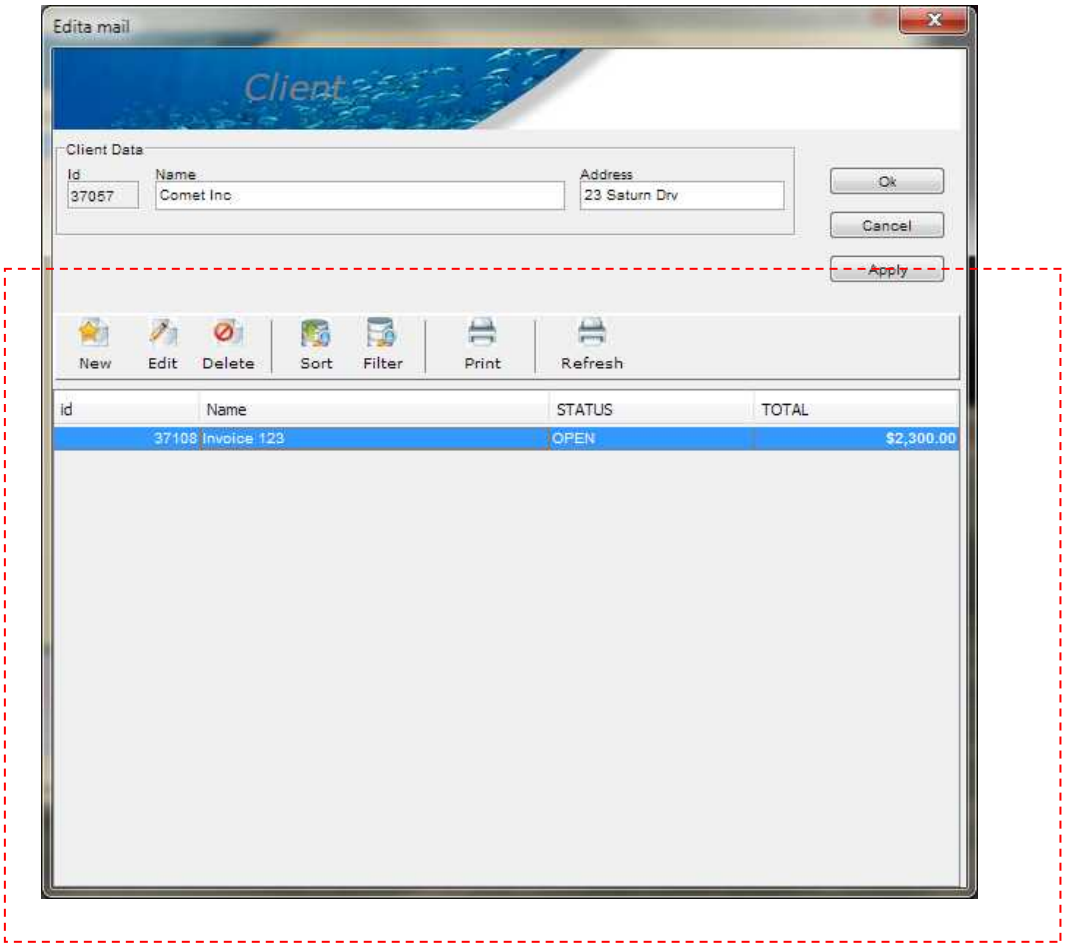
Note how the JTABLE is inside a JScrollPane to enable the use of scroll bars. The framework will handle the bars, but don't put a JTABLE with out its scroll pane.

4) Find the setupTables function in the Dialog and build the Browser and wire the events. No need to code events, we will reuse the ones already coded in the browser, we just need to hook it up. Note that are given the parented as an argument in the function.

```
private void setupTables( long id )
throws KExceptionClass
{

    //-----------------------------------------------------------



    browser = new facturaBrowserClass(
            configuration, log, FacturasBrowserJTable,
            facturaBrowserClass.INVOICES_BY_CLIENT, id, this );

    browser.initializeTable();

    //setup container button
    newButton.addActionListener( browser );
    deleteButton.addActionListener( browser );
    editButton.addActionListener( browser );
    sortButton.addActionListener( browser );
    filterButton.addActionListener( browser );
    printButton.addActionListener( browser );
    refreshButton.addActionListener( browser );


    newButton.setEnabled( true);
    deleteButton.setEnabled( true);
    editButton.setEnabled( true);
    sortButton.setEnabled( true);
    filterButton.setEnabled( true);
    printButton.setEnabled( true);
    refreshButton.setEnabled( true);
}
```

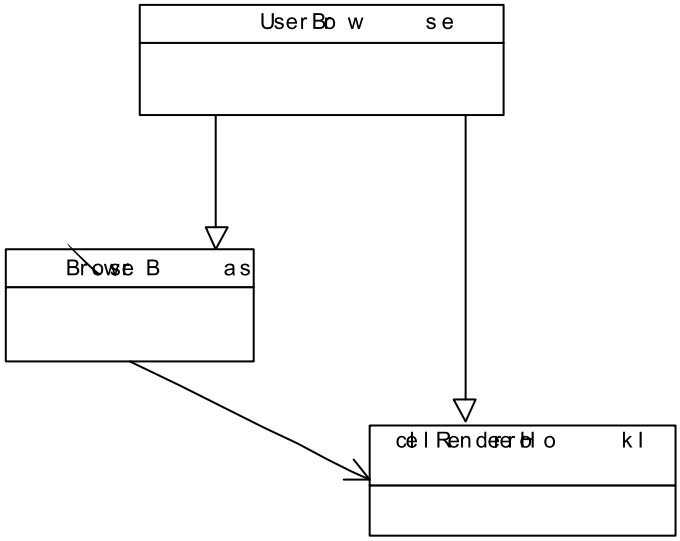
5) That's it, lets test. The tool bar should work automatically, and referential integrity maintained. Just note that the browser is disabled until the window switched to EDIT_MODE.

Client Edit, with Invoice View



3.4.6. Customizing the display of cells / rows on actual data at runtime

To customize cell's appearance at run time you need to implement a hook or callback that is called for every rendering of a cell. You are given cells information, including the whole row data for you to decide and change what you need, say color, bolding, or hide the value.



1) First, add the interface to your browser, so that it receives the calls:

```
public class facturaBrowserClass
extends DataBrowserBaseClass
implements cellRenderingHookInterface  // to customize the data at runtime
{
```

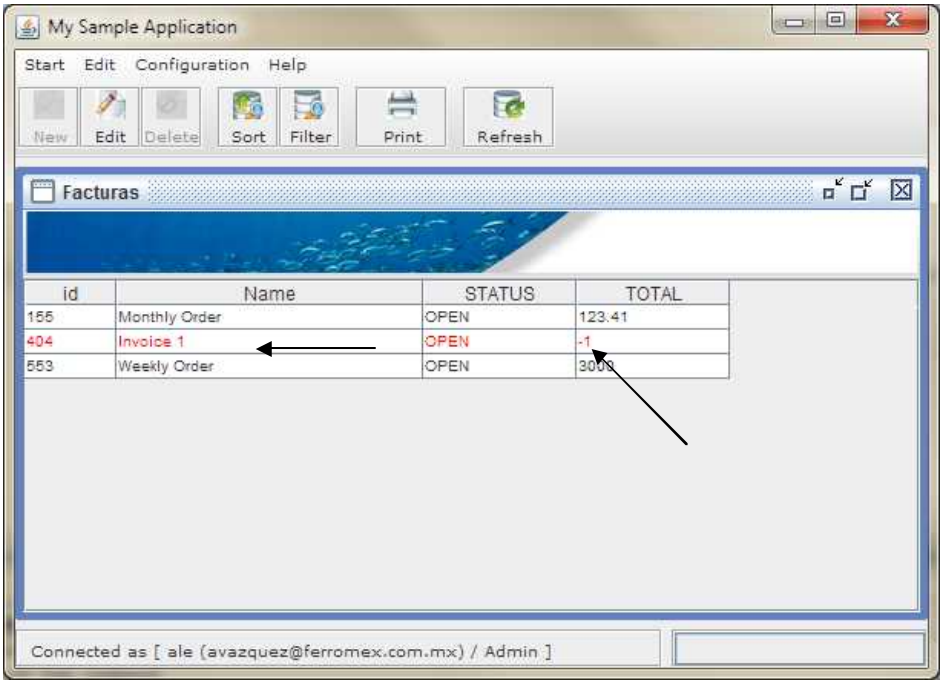
2) Register yourself to the underlying base browser. In the constructor is recommended.

```
public viajeBrowserClass(
        configurationClass configurationParam,
        logClass logParam,
        JTable tableParam,
        Component componentParam,
        int modeParam,
        long tipoViajeIdParam ) throws customExceptionClass
{

        // inherits
        super(
                configurationParam, logParam,
                true, tableParam, componentParam
        );

        // uses
        component = componentParam;
        systemResources = new JarResourcesClass( configuration, log );
        mode = modeParam;
        tipoViajeId = tipoViajeIdParam;

        // has

        // set
        setCellDisplayHook( this );

}
```

3) Then, just implement the hook ->

```
public boolean cellRenderingHook(
      int row, int col, // what cell are we executing for
      boolean isSelected, // is it currently highlited ?
      Component renderer, // here is the renderer, change it, or replaze it altogether
      String ColumnName, String value,  // data
      recordClass record, // the whole row data
      logClass log ) // the log used
      throws KExceptionClass
{
      boolean updateRenderer = false;


      // ------------------------------------------------------------------

      if( !record.getValueAt( 3 ).equals( "" ) ){  // total not null

          if( Double.parseDouble( record.getValueAt( 3 ) ) < 0 ){ // total negative, then

              renderer.setForeground( Color.red ); // its red, whole row, you could do only this column
              // by changing the first if for the specific columun: if( ColumnName.equals( "TOTAL" )...

              if( isSelected  ) renderer.setBackground( Color.yellow );

              updateRenderer = true;  // for performance. If you didnt change the renderer no bother.
          }


      }

      // ------------------------------------------------------------------

      return( updateRenderer );
```

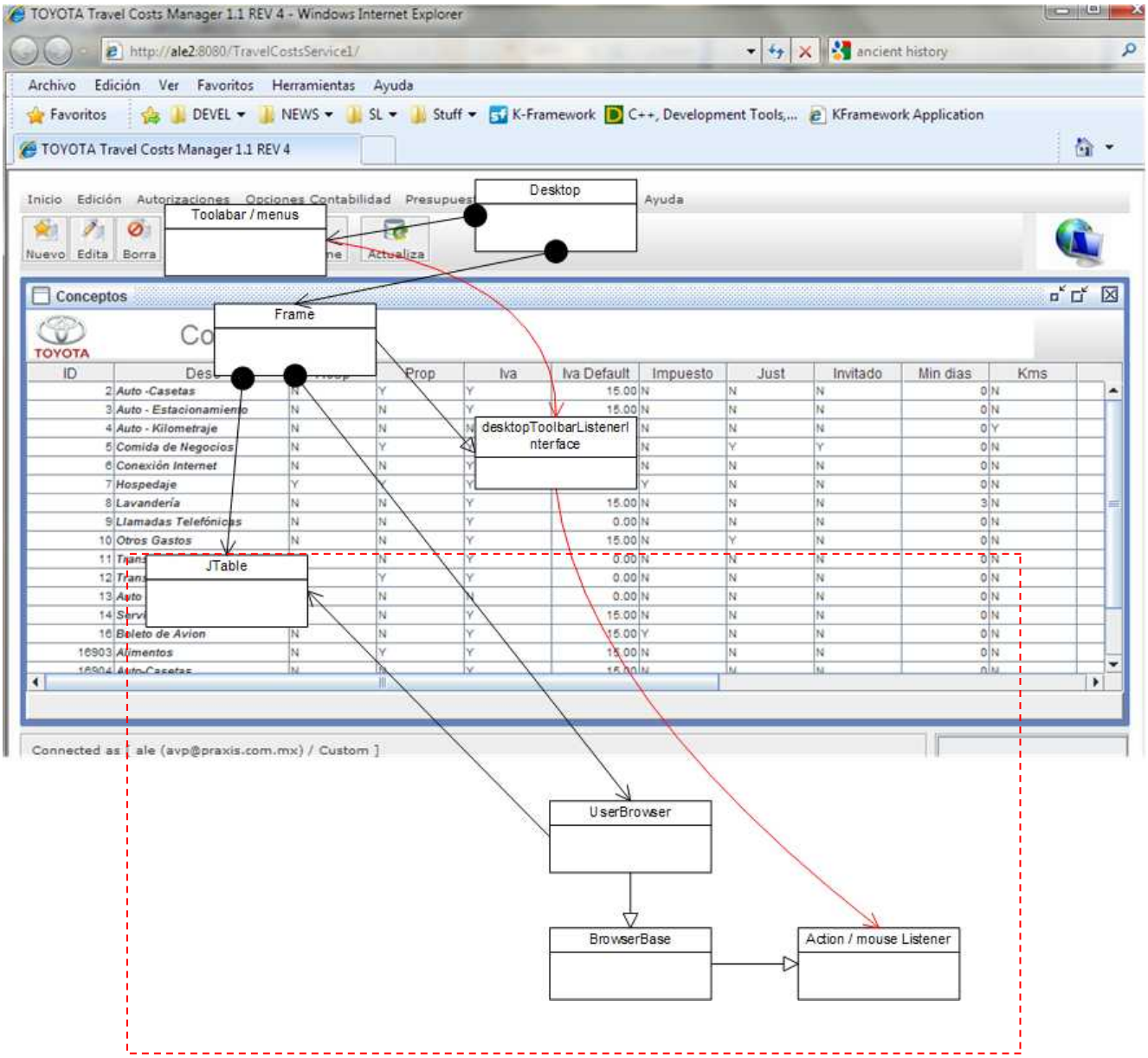
4) Try it ->


My Sample Application
Start Edit Configuration Help
New Edit Delete Sort Filter Print Refresh
Facturas
id Name STATUS TOTAL
155 Monthly Order OPEN 123.41
404 Invoice 1 OPEN -1
553 Weekly Order OPEN 3000
Connected as [ale (avazquez@ferromex.com.mx) / Admin]

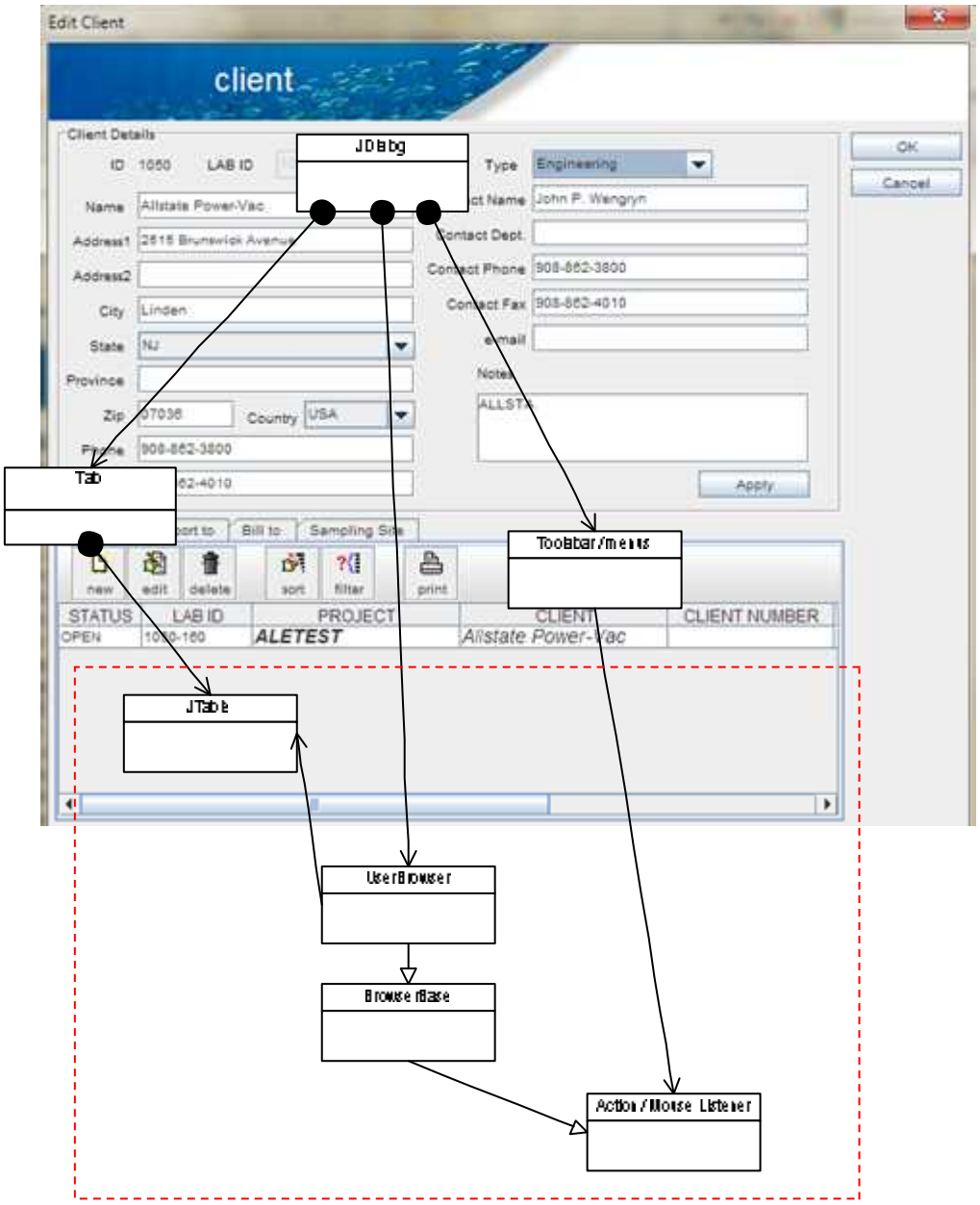
3.4.7. Handling Events in Browsers (Insert, edit, delete, filter, sort, print, refresh, mouse clicks, etc)

Browsers, no matter whether inside a frame or a dialog, will receive events; either by accepting a double click, or by a toolbar button click.

It is the same interface, whether an internal frame:



.. or an internal browser:



In either case, the browser will receive and dispatch events. There are user defined events, and automatic events.

3.4.8. User defined events

User defined events depend on the application and the PDC hierarchy. These are: New, Edit, and Delete.

1) New shall open the edit dialog of the corresponding object in NEW mode, and pass the ID of the parent, if any. -1 for a top hierarchy PDC object.
2) Edit shall open the edit dialog of the corresponding object in EDIT mode, and pass the ID of the desired object, always.
3) Delete shall call the POM directly and delete the object, upon confirmation.

Example New, Edit and Delete in a browser:

1) New

```
public void newButtonActionPerformed()
{

        try{

                log.log( this, "Openning create  screen..." );

                component.setCursor( new Cursor( Cursor.WAIT_CURSOR ) );

                clientEditDialogClass Dialog
                    = new clientEditDialogClass(
                        configuration, log,
                        metaUtilsClass.getParentFrame( component ),
                        -1,
                        mailEditDialogClass.CREATE_NEW_MODE );

                Dialog.setTitle("Factura" );

                log.log( this, "Openning create  screen completed." );

                Dialog.show();

                refresh();
                component.setCursor( new Cursor( Cursor.DEFAULT_CURSOR ) );
        }
        catch( Exception error  ){

                component.setCursor( new Cursor( Cursor.DEFAULT_CURSOR ) );
                // log error
                log.log( this, metaUtilsClass.getStackTrace( error ) );
                // show error message
                metaUtilsClass.showErrorMessage( component, error.toString() );
        }
```

In this case we create a new object from a top hierarchy browser, and we pass -1. If this was a browser from inside an editor, we would pass the editors current ID, or the parent ID, since an editor and internal browser shall implement the corresponding Parent-Child relationship in the Business Object Diagram.

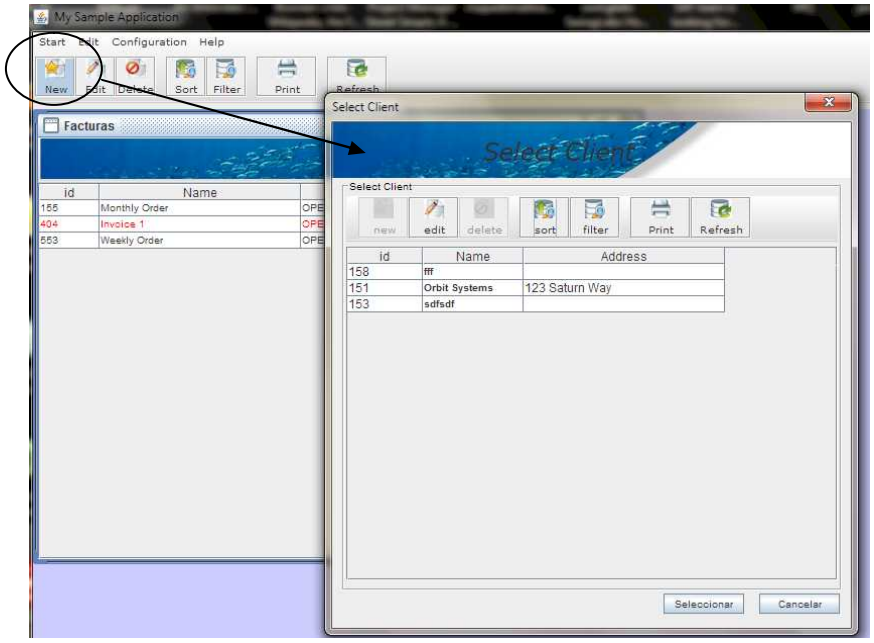
3.4.9. Using the automatic object select dialog

Some times we access a browser for a non top hierarchy object, from a sample listing for example, or an expense list, for which we access directly with out selecting a specific parent. In other words, some time we want to see all expenses for a user, no matter the trip. Or all samples for a client, no matter the project. In this case, when you click new from these views you do not have a parent, but it is required, so you need to ask for it. The framework provides a "parent selector". To use it, construct a browser for the parent objects and use de selectDialog in the newButtonActionPerformed as follows:

```
public void newButtonActionPerformed()

{

      try{
            // ------------------------------------------------
            // when not inside a client edit dialog
            if( mode == ALL_INVOICES ){

                // build a client browser
                   clientBrowserClass clientBrowser = new clientBrowserClass(
                            configuration, log, new javax.swing.JTable(), component );

                   clientBrowser.initializeTable();


                   selectDialogClass selector = new selectDialogClass(
                           configuration, log, clientBrowser, "Select Client", component );

                   // dont want to allow this, for exaple
                   selector.getNewButton().setEnabled(false);
                   selector.getDeleteButton().setEnabled(false);

                   parentID = selector.showDialog();

                   if( parentID == -1 ) throw new KExceptionClass( "You must select a client for th
            }

            // when not inside a client edit dialog
            // ------------------------------------------------
```

```
        // set category edit dialog
        log.log( this, "Openning create  screen..." );

        component.setCursor( new Cursor( Cursor.WAIT_CURSOR ) );

        facturaEditDialogClass Dialog
            = new facturaEditDialogClass(
                configuration, log,
                metaUtilsClass.getParentFrame( component ),
                parentID,
                mailEditDialogClass.CREATE_NEW_MODE );

        Dialog.setTitle("Factura" );

        log.log( this, "Openning create  screen completed." );

        Dialog.show();

        refresh();
        component.setCursor( new Cursor( Cursor.DEFAULT_CURSOR ) );
}
catch( Exception error  ){

        component.setCursor( new Cursor( Cursor.DEFAULT_CURSOR ) );
        // log error
        log.log( this, metaUtilsClass.getStackTrace( error ) );
        // show error message
        metaUtilsClass.showErrorMessage( component, error.toString() );
}
```

This will show an automatic selector. The provided selector as coded above:



The selector can return multiple IDs when need, just use showMultiSelectionDialog() instead of showDialog(), and it will return an a Vector of IDs.

2) Edit

```
public void editButtonActionPerformed()
{

        try{

            log.log( this, "Openning edit  screen..." );
            component.setCursor( new Cursor( Cursor.WAIT_CURSOR ) );

                clientEditDialogClass Dialog
                    = new clientEditDialogClass(
                        configuration, log,
                        metaUtilsClass.getParentFrame( component ),
                        getSelectedRowKey(),
                        mailEditDialogClass.EDIT_MODE );

                Dialog.setTitle("Edita mail");

                log.log( this, "Openning    completed." );

                Dialog.show();

                refresh();
                component.setCursor( new Cursor( Cursor.DEFAULT_CURSOR ) );
        }
        catch( Exception error  ){

                component.setCursor( new Cursor( Cursor.DEFAULT_CURSOR ) );
                // log error
                log.log( this, metaUtilsClass.getStackTrace( error ) );
                // show error message
                metaUtilsClass.showErrorMessage( component, error.toString() );
        }
```

Note that edit is also fired by double clicking on a record, not only from the button in the toolbar. If you want to disable editing, don't forget to, aside from disabling the buttons as explained above, disable the browser's double click event. To do that, just call this in the initialize of the browser:

```
setDoubleClickEnabled(false);
```

3) Delete

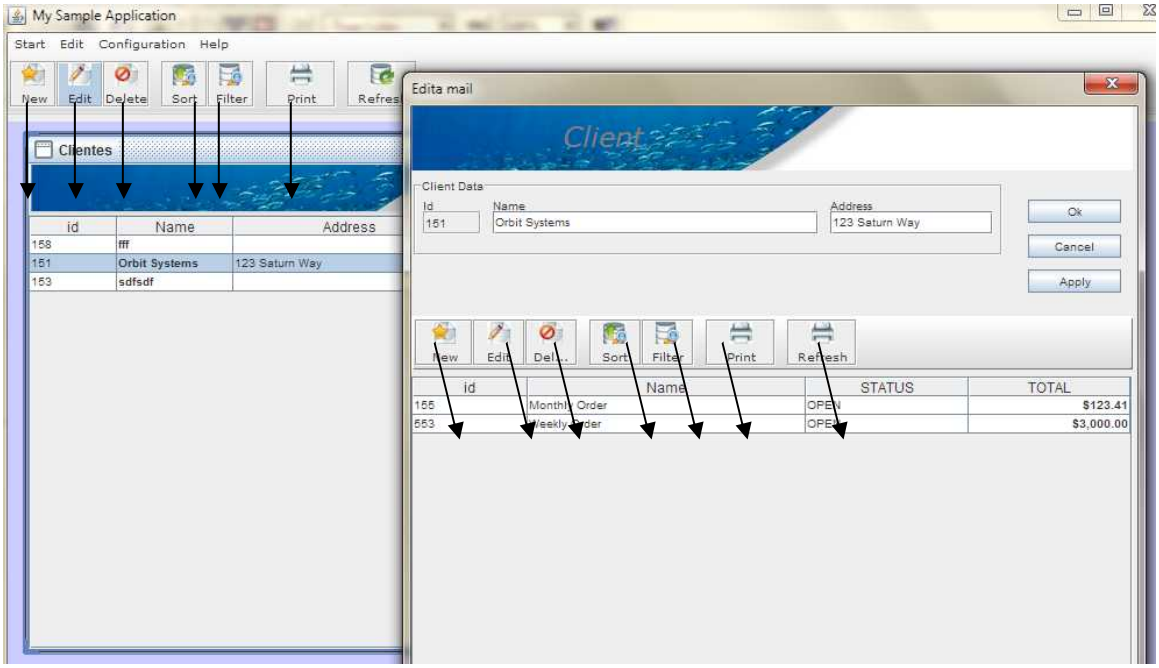
```
  @Override
public void deleteButtonActionPerformed()
{


      try{

          long id = getSelectedRowKey();

          String message = "Esta seguro de borrar este mail ?";

          if( metaUtilsClass.showConfirmationMessage( component, message ).equals

              persistentObjectManagerClass persistentObjectManager
                  = new persistentObjectManagerClass( configuration, log );

              persistentObjectManager.delete( id, sample_client.class.getName() ).
          }

          refresh();
       }
      catch( Exception error  ){

              // log error
              log.log( this, metaUtilsClass.getStackTrace( error ) );
              // show error message
              metaUtilsClass.showErrorMessage( component, error.toString() );
      }
```

Note that the server handles referential integrity and calculations, so you shouldn't need to do anything else for a delete.

3.4.10. Automatic Events:

Implementation is provided for filter, sort, print, refresh, but you can override these functions. These functions are also handled in the browser, and the framework takes care on calling the right browser depending on frame's focus.

Events sent to browsers



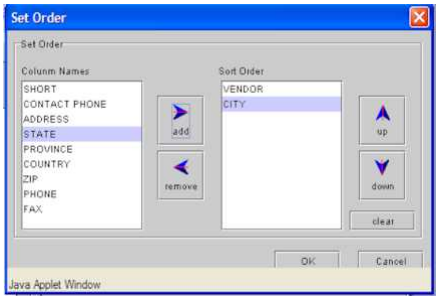
These events are handled automatically in the browser by the following functions you can override:

```
sortButtonActionPerformed();
filterButtonActionPerformed();
refreshButtonActionPerformed();
printButtonActionPerformed();
```

You don't need to configure any of these, they will automatically recognize the data and setup themselves automatically.

1) To sort a browser, click the sort button. 

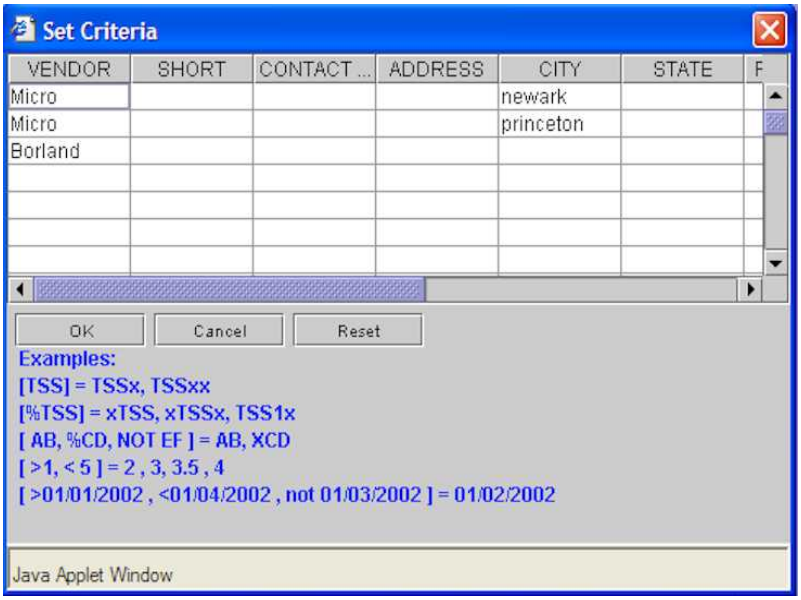
The "set Order" dialog will appear. Select the field or fields to sort by and click OK.



You don't need to configure any of these, the handler will automatically recognize the data and setup itself automatically.

2) To filter or query the browser, click the filter button. 

The "Filter" dialog will appear. Enter the selection criteria, and click OK.



To remove filter criteria from a browser, click the filter button again and then click the "reset" button.

You can have as many criteria lines as needed. Each line will define on set of criteria.

More lines of selection criteria will select rows in addition of the previous filters.

To add multiple criteria on a single row, separate the rules with a comma.

You can use the >,< and NOT operators.

Example for field VENDOR

```
Micro, Borland, not Micros
```

Example for numeric field

```
> 10 , < 100
```

For operators to work, make sure the corresponding column was defined as numeric in the browser's initialize.

You don't need to configure fields; the handler will automatically recognize the data and setup itself automatically.

3) All browsers come with automatic basic printing

You don't need to configure fields; the handler will automatically recognize the data and setup itself automatically.

4) The refresh action will reload the browser

3.4.11. Using Dynamic Data Filtering

3.4.12. Binding labels to the browser for totals and counts etc.

3.4.13. Using Headers with calculations

3.4.14. Using Read / Write Browsers

3.5. Using the advanced tree viewer

3.6. Using Menus

3.7. Using the Desktop toolbar and toolbars in general

3.8. Using SWING Widgets / KWidgets

3.8.1. Using Data Bound Text Fields, Labels and Numeric Boxes

3.8.2. Using Constant Data Combo Boxes

3.8.3. Using Data Bound Combo Boxes

3.8.4. Using Data Bound Check Boxes

3.8.5. Using Data Bound Calendars

4. Reporting Engine Reference

4.1. Print Job Setup

4.2. Setting up the printer and page

4.3. Report Sections

4.3.1. Text tools

4.3.2. Drawing Tools

4.3.3. Picture Tools

4.3.4. Bar Code Tools

4.3.5. Charting with JFree Graphics

5. General Tools Reference

5.1.1. Using General Message Boxes

5.1.2. Using IBM Alpha Works Charting

5.1.3. Using JFree Charting

5.1.4. Using System Exit

5.1.5. Using User manager

5.1.6. Using Audit Trial Viewer

5.1.7. Using Mailer Viewer

5.1.8. Displaying help and manuals

Appendix A

A.1 Coding Conventions

A.2 Exception Handling and the KExceptionClass

A.3 Using the ConfigurationClass

A.4 Using the KFramework LogClass

A.5 Using the Text File manager Tool

Appendix A GNU License

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the

library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes
a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the

users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse

engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any

particular circumstance, the balance of the section is intended to apply,
and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add
an explicit geographical distribution limitation excluding those countries,
so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE

IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS