Programmer's Guide

Version 7.0
April 2013

We believe that a good documentation is a key point for a good product.
Help us to improve it.
Do you think that there is something missing or wrong with this document?
If you have doubts with something that is undocumented volunteer to write a good explanation that can be useful to other users.
Mail your suggestions to translations@hipergate.org

1 Introduction

Application layers

Hipergate is divided into five layers:

1. Data model and Stored Procedures
2. Compiled Java Classes
3. Java BeanShell Scripts
4. Servlets / JSP
5. Client JavaScript

This guide is structured bottom-up, starting with the data model until reaching the final presentation layer. The purpose is to give a complete vision of how to write functional extensions from ground up.

2 Data Model

Hipergate is based on a data model written to be portable among different relational database management systems (DBMS).

Portability is combined with special care for taking advantage of each DBMS native features and optimizations.

Where to find data model source SQL/DDL

Original data model sources are stored as resources inside a hipergate.jar file.
They must be searched at **com.knowgate.hipergate.datamodel** package. Usually it is not necessary to do anything directly with these original sources, except when writing a build for a new DBMS.

The data model is divided into logical submodules that represent parts of the whole application. Some basic submodels are always required (security, categories, …), others are optional.

Conventions

Naming conventions

Database object names are lower case.

ORACLE

Object names will be automatically converted to upper case when the data model is created on Oracle. hipergate libraries are case insensitive with regard of object names but other applications or custom SQL queries may not be.

Tables and Views

Tables Tables start with prefix "k_".

Fields The following prefixes are used, depending on the concept represented by field:

"gu_" : Global Register Identifier (GUID).
"id_" : Object Identifier Unique only at this table scope.
"tx_" : Text.
"de_" : Object Description.
"tl_" : Object Title.
"dt_" : Date.
"nm_" : Object Name.
"tp_" : Object Type.
"od_" : Ordinal position inside a list.
"tr_" : Translated text for a particular language.
"pg_" : Sequence.
"bo_" : Boolean Flag (1=true, 0=false).
"is_" : Yes/No Flag (1=yes, 0=no).
"nu_" : Object count.
"ny_" : Elapsed years.
"nd_" : Elapsed days.
"pr_" : Price (monetary units).
"im_" : Amount (monetary units).
"pct_" : Percentage.

Example (SQL Server):

```
CREATE TABLE k_companies (
gu_company    CHAR(32)    NOT NULL,          /* Company Unique Id*/
dt_created    DATETIME    DEFAULT GETDATE(), /* Register Creation Date*/
nm_legal      VARCHAR(50) NOT NULL,          /* Company Legal Name */
gu_workarea   CHAR(32)    NOT NULL,          /* Workarea GUID */
nm_commercial VARCHAR(50)     NULL,          /* Comercial Name */
dt_modified   DATETIME        NULL,          /* Date Modified */
id_legal      VARCHAR(16)     NULL,          /* Legal Identifier */
id_sector     VARCHAR(16)     NULL,          /* Sector Code Id. */
id_status     VARCHAR(30)     NULL,          /* Status:active,bankrupt..*/
id_ref        VARCHAR(50)     NULL,          /* External Identifier */
tp_company    VARCHAR(30)     NULL,          /* Company Type */
de_company    VARCHAR(254)    NULL,          /* Activity Description */

CONSTRAINT pk_companies PRIMARY KEY (gu_company),
CONSTRAINT u1_companies UNIQUE (gu_workarea,nm_legal))
```

Views Views start with preffix “v_”.

Procedures Procedures and Functions start with preffix “k_sp_”.

Triggers Triggers start with preffix “k_tr_”.

Data Types

For making the data model portable, only a restricted data type subset is used.

Base Type	SQL Server	Oracle	PostgreSQL
Signed 16 bits integer	SMALLINT	NUMBER(6)	SMALLINT
Signed 32 bits integer	INTEGER	NUMBER(11)	INTEGER
Fixed length ASCII characters	CHAR	CHAR	CHAR
Varying length ASCII	CHARACTER VARYING	CHARACTER VARYING	CHARACTER VARYING
Varying length Unicode chars	NVARCHAR	VARCHAR2 *	VARCHAR
Undefined length Unicode chars	NTEXT	CLOB	TEXT
Date and Time	DATETIME	DATE	TIMESTAMP
Exact Decimal	DECIMAL(n,m)	NUMBER(n,m)	DECIMAL(n,m)
Double Precision floating point	FLOAT	NUMBER	FLOAT
Long binary	IMAGE	BLOB	BYTEA

ORACLE There is no partial Unicode support for Oracle using the NVARCHAR2 columns. The whole database must be created with Unicode encoding. For a guide about creating Oracle databases, refer to Creating an Oracle Database and enabling multilingual support with Unicode databases.

☞ The data access package included with hipergate is generic and does not impose any restriction on the data types that may be used. But hipergate testing only includes the above data types. Any other data types are not granted to work.

Global Unique Identifiers (GUIDs)

Almost always, tables use a common format for the primary key : CHAR(32)

Source code often reference these identifiers as GUIDs

GUID value is granted to be unique for identifying a register at a database and even at other databases generated independently.

There are several ways of creating a GUID:

1. **From the command line** :
   `$>java com.knowgate.misc.Gadgets uuidgen`

2. **From Java Code** :
   `com.knowgate.misc.Gadgets.generateUUID();`

3. **With Microsoft Visual Studio** :
   Program `UUIDGEN.EXE` included with Visual Studio common tools may be used for creating GUIDs for DOS command line.

4. **Using Win32 API**:
   Use function `UuidCreate()` from dynamic link library `RPCRT4.DLL`. Declarations obtained from VisualBASIC are :

```
Type TUUID
  Data1 As Long
  Data2 As Integer
  Data3 As Integer
  Data4 As String * 8
End Type
```

```
Declare Function UuidCreate Lib "RPCRT4" (uuid As
TUUID) As Long
```

5. **From JavaScript**
   Use a function for generating a String of 32 random characters:

```
function createUuid() {
  var chrset = "abcdefghjkmnpqrstuvwxyz23456789";
  var guid = "";
  for (var i=0; i<32; i++) {
    guid += chrset.charAt (Math.round
                           (Math.random()*30));
  } // next
  return guid;
}
```

6. **From PL/pgSQL**
   Use a function for generating a String of 32 random characters:

```
CREATE FUNCTION k_sp_create_guid () RETURNS CHAR AS '
DECLARE
  c INT;
  g CHAR(32);
BEGIN
  g:='''';
  c:=0;
  LOOP
    g:=g||chr(CAST (floor(random()*25)+97 AS INT));
    c:=c+1;
    EXIT WHEN c=32;
  END LOOP;
  RETURN g;
END;
' LANGUAGE 'plpgsql'
```

Creation and Modification Dates for registers

Many tables use fields `dt_created` and `dt_modified` for creation and last modification date stamps.

Creation dates must not be written from the application level as they always have default values at the database. Thus, creation dates always contain the date when the register was created, obtained from the database server.

Modification dates are written from Java code. Field `dt_modified` is optional and if it is set to NULL, then the register has never been modified since it was created. Because creation date is written from the database

but modification date is written from the application server, a potential mismatch may occur between these two dates if the database server and application server clocks are not synchronized.

Active registers flags

For marking registers as active or inactive field `bo_active` is used.
A value of 1 means that the register is active, 0 means inactive.
Deactivating a register is a useful way of performing logical deletion.

Hierarchical data walkthrough

A common problem in relational database design is how to represent hierarchical data. In hipergate this happens at Category tree, Projects and Thesauri.

There is no single and optimal solution for this problem.

If the maximum number of depth levels is known and limited, the best choice is usually to list all ancestors of an object inside that object register itself (this is the way hipergate thesauri works) or use a limited number of JOINs in search queries.

If there is no limit for the number of permitted levels, the best solution depends upon the DBMS used.

Three most common tactics are:

1ª) Use temporary tables for holding intermediate results during tree walkthrough. This tactic is used, for example, at Transact/SQL version of stored procedure `k_sp_cat_expand`.

2ª) Use a memory stack pile for expansion process.

3ª) Make recursive calls to the same procedure. See for example, `k_sp_cat_del_grp` Transact/SQL version.

4ª) Manipulate the tree as a set of nested subsets. This technique provides the best way of finding all child trees from a node without a previous expansion process. A good reference may be found at Joe Celko book *SQL for Smarties*, or in his article at Intelligent Enterprise (Oct 2000) http://www.intelligententerprise.com/001020/celko1_1.shtml.

5ª) Write an expansion process as an external C or Java routine. See, for example, method browse() of com.knowgate.hipergate.Category.

Oracle
Oracle has a native query syntax for hierarchical expansion:
START WITH ... CONNECT BY ...

PostgreSQL
It is possible to find a C routine for expansion from Joe Conway at:
http://developer.postgresql.org/docs/pgsql/contrib/tablefunc

Microsoft SQL Server
There is no native mechanism for expansion.

Data model versioning

Installed data model version may be found at field `vs_stamp` of table `k_version` from build 1.1.0.

Segmentation by Workareas

A field of type GUID is used for identifying the Workarea to which belong data contained within tables shared by several Workareas.

This field is called `gu_workarea` or, sometimes, `gu_owner`.

Filtering by a Workarea is the method used by hipergate for showing only data for a particular Workarea to a given User.

☞ When developing multi-entity applications proper usage of `gu_Workarea` field is absolutely critical.

The available Workareas list is stored at **k_workareas** table.

This table stores values that never change over time.

There are to classes of value tables: 1st) global values tables
2nd) tables with values per Workarea.

Global static values tables start with prefix "`k_lu_`".

Tables

k_lu_languages ISO 639 language codes.

k_lu_countries ISO 3166 country codes.

k_lu_currencies ISO 4217 currency codes.

k_lu_states States per country.

k_lu_status Generic status for products and documents.
Predefined status are:
-2 → Retired.
-1 → Corrupt.
0 → Pending of Approval.
1 → Active.
2 → Blocked.

k_lu_prod_types MIME archive types.

k_lu_cont_types Container Types.
Standard values are.

1	Local File System (protocolo file://)
2	Hiperlink (protocolo http://)
3	Hiperlink (protocolo https://)
4	Remote Files (protocolo ftp://)
5	Data Base (protocolo jdbc://)
6	Lotus Notes (protocolo notes://).
100	Physical Location (protocolo ware://)

These tables contain values that are only visible from a particular Workarea.

Each *lookup table* is always associated to a *base table*. Following a standardized mechanism for lookup creation, it is possible to re-use the

Lookup tables have the same name as their corresponding base table but ended with "`_lookup`".

Example:

Base Table

```
CREATE TABLE k_companies
(
gu_company    CHAR(32) NOT NULL,  /* Company GUID */
gu_workarea   CHAR(32) NOT NULL,  /* WorkArea GUID */
nm_legal      VARCHAR(50),
id_legal      VARCHAR(16),
id_sector     VARCHAR(16),  /* Sector Code Lookup */
id_status     VARCHAR(30),  /* Status Lookup */
tp_company    VARCHAR(30),  /* Company Type Lookup */
de_company    VARCHAR(254),

CONSTRAINT pk_companies PRIMARY KEY(gu_company)
)
```

Lookup Table

```
CREATE TABLE k_companies_lookup
(
gu_owner   CHAR(32) NOT NULL,    /* Workarea GUID */
id_section VARCHAR(30) NOT NULL, /* Field name at base table */
pg_lookup  INTEGER  NOT NULL,    /* Value Progressive Id */
vl_lookup  VARCHAR(255) NULL,    /* Actual lookup value */
tr_es      VARCHAR(50)  NULL,    /* Translated value (spanish) */
tr_en      VARCHAR(50)  NULL,    /* Translated value (english) */

CONSTRAINT pk_companies_lookup PRIMARY KEY (gu_owner,id_section,pg_lookup),
CONSTRAINT u1_companies_lookup UNIQUE (gu_owner,id_section,vl_lookup)
)
```

Example values for a Workarea Lookup

id_section	pg_lookup	vl_lookup	tr_en
id_sector	1	ID72	COMPUTER SCIENCE
id_sector	2	ID53	RETAIL
id_status	1	A	ACTIVE
id_status	2	B	BANKRUPT
tp_company	1	CLIENT	CLIENT
tp_company	2	COMPETITOR	COMPETITOR
tp_company	3	PARTNER	PARTNER

☞ If the standard facilities for visualizing and updating lookups is to be used, then each new lookup table must be created following the previous example. Lookup tables must comply with the following restrictions:

1. there must be a field called `gu_owner` that identifies the Workarea to which data at each register belongs.
2. there must be a field called `id_section` that identifies the field at the base table to which lookup register is referred.
3. for each value there must be a field called `pg_lookup` that identifies an incremental sequence.
4. there must be a field called `vl_looukp` that contains the actual lookup value in a language independent way.
5. there must be translated labels fields with prefix "tr_" + language code stored at `k_lu_languages` table.
6. at base table, any value that has a lookup must be of type VARCHAR(1...254).

User defined attributes

hipergate has a feature for creating attributes (fields) defined by user for each base table.

There are 2 stages in the creation of new user defined attributes :

1st) Meta-data definition: Define which attributes will be created on which table.

2nd) Fill Data:. At table maintenance forms, fill the proper date for new attributes.

k_lu_meta_attrs This table keeps the information (meta-data) of which user defined attributes exist for each base table and Workarea. table k_lu_meta_attrs does not contain the data itself but only the definition.

Example values for table k_lu_meta_attrs

gu_owner	nm_table	id_section	tp_attr	pg_attr	max_len	tr_es
[*GUID*]	k_companies_attrs	tx_owners	1	1	100	Owners
[*GUID*]	k_contacts_attrs	bo_xmas	1	1	1	Send Xmas Card

`gu_owner` Workarea to which attribute definition belongs.

`nm_table`	Base table name.
`id_section`	Name of attribute.
`tp_attr`	Attribute Type. Currently only Type 1 (Plain Text) is allowed.
`pg_attr`	Ordinal progressive value for attribute.
`max_len`	Attribute data maximum length.
`tr_es`	Translated label for spanish.

How to add your own custom columns to the standard data model

There are basically three different options for adding new columns to the data model:

1. With user defined attributes metadata.
2. Adding the new columns directly in the standard tables with an ALTER TABLE command.
3. Creating a subrecords table inheriting the primary key from a standard table.

Each option has is advantages and disadvantages.

Using user defined attributes is the easiest, but it is also the most difficult to manage when trying to write data reports or forcing foreign key constraints. User defined attributes mix data and metadata making it complicated to perform simple SQL queries on the stored values.

If a new column is added to a standard table, the application will still be able to read and write that table without any problem. hipergate automatically detects the table structure and adjusts to it. After adding a new column the web server must be rebooted because hipergate maintains a cache in memory of the data model structure and this cache is only read once on application startup. Be careful to add your custom column to the edition and storage forms that read and write the register from and into the database. You must write every column of a record each time you update it. If you add a new column and fill it but do not add it to the user interface forms then the values will be lost when the user modifies the register via the web interface.

Creating new tables for custom values is the safest way of achieving the goal of managing additional data associated to a hipergate base entity. But it is also the one that requires more work since you must read and write from two tables instead of one. Many of the hipergate native classes have their delete routines externalized to stored procedures in the data base. When you delete a register, it is not the Java code which performs the actual deletion but the PL/SQL or Transact SQL code. Thus you may add new tables without recompiling the hipergate.jar library, but by just altering the stored procedures code.

Data Localization

The data model uses several ways of keeping translated data for different languages.

In some cases, such as static values tables and lookup tables, there is a field for each supported language in a single register. These cases imply that for adding a new language it is necessary to physically alter these tables.

In other cases, labels for categories, translations are placed vertically, with a register per language. In these cases it is not necessary to alter the physical data model when adding a new language.

Categories Submodel

hipergate has a generic category tree.

This tree is made-up of a hierarchy of nodes (categories). Each category may contain a set of heterogeneous objects inside.

The tree allows that labels for nodes (categories) are presented to the end-user in different languages, providing the possibility of creating multi-language directories without a need to duplicate nodes.

Tables and Views

Tables

k_categories Category List.

`gu_category` Category GUID.

`gu_owner` User owner of Category (from `k_users`).

`nm_category` Category Name. This column is an alternative primary key. It is designed for facilitating the action of finding a category directly with a user friendly name. Category names are also used for generating directory paths. It is recommended that only ASCII-7 characters are used for

category names. The application automatically fills `nm_category` by calling method `com.knowgate.hipergate.Category.makeName()`, refer to the JavaDoc API for more details.

`bo_active` Tells whether category is active. This flag is designed for showing or hiding categories.

`dt_created` Register creation date.

`dt_modified` Register modification date.

`nm_icon` First icon name for Category. The first icon is used for closed nodes images (not expanded at category tree when visualizing it). Only the icon file name is stored -not the full path- At the standard setup images for icons are at `/web/images/tree` and `/web/skins/skin/nav` depending on whether fixed or per skin icons are used.

`nm_icon2` Second icon name for Category.. The first icon is used for opened nodes images.

`id_doc_status` Initial status for products and documents added to category. Values are taken from `k_lu_status` table :

1 Active

0 Pending of Approval

k_root_cats Root categories. The standard database has only one root category for all others, but additional root categories may be created..

k_cat_labels Translated labels for each language.

`gu_category` Category GUID.

`id_language` Language Code from .`k_lu_languages`.

`tr_category` Category label for language.

`url_category` Category URL (optional)

	k_cat_tree	Category Tree. Keeps the parent-child relationship between categories. The model allows several parents to exist for a category, although the standard user interface limits parents to one.
	k_cat_expand	Pre-expanded list of child and grand child categories of a given one. This table is an intermediate cache that the application automatically maintains.
	k_x_cat_objs	Relationship between a category and objects it contains. Based on the fact that all objects of hipergate are identified by a GUID, this table uses a single column called `gu_object` that keeps references to any register of another table having a GUID has its primary key.
	`gu_category`	Category GUID.
	`gu_object`	Object GUID.
	`id_class`	Numeric identifier of class to which contained object belongs. It must be a value from column `id_class` of table `k_classes`. It is also the value of static property `IdClass` from each Java class of hipergate.
	`od_position`	Relative position of object inside category. It is the caller program responsibility to assign positions. Several objects may share the same position, in that case order of appearance is not defined.
	`bi_attribs`	Bits mask for additional containership information.
Vistas	**v_cat_tree**	Category tree but showing category names instead of GUIDs for easier reading.
	v_cat_tree_labels	View of the category tree with translated labels. SQL Server : Indexed views with schema binding are used for faster access to tree labels.
	v_cat_group_acl	Permissions mask for each group and category.

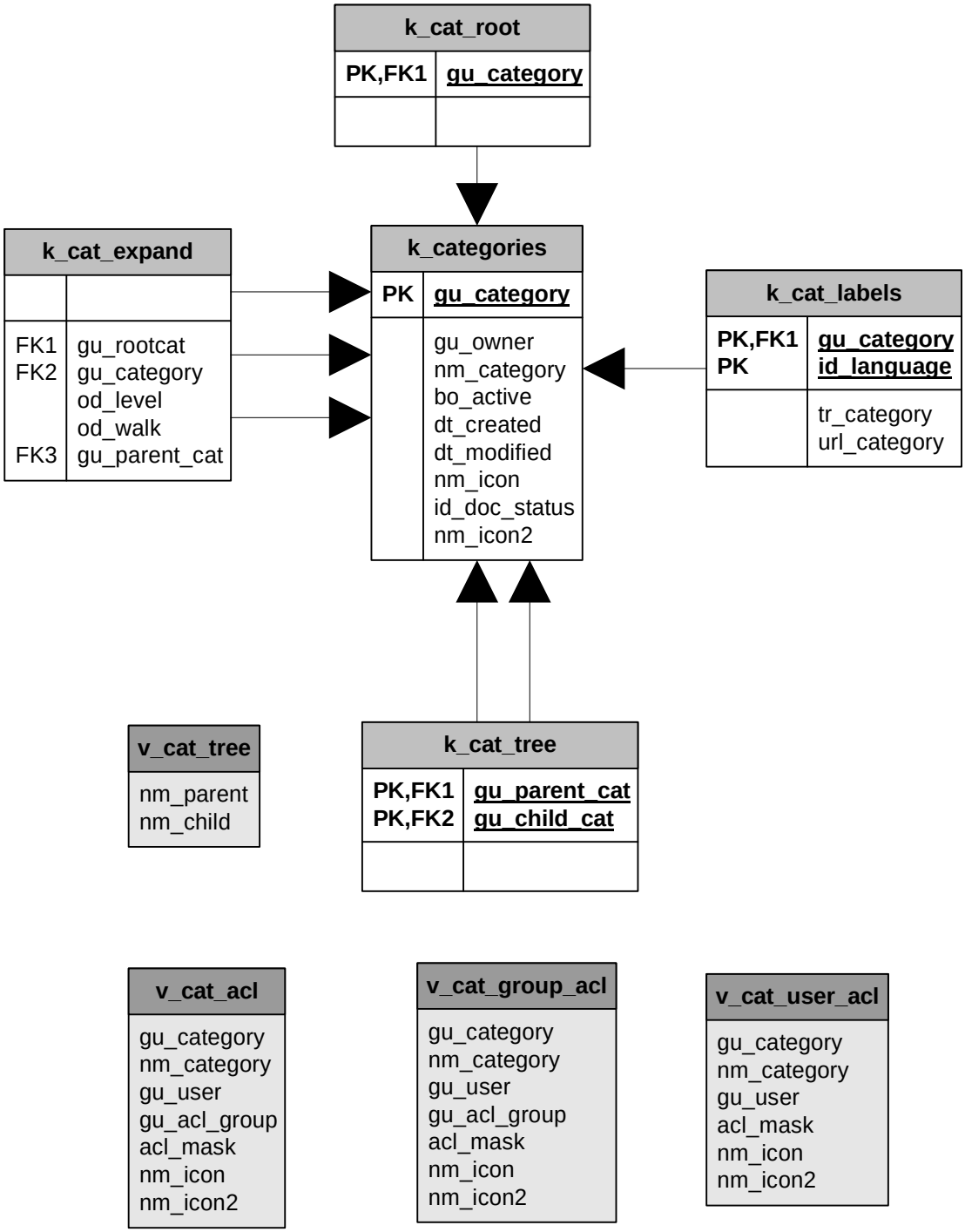
v_cat_user_acl Permissions given on each category directly to a specific user.

v_cat_acl Union of v_cat_group_acl and v_cat_user_acl. This is the total permissions set for a user in a Category, taking into account permissions granted through group membership and granted directly.

How to create categories directly on the model

Creating a new category is not a trivial task; the following steps must be accomplished to do so:

1. Create a GUID for the new Category.
2. Insert main register at `k_categories` table.
3. If it is a root category, insert a register in `k_cat_root`. If it is not root, insert register in `k_cat_tree`.
4. Insert translated labels in `k_cat_labels`.
5. Asign permissions per User Group by inserting registers in `k_x_cat_group_acl`.
6. Asign permissions per User by inserting registers in `k_x_cat_user_acl`.



hipergate relies on a role based security model.
This model may be complemented or substituted by another one, but the native model is already designed for providing most needed security features.

Security constraints restrict user actions in two ways :

1st) restricting things that he can or cannot do on each application depending on his role. There are 4 fixed roles: administrator, power user, user and guest. Things permitted and forbidden for each role are hard-coded into the application and cannot be changed without rewriting JSP source code.

2nd) restricting access over objects on which he does have permissions.

Domains

A Domain is the basic administrative unit in hipergate. Each domain contains

Each domain contains Workareas, Users Groups and Individual Users.

Tables

k_domains	List of all domains.
`id_domain`	Domain Numeric Identifier. Each Domain is identified by an integer number. Numbers from 0 to 2048 are reserved and should not be used.
`dt_created`	Creation Date.
`bo_active`	1 is Domain is Active 0 if Domain is inactive.
`nm_domain`	Domain Name. Domains Names are unique. This is an alternative primary key for `k_domains` table.

`gu_owner`	GUID of Administrator User of Domain. A Domain can have several administrators but one of the administrators must always be directly referenced in k_domains. This is done to avoid accidental deletion of all domain administrators, leaving the domain impossible to administer.
`gu_admins`	GUID of Administrators Group for Domain.
`dt_expire`	Date when domain expires (for time-bombed accounts).

Workareas

A Workarea is a finer grain division than Domain. Its purpose is to separate data from different workgroups inside an organization. Conceptually the Domain may represent an organization and Workareas may represent departments inside the organization.

Table	**k_Workareas**	List of Workareas. Each Workarea belongs to a Domain. A Domain can contain un unlimited number of Workareas.

Roles

Applications from hipergate suite recognize only 4 predefined roles: administrator, power user, user and guest.

Users and Groups

There can be an unlimited number of user groups, although only 4 standard groups are created in the default set-up.

Each user can be member of any number of groups.

Permission for a user is the sum of permissions given to him for each group. There is no “deny permissions” feature.

Tables	**k_acl_users**	Users per Domain.
	`gu_user`	User GUID.
	`id_domain`	Domain to which user belongs.
	`tx_nickname`	Short Name (nickname). Nicknames are unique for each Domain, although they may be repeated across different domains. The pair [`id_domain` , `tx_nickname`] is an alternative primary key for `k_users`.
	`tx_pwd`	User password.
	`bo_change_pwd`	1 if user may change its own password, 0 if not.
	`bo_active`	1 if user is activate, 0 if is inactive.
	`len_quota`	Disk quota currently in use by user.
	`max_quota`	Maximum allowed quota for user.
	`tp_account`	Type of user billing account.
	`id_account`	Billing account identifier.
	`dt_last_update`	Last update of user information.
	`dt_last_visit`	Last date when user logged in.
	`dt_cancel`	Date when user login was cancelled.
	`tx_main_email`	Main e-mail. This field is an alternative primary key
	`tx_alt_email`	Alternative e-mail.
	`nm_user`	User Name.
	`tx_surname1`	User First Surame or Middle Initial.
	`tx_surname2`	User Second Surame.
	`tx_challenge`	Question for retrieving lost password.

`tx_reply`	Reply to lost password question.
`dt_pwd_expires`	Date when password expires.
`gu_category`	GUID of initial category (home) for user.
`gu_workarea`	GUID of default Workarea to which user will be directed if he logs in using just his e-mail and password. A user may have simultaneous access to many Workareas but can only be logged simultaneously to one of them.
`nm_company`	Name of Company for User
`de_title`	User Job or Title.
`id_gender`	Gender: M= Masculine, F=Feminine
`dt_birth`	Birth date.
`ny_age`	Age.
`marital_status`	Marital Status.
`tx_education`	Studies level.
`icq_id`	ICQ Identifier.
`sn_passport`	Identity document or social security number.
`tp_passport`	Identity document type.
`tx_comments`	Comments.

k_acl_groups	User Groups per Domain.
k_x_group_user	Relationship between Users and Groups.
k_x_cat_group_acl	Permissions of Groups over Categories.
k_x_cat_user_acl	Permissions of Users over Categories.

hipergate suite is divided into applications. Each application handles 4 different user roles. Combining User Groups, Workareas and Applications is necessary because users for a given Workarea have a different application set available than users of other Workareas. For example, maybe it is required that a salesforce have access to contact manager, project tracker, and intranet tools whilst the support department only has access to contact manager and project tracker.

Tables

k_apps Installed Applications.

Preloaded values for standard version

Id. Application Bit	Description
10	Bug Tracker
11	Duty manager
12	Project Manager
13	Mailwire
14	Web Builder
15	Virtual Disk
16	Sales
17	Collaborative Tools
18	Marketing Tools
19	Directory
20	Shop
30	Configuration

k_x_app_workarea Groups assigned to each role for each Workarea.

`id_app` Application Numeric Identifier.

`gu_workarea` Workarea GUID.

`gu_admins` GUID of Administrators Group for Workarea.

`gu_powusers` GUID of Power Users Group for Workarea.

`gu_users` GUID of Users Group for Workarea.

`gu_guest` GUID of Guests Group for Workarea.

`gu_other` GUID of Customizable Group for Workarea.

`path_logo` Path to physical files associated with Workarea.

`len_quota` Disk quota currently in use by Workarea.

`max_quota` Maximum allowed quota for Workarea.

Standard permissions masks

Access level of a User or Group over a Category is set by setting a permissions bit mask.

Predefined permissions are :

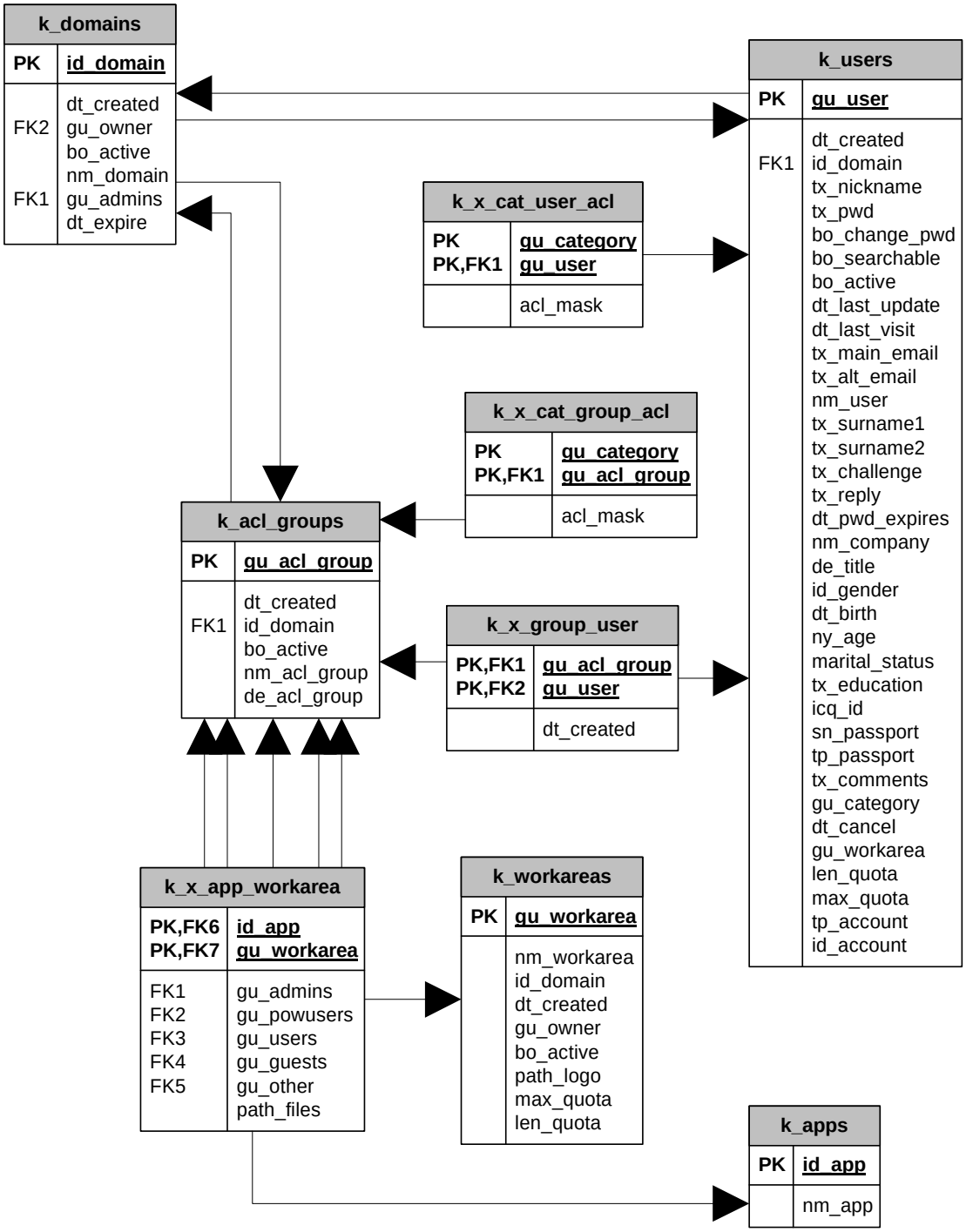
Table **k_lu_permissions**

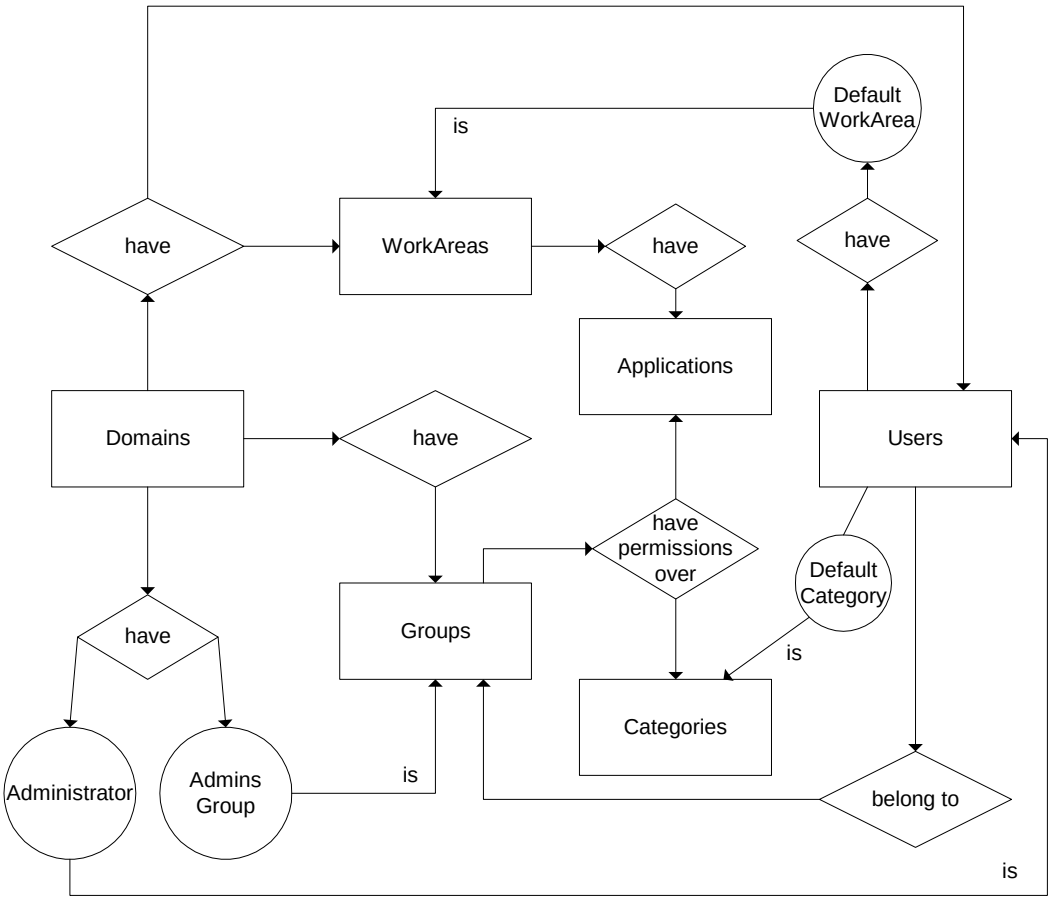
List	Read	Add	Delete	Modify	Moderate	Write	Manage Permissions	Full Control
1	2	4	8	16	32	64	128	2147483647

Initial data loading for Security submodel

The security submodel comes loaded with three predefined domains at `k_domains` table :

- **Domain 1024 (SYSTEM)**: This is a special domain for system wide administrative purposes only. Some applications recognize the SYSTEM domain and behave differently when logged into it. For the SYSTEM Domain there is only one administrator user and one administrator group and no other users and groups. Do not work on the SYSTEM domain or add any data to it.
- **Domain 1025 (MODEL)**: This domain is used as a template for creating new domains. It has 4 groups: administrators, powers users, users and guests and one user for each group. Do not work on the MODEL domain or add any data to it.
- **Domain 1026 (TEST1)**: This is a preloaded domain for development purposes (the one you can start developing and breaking just now).
- **Domain 1027 (ACEP1)**: Domain for validation testing.
- **Domain 1028 (PROD1)**: Empty Production Domain.





Login auditing

From version 2.2, all login attempts are saved on table **`k_login_audit`**.

`bo_success` ‘1’ if login was successful or ‘0’ if not

`nu_error` 0 if login was successful or the error code otherwise:

-1	User not found
-2	Invalid password
-3	Billing account deactivated
-4	Session expired
-5	Domain not found
-6	Workarea not found
-7	Workarea not set
-8	User account cancelled
-9	Password expired
-10	Captcha mismatch
-11	Captcha timeout
-255	Internal server error

`dt_login`	Timestamp
`gu_user`	User GUID
`tx_email`	User e-mail
`tx_pwd`	Password
`gu_workarea`	Workarea for logged user
`ip_addr`	IP address

Portlets submodel

There is a single table needed for holding hipergate portlets configuration which is located at `security.ddl` file.

`k_x_portlet_user`	This table contains page layout information for portlets and user.
`id_domain`	User domain identifier.
`gu_user`	User GUID.
`gu_workarea`	User Workarea. May be the same Workarea as `gu_workarea` from `k_users` for `gu_user` or another one.
`nm_portlet`	Name of the Java class implementing `javax.portlet.GenericPortlet` interface that will generate the portlet contents. For example, `com.knowgate.http.portlets.CalendarTab`.
`nm_page`	Name of the JSP page that contains the portlet.
`nm_zone`	Name of the page zone where the portlet will be located. Zone Names are arbitrary and may change from one page to another. Usual zones are `top`, `bottom`, `left` and `right`.
`op_position`	Ordinal top-down position of the portlet in its zone. Portlets with smaller positions are placed on top of those with greater ones when composing the page.

`id_state`	Either `NORMAL` or `MINIMIZED`.
`dt_modified`	Date and time when the data that the portlet must show was last updated. This timestamp does not refer to the portlet row at k_x_portlet_user but to the date when its shown data was modified. This information is used for either painting the portlet contents from a cached file or reloading them from the database. If the cached file is after dt_modified, then the file is served, else if the file was modified before last updated to the portlet's underlying data then the file is discarded and rebuilt.
`nm_template`	Name of the XSL style sheet that will be used for rendering the portlet output as XHTML.

Thesauri, Addresses and Images submodel

Thesauri

A thesauri is composed, basically, of the following elements :

- A list of word or concepts represented by combinations of words (terms).
- A knowledge scope to which terms apply.
- A list of synonyms for each term.
- A hierarchical relationship among terms that extends or specifies the meaning of them.
- Optionally, a description of the term meaning in the context that it is used.

For simplifying implementation, hipergate imposes three ad hoc restrictions to the thesauri:

1st) Not all terms may have synonyms. There are *primary terms* which may have synonyms and *synonym terms*, but no synonyms of synonyms.

2nd) Hierarchical relationship goes from most generic to most specific terms, there are no lateral relationships among similar terms.

3rd) Maximum ten levels of depth hierarchy.

Example:

Primary Term:	Vehicle
Synonym Term:	Transport
Level 1:	Car
Level 2:	Hired Car

Tables	**`k_thesauri_root`**	Top level terms.
	`gu_rootterm`	Term GUID.
	`tx_term`	Term text
	`id_scope`	Knowledge Scope.
	`id_domain`	hipergate security domain to which term belongs.
	`gu_workarea`	Workarea to which term belongs.
	`k_thesauri`	Terms
	`gu_rootterm`	GUID of root term for this term.
	`gu_term`	GUID of this term.
	`dt_created`	Register Creation Date.
	`id_language`	Language . (see `k_lu_languages`).
	`bo_mainterm`	1 if it is primary term, 0 if it is synonym.
	`tx_term`	Text for Term. Singular, all uppercase.
	`tx_term2`	Plural form of term.
	`id_scope`	Knowledge Scope.
	`id_domain`	Security Domain.
	`gu_synonym`	GUID of primary term for this synonym. Thus, terms which gu_synonym is null are primary terms.
	`de_term`	Term description. Briefly explains term meaning in the knowledge scope that it is used.
	`id_term[0..9]`	Keys for all parents of this term. A term has a GUID and a short numeric key. Each register holding a term contains the list of all numeric keys for parent terms.

There is a single addresses table for all addresses across the application suite. Each submodel links tables related with address with tables of the style *k_x_addr_myclass*.

Tables

k_addresses

gu_address	Address GUID.
ix_address	Address ordinal index. Can be used for ordering address.
gu_workarea	Workarea to which Address belongs.
dt_created	Register Creation Date.
bo_active	1 if address is active, 0 if it is inactive.
dt_modified	Register Modification Date.
gu_user	User (from k_users) to with address is associated.
tp_location	Address Type: “Central Office”, “Warehouse”, “Personal”, etc
nm_company	Company Name.
tp_street	Street Type
nm_street	Street Name.
nu_street	Street Number.
tx_addr1	Address Line 1.
tx_addr2	Address Line 2.
id_country	Country Code. See k_lu_countries.
nm_country	Country Name.

`id_state`	Estate Code.
`nm_state`	Estate Name.
`mn_city`	City Name.
`zipcode`	Postal Code.
`work_phone`	Work phone.
`direct_phone`	Direct phone.
`home_phone`	Personal phone.
`mov_phone`	Mobile phone.
`fax_phone`	Fax.
`other_phone`	Additional phone.
`po_box`	Post office box.
`tx_email`	E-mail.
`url_addr`	URL.
`coord_x`	X Coordinate.
`coord_y`	Y Coordinate.
`contact_person`	Contact Person.
`tx_salutation`	Salutation.
`id_ref`	External reference for interfacing with other applications.
`tx_remarks`	Comments.

k_addresses_lookup

gu_owner	Workarea GUID.
id_section	Name of column at k_addresses table.
pg_lookup	Progressive ordinal for lookup.
vl_lookup	Lookup actual value.
tr_es	Spanish Label.
tr_en	English Label.

k_distances_cache Cache of distances between two points.

lo_from	String identifying origin point.
lo_to	String identifying destination point.
nu_km	Number of kilometers between the two points.
id_locale	Locale in which origin and destination are expressed.
coord_x	Longitude.
coord_y	Latitude.

Images

The table k_images keeps references to images stored as disk files. These references are used for speeding up listing available images and for pre-computing disk quotas used by Users and Workareas.

k_images has special purpose columns for the two most common entries having images: PageSets and Products.

Tables

k_images

gu_image	Image GUID.
path_image	Image absolute path.

`dt_created`	Register Creation Date.
`gu_writer`	User owner of Image.
`gu_workarea`	Workarea.
`dt_modified`	Register Modification Date.
`nm_image`	File Name.
`tl_image`	Image Title (HTML ALT tag).
`tp_image`	Image Type: “Thumbnail”, “Detail”, etc.
`dm_width`	Width in pixels.
`dm_height`	Height in pixels.
`id_img_type`	Image extension: “GIF, “JPG”, etc..
`len_file`	File length in bytes.
`gu_pageset`	PageSet to which Image belongs.
`gu_block`	Block of PageSet to which Image belongs.
`gu_product`	Product to which Image belongs.
`url_addr`	Hiperlink.

Bank Accounts

Tablas	**`k_bank_accounts`**	Bank Accounts and Credit Cards.
	`nu_bank_acc`	Bank Account number (no spaces nor hyphens).
	`gu_workarea`	Workarea to which Bank Account belongs.
	`dt_created`	Register Creation Date.
	`bo_active`	1 is account is active, 0 if it is inactive.
	`tp_account`	Type of Bank Account.

`nm_bank`	Bank Name.
`nm_cardholder`	Name on Card.
`nu_card`	Credit Card number (no spaces or hyphens).
`tp_card`	Credit Card Type: “VISA”, “MASTERCARD”, etc.
`tx_expire`	Expire date in text formatted like MM/YYYY.
`nu_pin`	Card PIN.
`nu_cvv2`	Card CVV2.
`im_credit_limit`	Credit Limit.
`de_bank_acc`	Bank account description.

Job Scheduler Submodel

Job Scheduler submodule is designed for providing support to a wide variety of batch processes.

Each Job is composed of atomic units called Atoms.

Commands

Each Job has an associated Command. The Command identifies what the Job must do. When writing Java code, there is a generic abstract interface for Job and a subclass of Job implementing for each Command.

Tables

k_lu_job_commands List of recognized Commands.

`id_command` Command Identifier.

[version 1.0]
- MAIL → Send Atoms by e-mail.
- SAVE → Save files.
- FTP → Send by FTP.
- FAX → Send by FAX.

`tx_command`	Brief Command Description.
`nm_class`	Name of Java subclass of com.knowgate.scheduler.Job that contains the actual code implementing command behaviour.
k_lu_job_status	Job Status Lookup.
k_jobs	List of Jobs, all, pending and processed. This table is used as a processing queue. Processing order is not determined by data model but a concern of the Java code implementation.
`gu_job`	Job Unique Identifier.
`gu_workarea`	GUID of Workarea.
`gu_writer`	GUID of User that created the Job.
`id_command`	Command Identifier.
`id_status`	Job Status.
`dt_created`	Date Created.
`dt_execution`	Date scheduled for execution (NULL = as soon as possible).
`dt_finished`	Finished on Date.
`dt_modified`	Register Modification Date.
`tl_job`	Job Title.
`gu_job_group`	Job Group (unused in v1.x).
`tx_parameters`	Additional parameters for execution. This field contains a variable number of parameters with format "`name:value`" Each name:value pair is separated from the next with a comma.: "`id_pageset:123456,id_list:26879877`".
k_job_atoms	Atoms, for Jobs being executed..Atoms have preloaded all necessary information for its

processing without further database queries. This is a tactic for reducing database overload while processing atoms one by one.

k_job_atoms_archived Atoms for finished or cancelled Jobs. When a Job finishes, its atoms are moved from `k_job_atoms` to `k_job_atoms_archived` table.

Queries By Form (QBF)

Although Queries By Form are included in the Job Scheduler submodel, they are not part of the Job Scheduler itself.

Tablas

k_queries This table stores queries generated using a form wizard.
See Queries By Form at Java chapters reference for a more detailed explanation about QBFs.

`gu_query` Query GUID.

`dt_created` Creation Date.

`gu_workArea` Workarea GUID.

`tl_query` Query Name.

`nm_queryspec` Name (without extension) of an XML file that contains Query definition (metadata). These files are located on `/storage/qbf`.

`dt_modified` Date Modified.

`nm_field[1..3]` Name of Query field.

`tx_value[1..3]` Lookup label for query field.

`vl_code[1..3]` Value for query field.

`nm_operator[1..3]` Comparison operator for field value.
Recognized operators are:

=	Equal to
<>	Nor Equal to
>	Greater than
<	Less than
S	Starts with (substrings)
C	Contains (substrings)
N	Is Null
M	Is Not Null

☞ When composing queries, take into account that condition field=NULL is different from field IS NULL. Relational databases assume that no two NULL values are the same.

`tx_condition[1..2]` Logical operator between comparison operators.
{ AND, OR }.

`tx_columns` Columns to be retrieved by query.

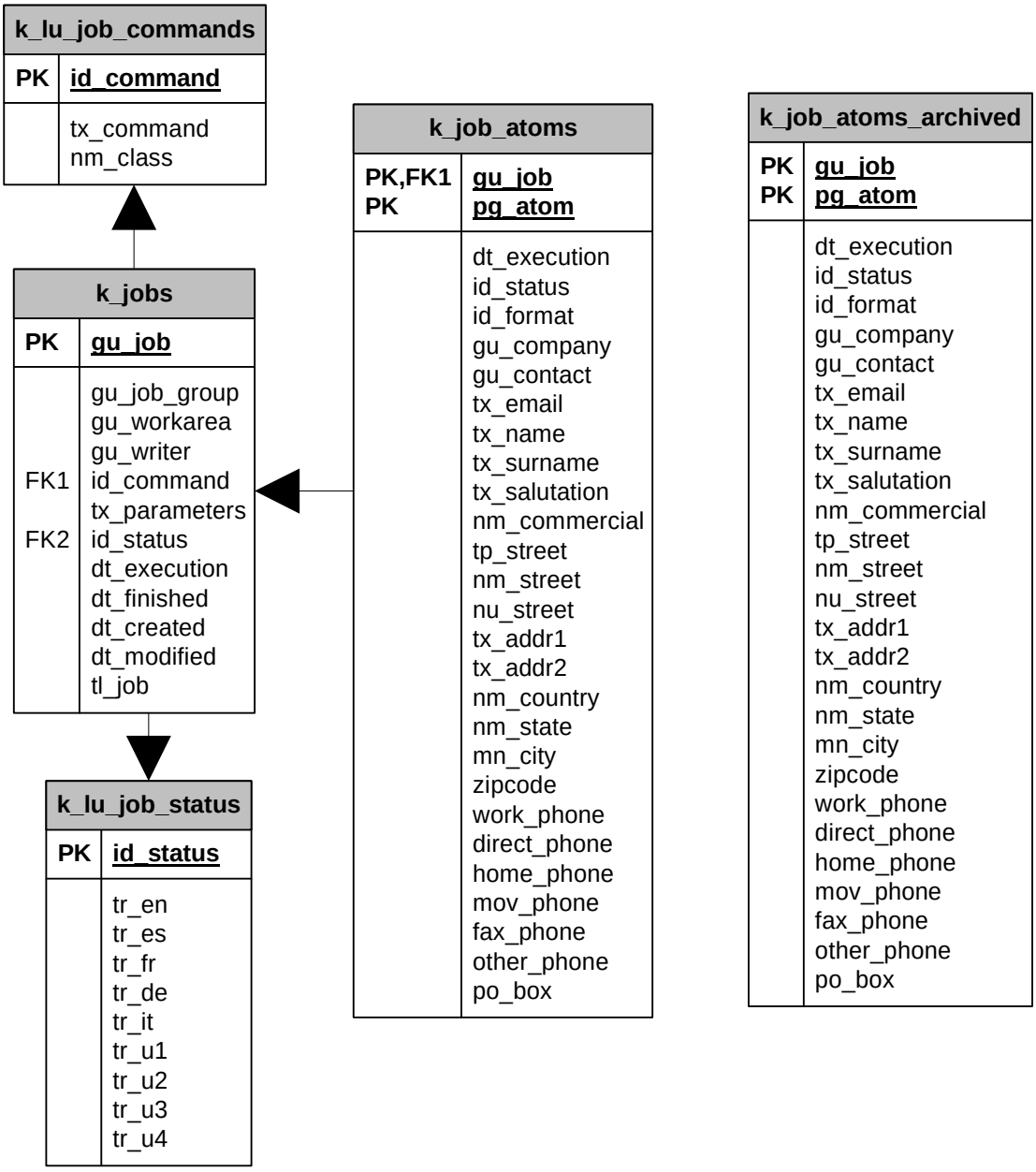
Example:

Query Name : High Priority Pending Duties.
QBF : duties [.xml]
SQL : SELECT * FROM v_duty_resource WHERE od_priority=3 AND tx_status='PENDIENTE'

Would be stored at `k_queries` table as :

tl_query	High Priority Pending Duties
nm_queryspec	Duties
nm_field1	od_priority
nm_field2	tx_status
nm_field3	NULL
nm_operator1	=
nm_operator2	=
tx_value1	HIGH (field k_duties_lookup.tr_en)
tx_value2	PENDING
vl_code1	3 (field k_duties_lookup.vl_lookup)
vl_code2	PENDIENTE
tx_columns	NULL

☞ The name of base view or table and additional filters per Workarea are not stored at k_queries table as they are common to all queries of the same type and thus are stored at query XML definition file.



Workgroup Submodel

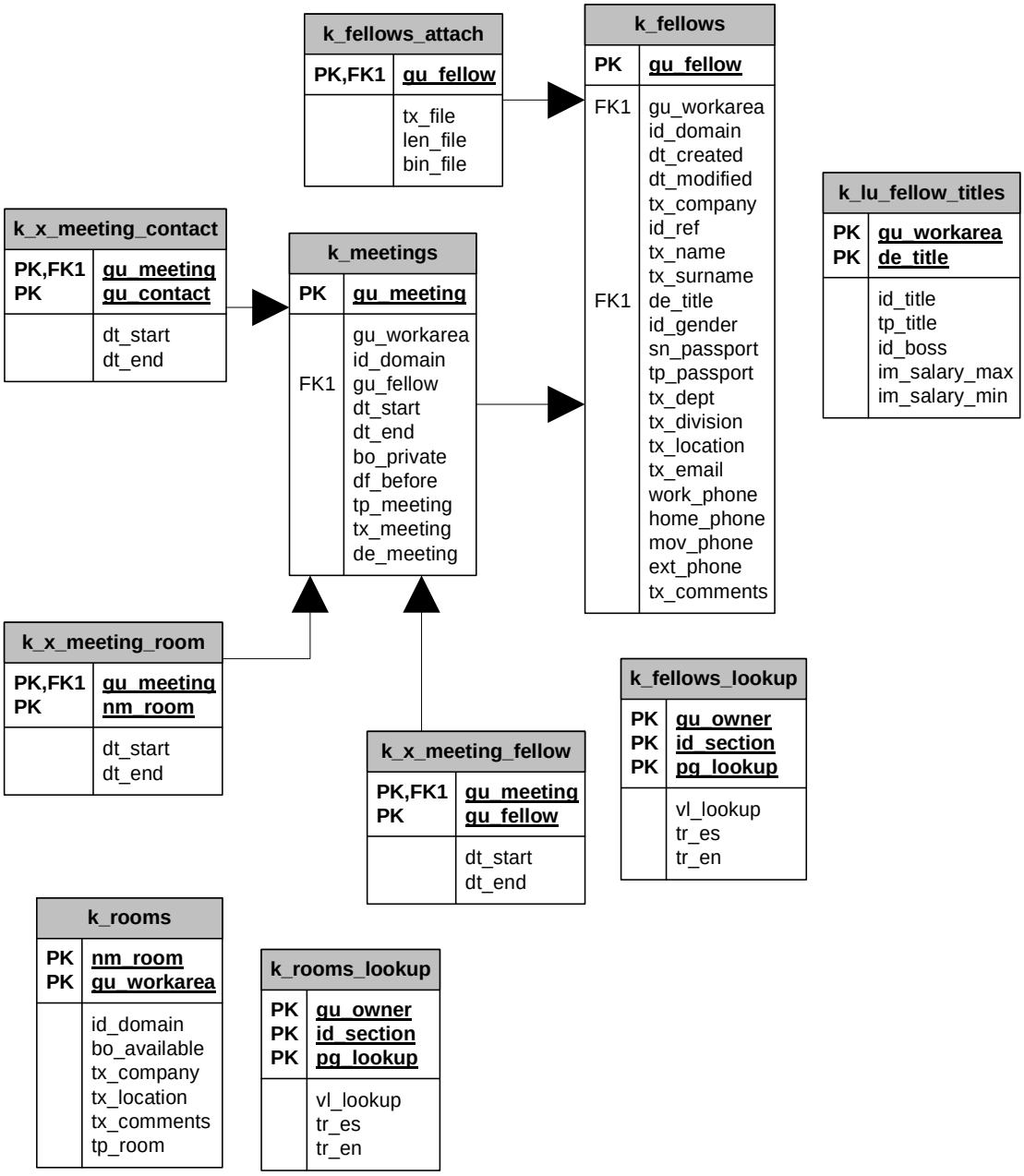
This submodel contains employee directory, shared resources booking and agenda.

Users and employees are independent entities, but if a logged user enters Collaborative Tools, a new employee for him would be automatically generated without request.

Tables	**k_lu_fellow_titles**	Contains a positions tree per Workarea. Only one organizational tree can be created per Workarea.
	`gu_workarea`	Workarea GUID.
	`de_title`	Position Description.
	`id_title`	Internal position code (optional).
	`tp_title`	Professional Category(optional).
	`id_boss`	Hierarchical dependency from other position.
	`im_salary_max`	Maximum Salary (HR).
	`im_salary_min`	Minimum Salary (HR).
	k_lu_fellows	Employee List.
	k_fellows_attach	Employee Photos.
	k_fellows_lookup	Employee lookup values.
	k_rooms	Rooms and resources to book.
	k_rooms_lookup	Rooms lookup.
	k_meetings	Scheduled Activities and Meetings.
	`gu_meeting`	Activity unique identifier.
	`gu_workarea`	Workarea GUID.

`id_domain`	Domain to which Activity belongs.
`gu_fellow`	Employee who created this Meeting.
`dt_start`	Start Date-time.
`dt_end`	End Date-time.
`bo_private`	1 is meeting is private, 0 if it is public, private meeting details are only visible by employee who created it.
`df_before`	Notify n-minutes before meeting start date-time.
`tp_meeting`	Meeting Type.
`tx_meeting`	Meeting Subject.
`de_meeting`	Meeting Description.

k_x_meeting_room	Rooms and Resources booked per Meeting.
k_x_meeting_fellow	Employees attending to a meeting.
k_x_meeting_contact	External Contacts attending to a meeting.



Work calendars

From version 4.0 onward, hipergate includes the handling work calendars. Each calendar defines a set of working / non working days between two dates.

Calendars are hierarchical and can be merged in order to create a new calendar which is a combination of two previous ones. It is possible to define from global calendars that apply to everybody to individual calendars that apply only to a single user. The final calendar which

applies to a user can be a combination of its own holidays, the working time of his enterprise, the working time of his state and the working time of his country.

Each day in the calendar is marked as either working or non working. Also, for working days it is possible to define working hours divided into morning and evening intervals.

Work calendars are stored at tables `k_working_calendar` and `k_working_time`.

Tables

k_working_calendar	Calendar master table
`gu_calendar`	GUID of calendar.
`gu_workarea`	GUID of Workarea to which the calendar belongs.
`id_domain`	Numeric identifier of domain to which the calendar belongs.
`nm_calendar`	Calendar descriptive name.
`dt_created`	Creation date.
`dt_modified`	Last modified date.
`dt_from`	First day defined at calendar.
`dt_to`	Last day defined at calendar.
`gu_user`	GUID of user to which the calendar applies.
`gu_acl_group`	GUID of group to which the calendar applies.
`gu_geozone`	GUID of geographic zone to which the calendar applies.
`id_country`	Identifier of country to which the calendar applies.
`id_state`	Identifier of state to which the calendar applies.
k_working_time	Working days and time for each day of calendar.

`dt_day`	Day. Integer with format yyyymmdd.
`gu_calendar`	GUID of calendar.
`bo_working_time`	Short Integer. 1 for workdays or 0 for holidays.
`hh_start1`	Short Integer. Start hour of morning working time [0..23] or -1 if no working time is defined for the morning.
`mi_start1`	Short Integer. Start minute of morning working time [0..59] or -1 if no working time is defined for the morning.
`hh_end1`	Short Integer. End hour of morning working time [0..23] or -1 if no working time is defined for the morning.
`mi_end1`	Short Integer. End minute of morning working time [0..59] or -1 if no working time is defined for the morning.
`hh_start2`	Short Integer. Start hour of evening working time [0..23] or -1 if no working time is defined for the evening.
`mi_start2`	Short Integer. Start minute of evening working time [0..59] or -1 if no working time is defined for the evening.
`hh_end2`	Short Integer. End hour of evening working time [0..23] or -1 if no working time is defined for the evening.
`mi_end2`	Short Integer. End minute of evening working time [0..59] or -1 if no working time is defined for the evening.
`de_day`	Day description (mainly for special holidays).

Products and Documents Submodel

Many objects that may be categorized (with the exception of Companies, Contacts and News Messages) are handled in a unified way using `k_products` table. This table contains products from the virtual shop and documents and links from the corporate library.

Concepts and Definitions

Product

Is a general abstract representation of an object contained at a Category. Products can be physical items, as in the virtual shop, or electronic documents or links as in the corporate library.

Location

There may be many copies of the same Product at different locations. Locations can mean :

- warehouses (with their corresponding stock).
- different versions from the same document.
- mirror sites from which to download a file.
- Pieces of a compound document.

Predefined Attribute

Products have a predefined collection of most common attributes hardwired into the data model.

Custom Attribute

For product attributes that do not match any predefined one, it is possible to define new custom ones.

Keywords

Each product can have a keyword set.

Tables **k_products**

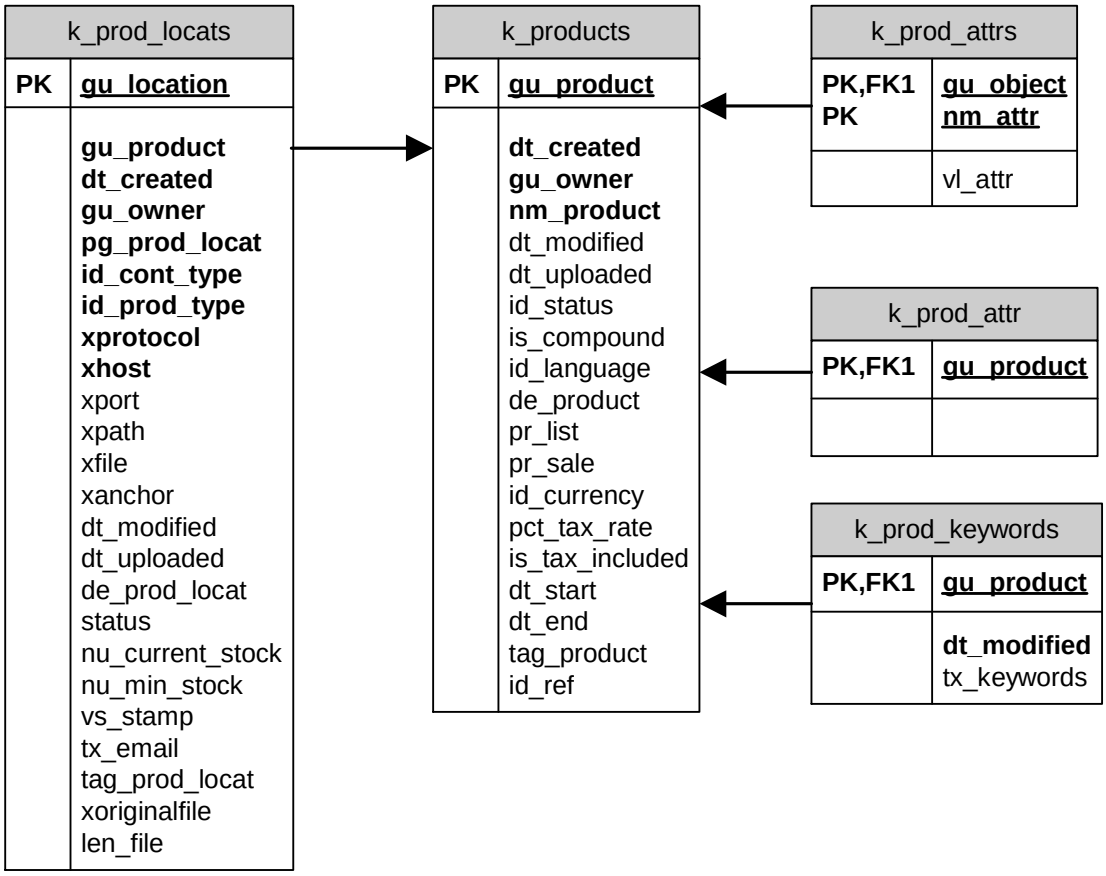
gu_product	Product GUID.
dt_created	Register Creation date.
gu_owner	GUID of User (k_users) owner of the Product.
nm_product	Product Name.
id_status	Status. See k_lu_status.
is_compound	0 if product is simple, or 1 if it is compound.
gu_blockedby	GUID of User that currently has the product blocked from his exclusive use.
dt_modified	Register Modification Date.
dt_uploaded	Date when Product was uploaded.
id_language	Language Code.
de_product	Product Description.
pr_list	List Price.
pr_sale	Sale price (offers and bargains).
id_currency	Numeric Code for price currency. See k_lu_currencies.
pct_tax_rate	Tax percentage. This must be a percentage. If taxes are 15% then pct_tax_rate must have value 15 and not 0.15.
is_tax_included	1 if taxes are already included in price, 0 if not.

`dt_start`	Offer/Bargain Start Date.
`dt_end`	Offer/Bargain End Date.
`tag_product`	Product additional text.
`id_ref`	External reference for interfacing with other applications.
k_prod_locats	There can be many copies of a Product at different locations.
`gu_location`	Product Location GUID.
`gu_product`	Product GUID.
`pg_prod_locat`	Location progressive value. Product GUID along with Location progressive are and alternative primary key for table `k_prod_locats`. The client application must correctly assign proper location progressives.
`dt_created`	Register Creation Date.
`gu_owner`	User owner of location.
`id_cont_type`	Container Type. Must be consistent with access protocol (`xprotocol`). See `k_lu_cont_types`.
`id_prod_type`	Product type. See `k_lu_prod_types`.
`len_file`	Length in bytes of associated file. For links and physical items must be zero.
`xprotocol`	Access Protocol.: file:// Local fields. ftp:// FTP files. http:// HTML links. https:// HTML links (secure). jdbc:// Database URL. ware:// Physical Items.

`xhost`	Host Name. Without delimiters. For example : "`files.hipergate.org`".
`xport`	Port (optional).
`xpath`	Path.Prefixed with a file separator and without file separator at the end. Examples: YES /opt/knowgate/knowgate NO ~~opt/ knowgate/knowgate~~ NO ~~/opt/ knowgate/knowgate/~~
`xfile`	Internal file name.
`xoriginalfile`	Original file name (in case it were internally renamed).
`xanchor`	Document anchor (for URLs) without sharp
`dt_modified`	Register Modification Date.
`dt_uploaded`	Date uploaded.
`de_prod_locat`	Location Description.
`status`	Status.
`nu_current_stock`	Current Stock.
`nu_reserved_stock`	Reserved Stock.
`nu_min_stock`	Minimum stock.
`vs_stamp`	Version Label.
`tx_email`	E-mail.
`tag_prod_locat`	Additional text for location.

k_prod_attr	This is a sub-register of `k_prod_attr` table. Contains predefined product attributes.
k_prod_attrs	Custom product attributes.

	`gu_object`	Product GUID.
	`nm_attr`	Attribute name.
	`vl_attr`	Attribute Value.
	k_prod_keywords	Keywords for Product.
	`gu_product`	Product GUID.
	`dt_modified`	Register Modification Date.
	`tx_keywords`	Free text up to 4000 characters.
Views	**v_prods_with_attrs**	Product Locations with product attributes.



Version Control

Version control is managed as a special case of keeping different copies of the same base document at different locations. The data model also allows exclusive blocking of files in use.

The process is as follows :

1st) Create Initial version.

1.1 Generate GUID for Product/Document.
1.2 Assign Product Name from client application.
1.3 Assign field `k_products.gu_owner` to GUID of User owner of the Product.
1.4 Set Product Status to 1 (Active).
1.5 Set field `is_compound` to 0 (compound documents are not versionable).
1.6 Set last modification date to NULL (not modified).
1.7 Set creation date from database server.
1.8 Set upload date from client application.
1.9 Generate GUID for first version (location).
1.10 Assign 1 to progressive of first version (location).
1.11 Assign field `k_prod_locats.gu_owner` to GUID of User owner of the Product.
1.12 Set version (location) status to 1 (Active).
1.13 Set same upload date for Location than was set for Product.
1.14 Set version label at field `k_prod_locats.vs_stamp` (optional).

2nd) Document Check-out.

2.1 Latest version is that with most recent upload date (`dt_uploaded`). For finding it first `dt_uploaded` field from table `k_products` is retrieved and then a register with same `dt_uploaded` date is searched at `k_prod_locats` Table.
2.2 Change status for Document and all its versions to 2 (Blocked).
2.3 Set field `k_products.gu_blockedby` to GUID of User checking-out the Document.

3rd) Document Check-in.

3.1 Create new version (location).

3.2 Update field `k_products.dt_uploaded` with upload date of latest version
3.3 Update fields `dt_modified` at `k_products` and `k_prod_locats` to current system date.
3.4 Change status for Document and all its Versions to 1 (Active).
3.5 Set field `k_products.gu_blockedby` to NULL.

4th) Undo check-out.

4.1 Set status for Document and Versions to 1.
4.2 Set field `k_products.gu_blockedby` to NULL.

5th) Remove a Document Permanently.

It is completely deleting document disk files and database references. May only be done by users with administrator role.

Managing Product at Several Warehouses

Locations acts as foundation for the virtual shop submodel. Each location keeps information about stock at a given warehouse.

Virtual Shop Submodel

The Virtual Shop Submodule is composed of the following elements :

- Catalogs
- Product Categories
- Products
- Orders
- Dispatch Advices
- Invoices

Catalogs
Each Workarea may contain one or more catalogs.
Catalogs are useful for keeping apart sets of products that belong to different shops, or just to establish an arbitrary division of products among families of them.

On any given catalog there is a single set of XSL templates for Orders, Dispatch Advices and Invoices.

Catalogs are stored at table `k_shops` and manager with Java class `com.knowgate.hipergate.Shop`.

Categories
Each catalog has one Root Category of which all the others are connected. Categories are organized into a hierarchy and there can be an unlimited number of sublevels.

At the data model, a product may belong to several categories, but the Standard user interface allows just one to be entered.

Categories are stored at table `k_categories shops` and manager with Java classes : `com.knowgate.hipergate.Categories`, `com.knowgate.hipergate.Category` and `com.knowgate.hipergate.CategoryLabel`.

Products
Each product is composed of :

- Basic Information
- Images
- Fares
- Stock locations
- Fixed attributes
- User defined attributes
- Attached files
- Keywords

Product Basic information is stored at table `k_products`. The main class for managing products is `com.knowgate.hipergate.Product`.

Each Product may have an unlimited number of images associated at `k_images` table. Each image is labeled with a given type at column `tp_image` of table `k_images`. The redefined image types are : `thumbview`, `normalview`, `frontview` and `rearview`. Other images types may be used, although changes are necessary to the Standard user interface to accommodate.
The physical image files are stored under the directory specified by property Workareasput of hipergate.cnf file, using the Workarea GUID and the Shop name on the following way :

Workareasput/apps/Shop/*k_shops.nm_shop/gu_product_tp_image*

Optionally, there may be several fares associated with a Product. The Basic Product information already has a list price and a discount price. So there is only necessary to use fares when product price actually varies from one customer to another.
Fares are stored at table `k_prod_fares` and handled with Java class: `com.knowgate.hipergate.ProductFare`.

The stock of a product may be located at one or several warehouses. There is always at least one Product Location at table `k_prod_locats` for each Product. The class used for managing Product Locations is `com.knowgate.hipergate.ProductLocation`.

For each Product there is a set of fixed attributes and another set of variable attributes. Fixed attributes are stored at table `k_prod_attr` and variable attributes are stored at en `k_prod_attrs`.

Files attached to a Product are handled as if they where Product Locations.

Orders
Orders are stored at table `k_orders`. Each Order has a set of Order Lines at table `k_order_lines`.

Orders have a progressive numbering at `pg_order` column assigned by `seq_k_orders` sequence.

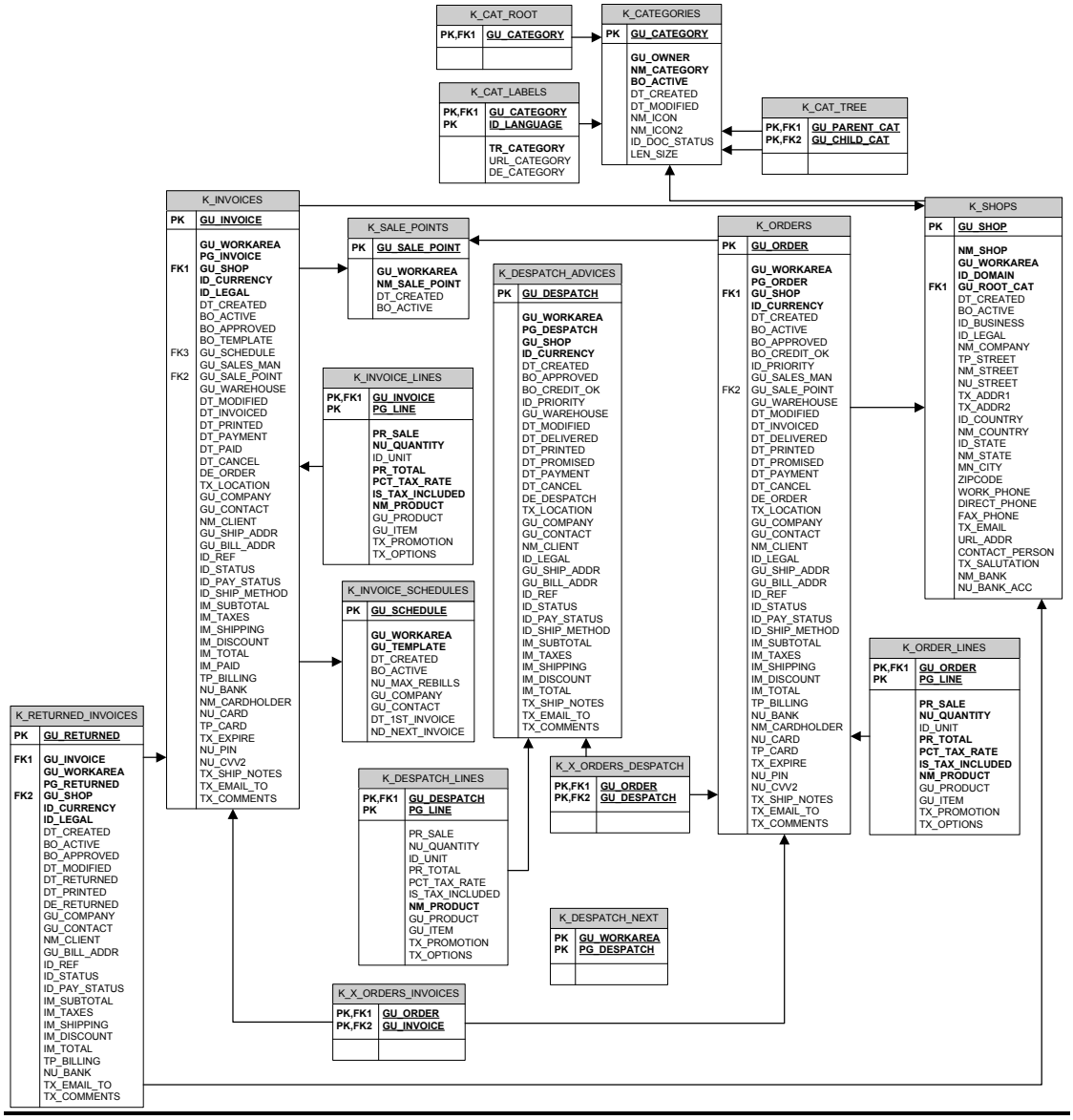
Dispatch Advices
Dispatch Advices are stored at `k_despatch_advices`. Each Dispatch advice has a set of Despatch Lines at `k_despatch_lines`. The association between Orders and Despatch Advices is maintained at table `k_x_orders_despatch`.

Despatch Advices are numbered separately for each Workarea at column `pg_Dispatch` by using table `k_Dispatch_next`.

Invoices
Invoices are stored at `k_invoices`. Each invoice has a set of lines at `k_invoice_lines`. The association between Orders and Invoices is maintained at `k_x_orders_invoices`.

Invoices are numbered separately for each Workarea at column `pg_invoice` by using table `k_invoices_next`.

Returned invoices are stored at able `k_returned_invoices`.



Sales Force Automation Submodel

Concepts and Definitions

Salesman	A salesman is a particular type of domain user. He has sales objectives associated.
Company	A Company may be a client, supplier, partner, or other type of organization.
Contacts	Contacts are individual people inside Companies. Contacts are not Domain Users nor Employees for Workgroup submodel.
Notes	Notes can be associated to Contacts for tracking phone calls, documents sent, etc.
Attached Files	Files attached to a Contact.
Opportunities	Sales Opportunities. Each opportunity is associated with a Contact.
Bank Accounts	Bank Accounts are defined at the thesauri submodel. Accounts can be associated to both Companies and Contacts.

Submodel elements

Tables	**k_sales_men**	Salesmen.
	k_sales_objectives	Sales Objectives.
	k_companies	Companies.
	`gu_company`	Company GUID.
	`dt_created`	Register Creation Date.
	`nm_legal`	Legal Name.
	`gu_workarea`	Workarea GUID.
	`nm_commercial`	Commercial Name.
	`dt_modified`	Register Modification Date.
	`dt_founded`	Company Foundation Date.

`id_legal`	Legal Id.
`id_sector`	Sector Code.
`id_status`	Status.
`id_ref`	Reference for interfacing with other applications.
`tp_company`	Company Type.
`gu_geozone`	Geographic Zone to where Company is.
`nu_employees`	Employee head count.
`im_revenue`	Annual Revenue.
`de_company`	Company Description.
k_x_company_bank	Bank Accounts per Company.
`gu_company`	Company GUID.
`nu_bank_acc`	Bank Account Number.
k_x_company_addr	Addresses per Company.
`gu_company`	Company GUID.
`gu_address`	Address GUID.
k_ companies_lookup	Company lookups table.
k_companies_attrs	User defined attributes for companies.
k_contacts	Contacts.
`gu_contact`	Contact GUID.
`gu_workarea`	Workarea GUID.
`dt_created`	Register Creation Date.

`bo_private`	1 if contact is private for the User that entered him, 0 if Contact is public inside the Workarea to which it belongs.
`nu_notes`	Count of Notes associated with Contact.
`nu_attachs`	Count of files attached to a Contact.
`dt_modified`	Register Modification Date.
`gu_writer`	User that entered the Contact.
`gu_company`	GUID of Company to which the Contact belongs.
`id_status`	Status (lookup).
`id_ref`	Reference for interfacing with other applications.
`tx_name`	Name.
`tx_surname`	Surname.
`de_title`	Position (lookup).
`id_gender`	Gender ('M'=Masculine, 'F'=Feminine)
`dt_birth`	Birth Date
`ny_age`	Age.
`sn_passport`	Number of identity document.
`tp_passport`	Type of identity document (lookup).
`tx_dept`	Department.
`tx_division`	Division.
`gu_geozone`	Geographic zone where Contact is.
`tx_comments`	Comments.

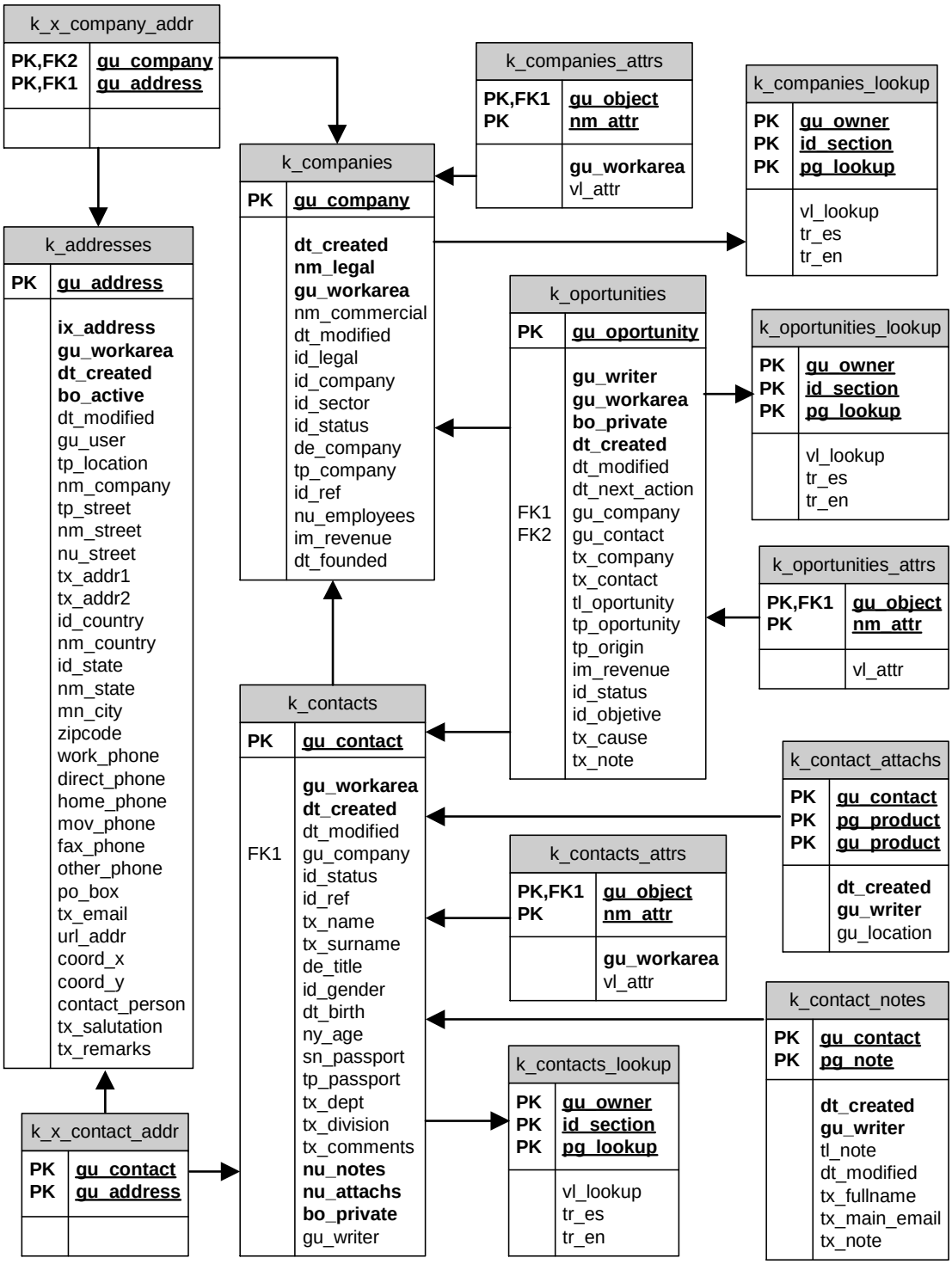
k_ contact_notes Notes for a Contact.

gu_contact	Contact GUID.
pg_note	Note progressive ordinal.
dt_created	Register Creation Date.
gu_writer	GUID of User who wrote the Note.
tl_note	Note Subject
dt_modified	Note last modified on date.
tx_fullname	Full Name of user who wrote the note.
tx_main_email	E-mail of user who wrote the note.
tx_note	Note Text (up to 4000 characters).

k_ contact_attachs	Files attached to a Contact. Attached files are handles as references to table k_products.
gu_contact	Contact GUID.
pg_product	Attachment progressive ordinal.
gu_product	GUID of Product that has the pointer to file disk at its location.
dt_created	Register Creation Date.
gu_writer	GUID of User who attached the Note.
k_x_contact_bank	Bank Accounts per Contact.
gu_contact	Contact GUID.
nu_bank_acc	N Bank Account Number.
k_ x_contact_addr	Addresses per Contacto.
k_x_contacts_lookup	Contacts lookup table.

k_x_contacts_attrs	User defined attributes for contacts.
k_oportunities	Opportunities.
`gu_oportunity`	Opportunity GUID.
`gu_writer`	GUID of User who created the opportunity.
`gu_workarea`	Workarea GUID.
`bo_private`	1 if opportunity is private for user who created it, 0 if opportunity is public inside the Workarea to which it belongs.
`dt_created`	Register Creation Date.
`dt_modified`	Register Modification Date.
`dt_next_action`	Next Action Date.
`gu_company`	Company GUID.
`gu_contact`	Contact GUID.
`tx_company`	Company Legal Name.
`tx_contact`	Contact Full Name (Name+Surname).
`tl_oportunity`	Opportunity Title.
`tp_oportunity`	Opportunity Type.
`tp_origen`	Opportunity Origin.
`im_revenue`	Foreseen or achieved revenue.
`id_status`	Opportunity Status.
`id_objetive`	Opportunity Objective.
`tx_cause`	Close Cause.
`tx_notas`	Comments.

	k_ oportunities_lookup	Opportunity lookups.
	k_ oportunities _attrs	User defined attributes for oportunities.
	k_oportunities_changelog	Keeps track of changes performed to opportunity status and other columns.
	`gu_oportunity`	Opportunity GUID.
	`nm_column`	Name of column modified at k_oportunities.
	`gu_writer`	GUID of User who modified the opportunity.
	`dt_modified`	Register Modification Date.
	`id_former_status`	Status before modification.
	`id_new_status`	Status after modification.
	`tx_value`	If nm_column='id_status' then this column is the value of tx_cause field from k_oportunities.
Views	**v_company_address**	Active Addresses per Company. (`k_address.bo_active<>0`).
	v_contact_titles	Contact Positions (`k_contacts_lookup .id_section='de_title'`).
	v_contact_company	Contacts per Company.
	v_active_contact_address	Active Addresses per Contact.
	v_contact_address	Active Addresses per Contact
	v_contact_list	Contacts including Company information.



Most of the required information for content production is stored within XML files outside the database.
The DBMS is only used for maintaining indexes that accelerate searching information.

Tables

k_microsites

Microsites available to all Workareas or per Workarea.

All surveys use the same Microsite which comes preloaded from 2.2 and above with GUID: `"SURVEYMICROSITEJIXBXMLDEFINITION"`

`gu_microsite` Microsite GUID.

`dt_created` Register Creation Date.

`tp_microsite` Microsite Type:

1	Microsites based on XSL templates
2	Microsites based on free HTML
4	Surveys

`nm_microsite` Microsite Name.

`path_metadata` Relative path of Microsite XML definition file from directory `/storage/xslt/templates`
For surveys this field is always `xslt/schemas/survey-def-jixb.xml`

`id_app` Numeric identifier of application that handles the Microsite :

- `13` *Mailwire*. For newsletters and other single page documents suitable for being sent by e-mail.
- `14` *Web Builder*. Multi-page websites.
- 23 *Surveys*.

`gu_workarea` GUID of Workarea to which Microsite belongs or NULL if Microsite is available for all Workareas.

k_pagesets Microsite page instances.

`gu_pageset`	PageSet GUID.
`dt_created`	Register Creation Date.
`gu_microsite`	Microsite GUID.
`gu_workarea`	Workarea GUID.
`nm_pageset`	PageSet Name.
`vs_stamp`	PageSet Version Label.
`id_language`	Language. See `k_lu_languages`.
`path_data`	Relative to PageSet XML files from directory `/storage/domains` For surveys this field does not contain the path of a file but of a directory like `xslt/templates/Survey`. The path to each definition page of the survey for JiXB binding is composed by concatenating `nm_pageset` with number of requested page (`k_pageset_pages.pg_page`). Thus if `nm_pageset='OpenSurvey'` then the XML definition file for page two of the survey will be at: `xslt/templates/Survey/OpenSurvey2.xml` This is because Newsletter and Website templates contain the layout for all pages in a single XML file, but surveys have their definition split among several files, one per page.
`id_status`	PageSet Status (lookup).
`dt_modified`	Register Modification Date.
`tx_comments`	Comments (up to 255 characters).

k_pageset_pages

`gu_page`	GUID of PageSet Page.

`pg_page`	Page Progressive Ordinal.
`dt_created`	Page Creation Date.
`dt_modified`	Page Modification Date.
`tl_page`	Page Title.
`path_page`	Page Path.
k_pagesets_lookup	PageSet lookups.
k_pageset_answers	
`gu_datasheet`	GUID of responses set.
`gu_page`	GUID of page.
`pg_answer`	Response sequence [1..n].
`dt_created`	Date when response was created.
`dt_modified`	Date when response was last modified.
`gu_pageset`	GUID of survey to which this responses belong.
`tp_answer`	Response type {`TEXT, MEMO, CHOICE, MULTICHOICE, LICKERT, BOOLEAN, LIST, MATRIX`}.
`nm_answer`	Unique name for this response inside the response set.
`gu_writer`	GUID of user who wrote the response (from `k_users.gu_user`)
`tx_answer`	Response content. For responses with multiple choice options, each option comes separated from the other typically by a semi-colon.

Forums Submodel

Forums submodel allows an unlimited number of NewsGroups organized hierarchically.

At low level, a NewsGroups is a subregister of a Category, so NewsMessages are actually stored inside Categories.

k_categories

NewsGroup is a subclass of Category. For each NewsGroup GUID (`gu_newsgrp`) there is a corresponding Category with the same GUID (`gu_category`).

k_newsgroups

`gu_newsgrp`	Newsgroup GUID. It is the same GUID as the Category which holds the NewsGroup.
`id_domain`	Domain Numeric Identifier.
`gu_workarea`	Workarea GUID.
`dt_created`	NewsGroup Creation Date.
`bo_binaries`	1 if NewsGroup admits binary attachments on messages, 0 if not.
`dt_expire`	Days that a message will be keep alive before expiring. Days start counting from message publishing date.
`de_newsgrp`	NewsGroup Description.
`tx_journal`	(Optional) XML data describing how a static copy of the Newsgroup must be built. See Creating a blog or a static posts page set section for more information.

k_newsmsgs

`gu_msg`	News Message GUID.
`nm_author`	Sender Full Name.

`gu_writer`	Sender GUID (from `k_users` table).
`dt_published`	Date when submitted for approval.
`dt_start`	Date when it message must become visible.
`id_language`	Language.
`id_status`	Status.
`id_msg_type`	Message Type. `TXT` Plain Text `HTML` HTML
`nu_thread_msgs`	Number of messages on this thread.
`gu_thread_msg`	GUID of first message on thread.
`gu_parent_msg`	GUID of previous message on thread.
`tx_email`	Sender e-mail address.
`tx_subject`	Subject.
`dt_expire`	Expire Date.
`dt_validated`	Date when message was validated by moderator.
`gu_validator`	GUID of user that validated the message.
`gu_product`	GUID of Product that holds message attachments.
`tx_msg`	Message Text (up to 16Mb).

k_x_cat_objs

	News Messages are associated to News Groups as objects inside categories.
`gu_category`	News Group GUID (`gu_newsgrp`).
`gu_object`	News Message GUID (`gu_msg`).

`id_class`	Class NewsMessage numeric identifier. This field is always 31 for NewsMessages.
k_products	
	GUID If the message has attachments there is a single register at `k_products` for that message.
k_prod_locats	
	There is an entry at `k_prod_locats` for each attached file. Physical files are stored outside the database under `/web/workareas/`*`gu_workarea`*`/apps/Forums/`*`nm_category`*

☞ Do not delete NewsMessages that have attachments directly from the database. Use the Java API or else the attached files will be left unreferenced forever at disk.

Projects and Incidents Submodel

This submodel allows the creation of Projects and Support Contracts associated with Companies or Individuals.

For each project it is possible to define Duties and Incidences.

Tables	**k_projects**	Projects.
	`gu_project`	Project GUID.
	`dt_created`	Project Creation Date.
	`nm_project`	Project Name.
	`gu_owner`	Workarea to which project belongs.
	`id_parent`	Parent Project.
	`id_dept`	Department assigned to project.
	`id_status`	Project Status.

`dt_start`	Project Start Date.
`dt_end`	Project End Date.
`pr_cost`	Cost.
`gu_company`	Client Company.
`gu_contact`	Client Individual Contact.
`id_ref`	Client External Reference.
`de_project`	Description (up to 1000 characters).

k_projects_lookup	Project Lookup values.
k_project_expand	Keeps a pre-expanded list of all project child's and grandchild's at all depth levels.
`gu_rootprj`	Root project GUID for this branch.
`gu_project`	GUID of child project.
`gu_parent`	Project immediate parent GUID.
`nm_project`	Child project name.
`od_level`	Depth Level (1 = Root project).
`od_walk`	Order for walkthrough at each depth level.
k_project_costs	Costs associated to a Project.
`gu_project`	Project GUID.
`dt_created`	Cost Creation Date.
`dt_modified`	Cost Last Modification Date.
`dt_cost`	Date when Cost must be applied.
`gu_user`	User responsible for the Cost.
`gu_writer`	User that created of modified the Cost.

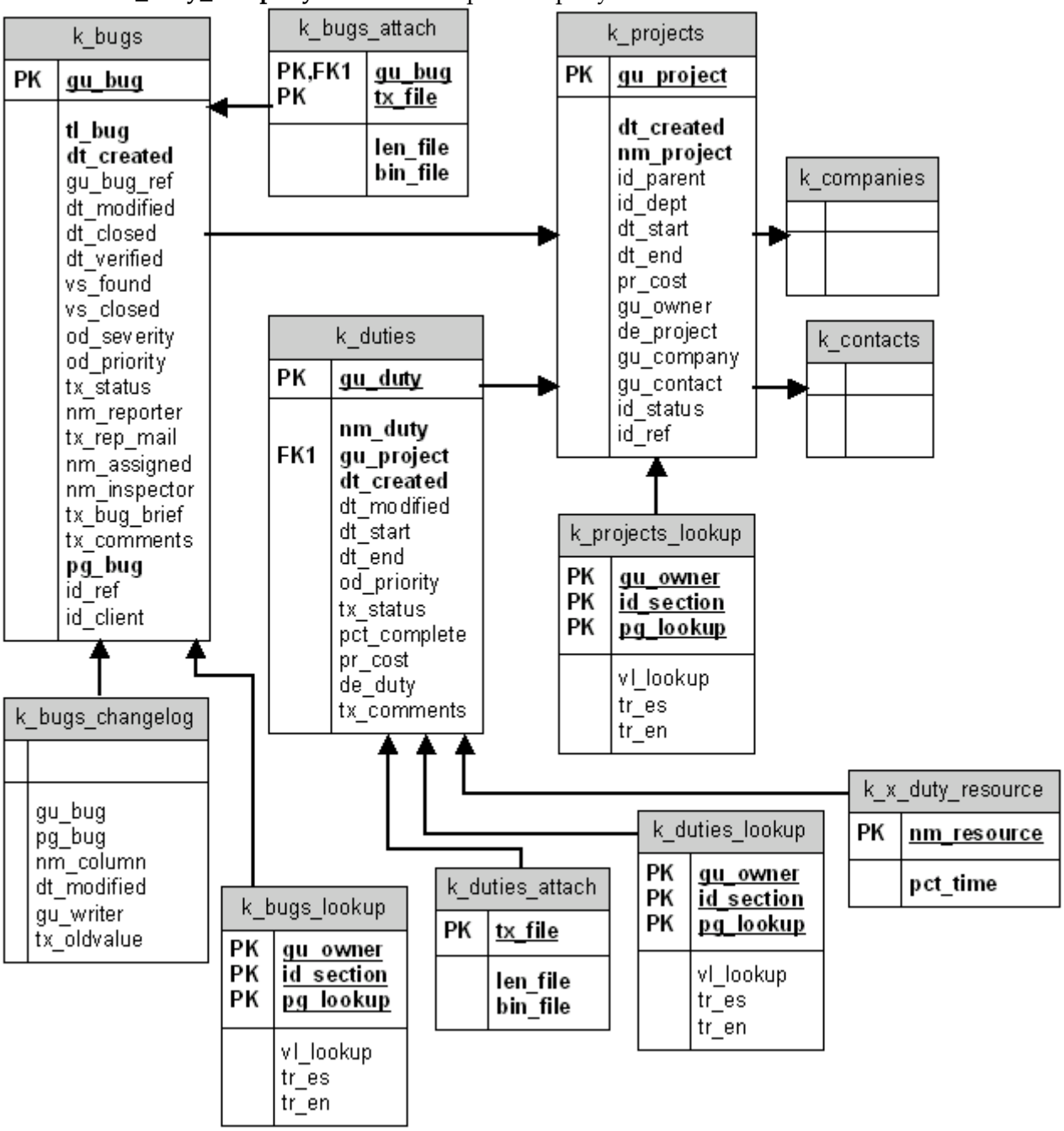
`tp_cost`	Cost Type.
`pr_cost`	Cost (monetary or time units).
`tl_cost`	Cost short title.
`de_cost`	Cost detailed description.
k_project_snapshots	Project Snapshots.
`gu_snapshot`	`Snapshot` GUID.
`gu_project`	GUID of Project over which snapshot was taken
`gu_writer`	GUID of User who took the Snapshot.
`dt_created`	Snapshot creation date.
`tl_snapshot`	Snapshot title.
`tx_snapshot`	Snapshot XML contents.
k_duties	Duties per Project.
`gu_duty`	Duty GUID.
`nm_duty`	Duty name.
`gu_project`	GUID of Project to which Duty belongs.
`dt_created`	Duty Creation Date.
`dt_modified`	Duty Last Modification Date.
`dt_start`	Duty Start Date.
`dt_end`	Duty End Date.
`od_priority`	Priority.
`tx_status`	Status.
`pct_complete`	Percentage completed.

`pr_cost`	Cost (monetary or time units).
`de_duty`	Description (up to 1000 characters).
k_x_duty_resource	Resources assigned to each task.
`gu_duty`	Task GUID.
`nm_resource`	Resource name.
`pct_time`	Percentage of time of the assigned resource.
k_duties_lookup	Lookup values for each task.
k_duties_attach	Attached documents for each task.
`gu_duty`	Task GUID.
`tx_file`	Document Name.
`len_file`	Document size in bytes.
`bin_file`	Document contents in binary form (BLOB).
k_duties_workreports	Work Reports.
`gu_workreport`	`Work Report` GUID.
`gu_project`	GUID of Project to which the Work Report belongs.
`gu_writer`	GUID of user who wrote the Work Report.
`dt_created`	Work Report Creation Date.
`tl_workreport`	Work report Title.
`de_workreport`	Briefing and general notes about the whole Work Report.
`tx_workreport`	Work Report XML contents.

k_bugs	Bugs/Incidents for each project.
`gu_bug`	Bug/Incident GUID.
`pg_bug`	Ordinal progressive value for bug/incident.
`tl_bug`	Bug/Incident name (title).
`gu_project`	Project of the Bug/Incident.
`dt_created`	Bug/Incident date of creation.
`gu_bug_ref`	GUID of a Bug/Incident similar to this one (for repeated bugs/incidents)
`dt_modified`	Bug/Incident date of last modification.
`dt_verified`	Bug/Incident date of verification.
`dt_closed`	Bug/Incident date of close.
`vs_found`	Version or serial number of the product where this bug/incident was found.
`vs_closed`	Version or serial number of the product where this bug/incident was closed/corrected.
`od_severity`	Severity.
`od_priority`	Priority.
`tx_status`	Status.
`nm_reporter`	Name of the person reporting the bug/incident.
`tx_rep_mail`	E-Mail address of the person reporting the bug/incident.
`nm_assigned`	Person assigned to the correction/follow-up of this bug/incident.
`nm_inspector`	Person who has inspected this bug/incident.
`id_ref`	Reference for interface with other apps.

	`id_client`	Customer's reference.
	`gu_writer`	GUID of user that last touched the bug.
	`tx_bug_brief`	Description of the bug/incident (up to 2000 characters).
	`tx_comments`	Comments (up to 1000 characters).
	k_bugs_lookup	Lookup values for each bug/incident.
	k_bugs_attach	Attached documents for each bug/incident.
	`gu_duty`	Bug/Incident GUID.
	`tx_file`	Document Name.
	`len_file`	Document Size in bytes.
	`bin_file`	Document contents in binary form (BLOB).
	k_bugs_changelog	Bugs change log.
	`gu_bug`	Bug/Incident GUID.
	`pg_bug`	Ordinal progressive value for bug/incident.
	`dt_modified`	Date when modification was done.
	`gu_writer`	GUID of the user who did the modification or `null` if it was an anonymous modification.
	`nm_column`	Name of column modified at table `k_bugs`.
	`tx_oldvalue`	First 255 characters of previous value at table `k_bugs`.
Views	**v_project_company**	Projects per company.
	v_duty_resource	Resources per task.
	v_duty_project	Tasks per project.

v_duty_company Tasks per company.



Concepts and Definitions

Distribution lists are used to send e-mail messages (generic or customized) to a set of email addresses (List Members).

e-mail addresses (Members) can be obtained and stored in several ways. Members Lists can be built in 3 different ways:

a. Getting the `tx_email` field from the `k_addresses` table, using the Addresses assigned to Companies in the `k_companies` table.
b. Getting the `tx_email` field from the `k_addresses` table, using the Addresses assigned to Contacts in the `k_contacts` table.
c. Loading them from a text file of importing them from a Windows Address Book file (Outlook Express).

On the other hand, Distribution Lists can be of 4 types:

a. **Dynamic Lists**. This lists obtain its members by running a stored query in the `k_queries` table. Each time that a email in scheduled for mailing, a new SQL query is launched, and members are recovered in real time.
b. **Static Lists**. This lists also obtains its members running a query on the database, but members are stored in the `k_x_list_members` table and it is never updated by the Distribution Lists module (although you can update members manually).
c. **Direct Lists**. This lists are directly loaded from a text file. Members don't have to be in `k_companies` or `k_contacts` tables.
d. **Black Lists**. This lists are a special kind of subset of email addresses to block email sending for specific members. A Black List is always associated with a Direct, Dynamic or Static list. The associated black list is that which `gu_query` field is the GUID of the base distribution list. Black Lists have type 4 at `tp_list` field. Thus, for finding the black lists associated to any given base list it is sufficient to use the SQL statement : `SELECT gu_list FROM k_lists WHERE gu_query='guid of base list' AND tp_list=4`

Tables

k_lists Distribution Lists.

`gu_list`	Distribution List GUID.
`dt_created`	Date of creation.
`gu_workarea`	GUID of the Workarea who owns this list.
`tp_list`	Distribution List Type. `1` Static List `2` Dynamic List `3` Direct List `4` Black List
`gu_query`	GUID of the SQL query associated to this list (only applicable to Dynamic lists) or GUID of the base list (only applicable to Black Lists)
`de_list`	List Description.
`tx_sender`	Sender's full name.
`tx_from`	Sender's email address.
`tx_reply`	e-mail address for the "Reply-To" field of the message.
`tx_subject`	e-mail's subject.

k_x_list_members Members for Static, Direct or Black Lists.

`gu_list`	Distribution List GUID.
`tx_email`	Member's email address.
`tx_name`	Member's Name.
`tx_surname`	Member's Surname.

`tx_salutation` Salutation (Mr/Ms) for this Member.

`bo_active` 1 if Member is active, 0 if Member is not active.

`dt_created` Entry's creation date.

`dt_modified` Entry's modification date.

`tp_member` Type of Member.
`90` Contact got from table `k_contacts`.
`91` Company got from table `k_companies`.
`95` Member loaded from a text file.

`gu_company` Company's GUID (if tp_member=91).

`gu_company` Contact's GUID (if tp_member=90).

`id_format` Preferred format for e-mail messages.
`TXT` Plain Text.
`HTML` HTML.

k_member_address The hipergate data model allows multiple contacts per company and multiple addresses per contact.
The flexibility of this schema implies the need of at least five tables for storing the data: `k_companies`, `k_contacts`, `k_addresses`, `k_x_company_addr` and `k_x_contact_addr`.
In many occasions it is convenient to access addresses as if they were in a single table. At versions 1.x this is achieved using `v_member_address` view. It is a complex view, heavy to execute by the database management system.
For improved performance, from version 2.0 the table `k_member_address` keeps a copy of all addresses associated to companies or contacts. This table is automatically maintained by triggers. Each time an address, company or contact is inserted the triggers modify `k_member_address` table.
If the table is deleted by mistake it may be restored simply executing the following SQL

		statement: `INSERT INTO k_member_address SELECT * FROM k_member_address`.
Views	**v_member_address**	This view is the union of three sets: { Active Addresses for Companies ∪ Active Addresses for Contacts assigned to a Company ∪ Active Address for Contacts not assigned to a Company }

Task Scheduler Submodel

Tables and Views

Tables	**k_jobs**	Jobs.
	`gu_job`	Job's GUID.
	`gu_workarea`	GUID of the Workarea who owns this Job.
	`gu_writer`	GUID of the user who created this Job.
	`id_command`	Command associated to this Job (from the `k_lu_job_commands` table).
	`id_status`	Status of this Job (from the `k_lu_job_status` table).
	`dt_created`	Job's date of creation.
	`tl_job`	Job's Title.
	`gu_job_group`	Jobs Group GUID (not yet implemented).
	`tx_parameters`	Extra parameters for command. This is a comma-separated list of fields and values, for example: "gu_pageset:P012345,gu_list:List1". Names and values are separated by colon.

	Client applications must parse this field to use the values stored here.
`dt_execution`	Full date when the Job must start or NULL if Job must start as soon as possible.
`dt_finished`	Date when Job finished or NULL of execution has not finished yet.
`dt_modified`	Entry's last modification date.
k_job_atoms	Atoms per Jobs pending to be executed. Each atom represents an email address to send a message, a fax number or a FTP URL to upload a file. To improve the access to database, each atom has the most typical fields for a Job.
`gu_job`	Job's GUID.
`pg_atom`	Ordinal progressive value for Atom.
`dt_execution`	Requested date of execution.
`id_status`	Status for Atom's process.
`id_format`	Data format (typically "TXT", "HTML").
`gu_company`	Company's GUID for postal, email of fax sending.
`nm_commercial`	Company's Commercial Name.
`gu_contact`	Contact's GUID.
`tx_email`	Destination e-mail Address.
`tx_name`	Recipient's Name.
`tx_surname`	Recipient's Surname.
`tx_salutation`	Recipient's Salutation (Mr/Ms).
`tp_street`	Street Type.

`nu_street` Street Number.

`tx_addr1` Street Address (row 1).

`tx_addr2` Street Address (row 2).

`nm_country` Country Name.

`nm_state` Province/State Name.

`mn_city` City Name.

`zipcode` ZIP/Postal Code.

`work_phone` Work Phone Number.

`direct_phone` Direct Work Phone Number.

`home_phone` Personal Phone Number.

`mov_phone` Mobile/Cell Phone Number.

`fax_phone` Fax Number.

`other_phone` Other Phone Number.

`po_box` P.O. Box.

k_job_archived Job's Atoms that have been executed.

k_lu_job_status Status lookup for Jobs.

Status Code	Description
-1	Canceled
0	Pending
1	Finished
2	Paused
3	Running

k_job_atoms_tracking Atoms Tracking. This table is used for receiving notifications from web beacons signaling read e-mails.

`gu_job`	Job's GUID.
`pg_atom`	Ordinal progressive value for Atom.
`gu_company`	GUID of recipient Company.
`gu_contact`	GUID of recipient Contact.
`ip_addr`	IP address of client machine
`tx_email`	Recipient e-mail address.

k_lu_job_commands Allowed Jobs' Commands.

`id_command` Command's Code.

Command Code	Description
VOID	Do nothing
MAIL	Send a newsletter
SEND	Send an e-mail
FAX	Send a fax
NTFY	Notify event by e-mail
SAVE	Store in local disk
FTP	Store in FTP Server

`tx_command` Command's Description.

`nm_class` Full name of the Java subclass that implements this command.

3 Stored Procedures

Stored Procedures PL/SQL (Oracle), Transact-SQL (MS SQL Server) or PL/pgSQL (PostgreSQL) are used for 2 purposes: 1st) get better performance and 2nd) externalize dependency management outside compiled Java code.

Where to find stored procedures source code

Look at **com/knowgate/hipergate/datamodel/procedures** at **hipergate.jar**. They are divided per database and functional module.

Why stored procedures?

Often when a multi-platform development is initiated, one of the first things done is discarding the use of proprietary database extensions. This decision is made with the objective of reducing porting effort.
By removing stored procedures, one of the most powerful database optimization techniques is lost.

hipergate use stored procedures where they can contribute to better performance and maintainability at the cost of making the code harder to port to new database management systems. Using stored procedures contribute to improve product quality in several ways

1º) Stores procedures are the fastest method for executing precompiled batches of SQL sentences.

2º) Stored procedures reduce network traffic between web server and database server.

3º) For functions that perform a lookup at the database and return a single value, it is more efficient to use stored procedures return values than generating a recordset.

4º) Stored procedures code is easier to modify than compiled Java code.

How to call stored procedures from Java

Usually there is no need to call directly the stored procedures from JSP code, since most procedures are written specifically for accelerating a particular process and are called from inside compiled Java methods from hipergate standard classes.

In this chapter we show how procedure `k_sp_autenticate` is called from method `autenticate()` of class `com.knowgate.ACL`.

This example is interesting because it shows differences that exist among database management systems.

```
public static short autenticate
(JDCConnection oConn, String sUserId, String sAuthStr, int iFlags)
throws SQLException, UnsupportedOperationException {

  /* Search a User at table k_users and verify if password given as
   parameter is the same as that stored at k_users table and, if
   so, check that the user is active and that password has not
   expired */

  short iStatus;
  CallableStatement oCall;
  Statement oStmt;
  ResultSet oRSet;
  String sPassword;

  switch (oConn.getDataBaseProduct()) {

    case JDCConnection.DBMS_ORACLE:

    oCall = oConn.prepareCall("{ call k_sp_autenticate (?,?,?)}");

    oCall.setString(1, sUserId);
    oCall.setString(2, sAuthStr);
    oCall.registerOutParameter(3, java.sql.Types.DECIMAL);

    oCall.execute();

     /* In Oracle there is no difference between SMALLINT and
        NUMBER. So here the result is retrieved as a decimal
        number and then parsed and converted to a 16 bits integer.
        Usually the JDBC driver can perform this conversion
        without notice if a Short value is requested as return
        value for the procedure.
     */

    iStatus = Short.parseShort(oCall.getBigDecimal(1).toString());

    oCall.close();
// Always close you statements or you will soon run out of them

    break;

    case JDCConnection.DBMS_MSSQL:

    oCall = oConn.prepareCall("{ call k_sp_autenticate (?,?,?)}");

    oCall.setString(1, sUserId);
    oCall.setString(2, sAuthStr);
    oCall.registerOutParameter(3, java.sql.Types.SMALLINT);

    oCall.execute();

    iStatus = oCall.getShort(3);

    oCall.close();
```

```
        break;

        case JDCConnection.DBMS_POSTGRESQL:

         /* In PostgreSQL k_sp_autenticate is PL/pgSQL function. In
            this case a ResultSet is used for retrieving function
            result.
         */

        oStmt = oConn.createStatement();

        oRSet = oStmt.executeQuery("SELECT k_sp_autenticate('"+sUserId
+ "','" + sPassword + "')");

        oRSet.next();

        iStatus = oRSet.getShort(1);

        oRSet.close();
        oStmt.close();
        break;

        default:
          throw new UnsupportedOperationException("proc. not found");

      } // end switch

      return iStatus;

    } // autenticate
```

Stored Procedures for Security and User Login

They can be found at **security.ddl** file on each DBMS folder under bajo **com/knowgate/hipergate/datamodel/procedures**.

k_get_domain_id

Get Domain numeric identifier from its name. Search is case sensitive.

`NmDomain VARCHAR`	Name of domain being searched.
`IdDomain INTEGER OUT`	Domain Id. or zero if not found.

k_get_workarea_id

Get Workarea GUID from its name. Search is case sensitive.

`NmWorkArea VARCHAR`	Name of Workarea being searched.
`IdDomain INTEGER`	Identifier to which Workarea belongs
`IdWorkArea CHAR OUT`	Workarea GUID or NULL if not found.

k_is_workarea_admin

Check whether a user belongs to Administrators Group of a Workarea

`IdWorkArea CHAR`	Workarea GUID.
`IdUser CHAR`	User GUID.
`IsAdmin INTEGER OUT`	1 if user belongs to Administrators Group, 0 if not.

k_is_workarea_ poweruser

Check whether a user belongs to Power Users Group of a Workarea

`IdWorkArea CHAR`	Workarea GUID.
`IdUser CHAR`	User GUID.
`IsPowUser INTEGER OUT`	1 if user belongs to Power Users Group, 0 if not.

k_is_workarea_ user

Check whether a user belongs to Users Group of a Workarea

`IdWorkArea CHAR`	Workarea GUID.
`IdUser CHAR`	User GUID.
`IsUser INTEGER OUT`	1 if user belongs to Users Group, 0 if not.

k_is_workarea_ guest

Check whether a user belongs to Guests Group of a Workarea

`IdWorkArea CHAR`	Workarea GUID.
`IdUser CHAR`	User GUID.
`IsGuest INTEGER OUT`	1 if user belongs to Guests Group, 0 if not.

k_get_user_from_email

Get a User GUID from his main e-mail address. E-mail addresses for users are unique across domains.

`TxMainEmail VARCHAR`	e-mail address from field `tx_main_email` of `k_users` table.
`Iduser CHAR OUT`	User GUID or NULL if e-mail was not found.

k_get_user_from_nick

Get User GUID from his nickname. Nicknames may be repeated from different users across domains.

`IdDomain INTEGER`	Identifier of Domain to which searched user must belong.
`TxNick VARCHAR`	User nickname.
`IdUser CHAR OUT`	User GUID or NULL if no user with such nickname was found at given domain.

k_get_group_id

Get Group GUID from its name. Search is case sensitive.

`IdDomain INTEGER`	Identifier of Domain to which group belongs.
`NmGroup VARCHAR`	Group Name.
`IdGroup CHAR OUT`	Group GUID or NULL if no group with such name was found at given domain.

k_sp_autenticate

Verify that a user/password pair for checking if user have access to the system. Password checking is case sensitive.

`IdDomain CHAR`	GUID of User to authenticate.
`PwdText VARCHAR`	Password.
`CoStatus SMALLINT OUT`	Result: 1 User and password are valid. -1 No user with such GUID exists. -2 Password is not correct. -3 User is deactivated (field `bo_active` is set to zero). -8 User login has expired (current date is greater than `dt_cancel`). -9 Password has expired.

k_sp_del_group

Delete a Users Group.

`IdGroup CHAR`	GUID of group to be deleted.

k_sp_del_user

Delete a user.

`IdUser CHAR`	GUID of User to be deleted.

☞ This stored procedured must not be used for directly deleting a User in a production environment. Many columns from several submodules have a foreign key reference to `k_users` table.

Java method `com.knowgate.acl.ACLUser.delete()` can be used to delete additional references on cascade; but even this Java method is unable to delete additional references not in the standard distro data model.

The preferable approach is never deleting user but simply deactivate them by setting `k_users.bo_active` to zero and `k_users.dt_cancel` to current date.

In case that it is really neccesary to delete a User, may be desirable to write an special purpose clean up procedure ran before actual user deletion.

Category Management Stored Procedures

k_sp_get_cat_id

Get a Category GUID from its name.

`NmCategory VARCHAR`	Category Name.
`IdCategory CHAR OUT`	Category GUID or NULL if not found.

k_sp_cat_descendant

Verifies if a Category is descendant of another one.

`IdCategory CHAR`	GUID of child Category.
`IdAncestor CHAR`	GUID of parent Category.
`BoChild SMALLINT OUT`	1 if Category `IdCategory` is descendant of `IdAncestor`, 0 if not.

This function does not exist on PostgreSQL.

k_sp_cat_level

Get depth level of a Category at the tree.

`IdCategory CHAR` Category GUID.

`CatLevel INTEGER OUT` Depth level (1 for root categories).

k_sp_del_category

Delete a Category.

`IdCategory CHAR` Category GUID.

☞ Because a Category can have an arbitrary number of associated objects through `k_x_cat_objs`, it is neccesary to use `com.knowgate.hipergate.Category` Java class for deleting categories. Calling `k_sp_del_category` directly may leave leaked objects such as forums or documents existing at the database but impossible to reach.

k_sp_del_category_r

Delete a Category and all its descendants.

`IdCategory CHAR` Category GUID.

k_sp_get_cat_path

Compose a path for a Category by concatenating `nm_category` field from all its ancestors separated by a slash '/' character. This procedure is used for creating file paths for categories.

`IdCategory CHAR` Category GUID.

`CatPath VARCHAR OUT` Category Path.

k_sp_cat_obj_position

`IdCategory CHAR` Category GUID.

`OdPosition INTEGER OUT` Position of object inside Category or NULL if object is not contained into category.

k_sp_cat_expand

Expand all descendants from a Category writing results at table `k_cat_expand`.
For performance reasons, in some cases it is convenient to have a pre-expanded list of all childs and grand childs from a category.

This procedure deletes all previous descendants at `k_cat_expand` and rewrites them.

`IdCategory CHAR`	GUID of category to be expanded.

☞ Implementation used for expansion varies from one database to another. In Oracle descendants are expanded using `SELECT … FROM k_cat_tree START WITH gu_parent_cat = ? CONNECT BY gu_parent_cat = PRIOR gu_child_cat` sentence. In Microsoft SQL Server a temporary table is used as a stack pile. In PostgreSQL recursive calls to `k_sp_cat_expand_node` function are used.

k_sp_cat_usr_perm

Get permissions of a User over a category, taking into account permissions granted directly and permissions granted by making the user member of a group that have permissions over the category.

In no explicit permission are given for the user over the category then permissions are taken from category parent or grandparents.

`IdUser CHAR`	User GUID.
`IdCategory CHAR`	Category GUID.
`AclMask INTEGER OUT`	Permissions bit mask of user for the category. See `bit_mask` field of `k_lu_permissions`.

k_sp_cat_del_grp

Remove permissions of a Group over a Category.

`IdCategory CHAR`	Category GUID.
`IdGroup CHAR`	Group GUID.
`Recurse SMALLINT`	Apply removal recursively to child categories.
`Objects SMALLINT`	Not used, must be zero.

k_sp_cat_del_usr

Remove permissions of a User over a category.

`IdCategory CHAR`	Category GUID.
`IdUser CHAR`	User GUID.
`Recurse SMALLINT`	Apply removal recursively to child categories.
`Objects SMALLINT`	Not used, must be zero.

k_sp_cat_set_grp

Grant permissions to a Group over a Category. If Group already had permissions over Category a duplicated primary key exception on `k_x_cat_group_acl` table will be raised.

`IdCategory CHAR`	Category GUID.
`IdGroup CHAR`	Group GUID.
`Recurse SMALLINT`	Apply grant recursively to child categories.
`Objects SMALLINT`	Not used, must be zero.

k_sp_cat_set_usr

Grant permissions to a User over a Category. If User already had permissions over Category a duplicated primary key exception on `k_x_cat_user_acl` table will be raised.

`IdCategory CHAR`	Category GUID.
`IdUser CHAR`	User GUID.
`Recurse SMALLINT`	Apply grant recursively to child categories.
`Objects SMALLINT`	Not used, must be zero.

Workgroup Stored Procedures

Can be found at file **addrbook.ddl** under **com/knowgate/hipergate/ data model/procedures**.

k_ sp_del_ meeting

Delete Meeting.

`MeetingId CHAR`	Meeting GUID.

k_ sp_del_ fellow

Delete Employee. Meeting for employee will be erased on cascade.

`FellowId CHAR`	Employee GUID.

k_ sp_del_ room

Delete Room or Shared Resource. If resource is in use by any meeting a foreign key violation exception will be raised.

`RoomNm`	`VARCHAR`	Room Name.
`WorkAreaId`	`VARCHAR`	Workarea GUID.

Product Management Stored Procedures

Can be found at file **products.ddl** under **www.bajo.com/knowgate/hipergate/ data model/procedures**.

k_ sp_del_ product

Delete Product.

`ProductId`	`CHAR`	Product GUID.

☞ Products must be deleted using Java method `delete()` from class `com.knowgate.hipergate.Product` and not calling directly to `k_sp_del_product`. Database stored procedures cannot delete external disk files. If `k_sp_del_product` is executed directly zombie files will be left at hard drive.

Customer Relationship Management Stored Procedures

Can be found at file **crm.ddl** under **com/knowgate/hipergate/datamodel/ procedures**.

k_ sp_del_sales_man

Delete Salesman.

`SalesManId`	`CHAR`	Salesman GUID.

☞ Salesmen are sub registers of Domain Users. When a Salesman is deleted Companies assigned to him will be set free of any assignment. Deleting a Salesman does not delete the underlying Domain User. Orders placed by a deleted Salesman are not deleted when he disappears.

k_ sp_del_contact

Delete Contact.

`ContactId`	`CHAR`	Contact GUID.

☞ Contacts must be deleted by calling Java method `delete()` from class `com.knowgate.crm.Contact` and not directly calling `k_sp_del_contact` because database stored procedures cannot delete external disk files attached to the Contact.

k_ sp_del_company

Delete Company.

`CompanyId CHAR`	Company GUID.

☞ Before deleting a Company it is neccesary to delete its Contacts. This is automatically done by Java class `com.knowgate.crm.Company`

k_ sp_del_oportunity

Delete Opportunity.

`OportunityId CHAR`	Opportunity GUID.

Task Scheduler Submodel

Tables and Views

Tables	**k_jobs**	Jobs.
	`gu_job`	Job's GUID.
	`gu_workarea`	GUID of the Workarea who owns this Job.
	`gu_writer`	GUID of the user who created this Job.
	`id_command`	Command associated to this Job (from the `k_lu_job_commands` table).
	`id_status`	Status of this Job (from the `k_lu_job_status` table).
	`dt_created`	Job's date of creation.
	`tl_job`	Job's Title.

`gu_job_group`	Jobs Group GUID (not yet implemented).
`tx_parameters`	Extra parameters for command. This is a comma-separated list of fields and values, for example: "gu_pageset:P012345,gu_list:List1". Names and values are separated by colon. Client applications must parse this field to use the values stored here.
`dt_execution`	Full date when the Job must start or NULL if Job must start as soon as possible.
`dt_finished`	Date when Job finished or NULL of execution has not finished yet.
`dt_modified`	Entry's last modification date.

k_job_atoms	Atoms per Jobs pending to be executed. Each atom represents an email address to send a message, a fax number or a FTP URL to upload a file. To improve the access to database, each atom has the most typical fields for a Job.
`gu_job`	Job' GUID.
`pg_atom`	Ordinal progressive value for Atom.
`dt_execution`	Requested date of execution.
`id_status`	Status for Atom's process.
`id_format`	Data format (typically "TXT", "HTML").
`gu_company`	Company's GUID for postal, email of fax sending.
`nm_commercial`	Company's Commercial Name.
`gu_contact`	Contact's GUID.
`tx_email`	Destination e-mail Address.
`tx_name`	Recipient's Name.

`tx_surname`	Recipient's Surname.
`tx_salutation`	Recipient's Salutation (Mr/Ms).
`tp_street`	Street Type.
`nu_street`	Street Number.
`tx_addr1`	Street Address (row 1).
`tx_addr2`	Street Address (row 2).
`nm_country`	Country Name.
`nm_state`	Province/State Name.
`mn_city`	City Name.
`zipcode`	ZIP/Postal Code.
`work_phone`	Work Phone Number.
`direct_phone`	Direct Work Phone Number.
`home_phone`	Personal Phone Number.
`mov_phone`	Mobile/Cell Phone Number.
`fax_phone`	Fax Number.
`other_phone`	Other Phone Number.
`po_box`	P.O. Box.

k_job_archived Job's Atoms that have been executed.

k_lu_job_status Status lookup for Jobs.

Status Code	Description
-1	Canceled
0	Pending
1	Finished

2	Paused
3	Running

k_lu_job_commands Allowed Jobs' Commands.

`id_command` Command's Code.

Command Code	Description
MAIL	Send an email
FAX	Send a fax
SAVE	Store in local disk
FTP	Store in FTP Server

`tx_command` Command's Description.

`nm_class` Full name of the Java subclass that implements this command.

Hipermail submodule

Mail storage

hipergate 2.1 and above have a local storage provider for JavaMail.

hipergate uses an hybrid mail storage system using flat files and a database. Messages are stored at disk files with MBOX format. MBOX files are just a concatenation of RFC 822 messages. MBOX files provide no indexing capabilities, so message search and retrieval must be implemented separately.

hipergate uses a relational database and Jakarta Lucene for indexing messages.

hipergate may store MIME messages in MBOX files or inside LONGVARBINARY columns at the database. By default MBOX storage is used. It is not recommended to store the whole message at the database since mailboxes may grow very large affecting database performance and back-up times.

Even if MBOX storage is selected, the database is still used for indexing.

There is an MBOX file for each messages folder. Folders are a subclass of hipergate Categories. Indeed there are no explicit entries for folders at the database but Categories are used.

MBOX files are placed under its corresponding category directory at /storage branch. For example for user with alias *ad6148* at domain TEST his path to the inbox file would look like:

```
…/storage/domains/2049/workareas/c0a801bffec3a42d52100000e12e2153/
ROOT/DOMAINS/TEST/TEST_USERS/TEST_ad6148/TEST_ad6148_email/TEST_ad
6148_inbox/ TEST_ad6148_inbox.mbox
```

2049 is the TEST domain numeric identifier and c0a801… is the Workarea GUID.

☞ When the default Workarea for a User is changed, its MBOX files will not be moved. This must be taken into account is any process relying on the Workarea GUID for composing a path is custom coded.

Linkage between e-mails and hipergate CRM

Each time a message is sent or received, its mail addresses are checked against `k_users` and `k_member_address` tables. If a User, Contact or Company with that e-mail exists at the same Workarea as the e-mail, then a reference between the recipient and the hipergate entity is recorded at `k_inet_addrs` table. This allows tracking all sent end received e-mails for Users and Contacts. In order for the linkage to take place, messages must be sent using hipergate and must be opened using hipermail web interface. Messages are not scanned for matching recipients until they are opened for the first time.

Message cache

Messages are not locally stored by hipergate until they are opened for the first time. Once the message has been opened, it is fully downloaded and stored at inbox local folder.

k_categories	This table holds message Folders.
k_mime_msgs	Mime messages index.
`gu_mimemsg`	Message GUID. This is a 32 characters internal hipergate GUID not the RFC 822 message id.
`gu_workarea`	GUID of Workarea to which message belongs.
`pg_message`	Message ordinal position within folder
`gu_category`	GUID of folder that contains the message. Mail folders are a subclass of Categories from `k_categories` table. There is no special register for mail folders but categories are used directly. For each user the application creates by default 6 folders under user's home category: *inbox*, *drafts*, *sent*, *received*, *delete* and *spam*.
`gu_parent_msg`	GUID of parent message. This is only used for messages that are attached inside other top level messages.
`nu_position`	Byte offset position of message inside MBOX file.
`id_message`	Message Id. generated by the mail server.
`len_mimemsg`	Message length in bytes.
…	
`by_content`	This column holds message main body, either plain text or html. It is used as a cache for rapidly displaying message content without accessing any other file or database register. The same text is contained in MBOX file or in one of the message parts at `k_mime_parts` depending on whether MBOX or BLOB storage is used.
k_mime_parts	Message parts.

`gu_mimemsg`	Message GUID.
`id_message`	Message Id. generated by the mail server.
`pg_message`	Message ordinal position within folder
`nu_offset`	Byte offset of part from message position.
`id_disposition`	May be `inline`, `attachment`, `reference` or `pointer`. `inline` and `attachment` are used by final messages whilst `reference` and `pointer` are used by draft messages. `reference` means that the part is not contained at an MBOX file or at the "`by_content`" column but at an external file. For reference parts, `file_name` columns hold the full path to the file. `pointer` means that the part is taken from another MBOX file.
`by_content`	Part source. If MBOX storage mode is used, then this column is always NULL. ☞ When using BLOB storage, message source is not completely stored anywhere, instead, each message part is stored separately at k_mime_parts. This allows reading or downloading an attachment without having to read the whole message.
k_inet_addrs	Message recipients.
`gu_mimemsg`	Message GUID.
`id_message`	Message Id. generated by the mail server.
`pg_message`	Message ordinal position within folder.
`tx_email`	Recipient e-mail address.
`tp_recipient`	{ `from` | `to` | `cc` | `bcc` }.
`tx_personal`	Recipient full name.
`gu_user`	User whose `tx_main_email` at `k_users` table is the same as `tx_email`.

`gu_contact`	Contact whose mail address at `k_mem_address` table is the same as `tx_email`.
`gu_company`	Company whose mail address at `k_member_address` table is the same as `tx_email`.

3 Stored Procedures

Stored Procedures PL/SQL (Oracle), Transact-SQL (MS SQL Server) or PL/pgSQL (PostgreSQL) are used for 2 purposes: 1st) get better performance and 2nd) externalize dependency management outside compiled Java code.

Where to find stored procedures source code

Look at **com/knowgate/hipergate/datamodel/procedures** at **hipergate.jar**. They are divided per database and functional module.

Why stored procedures?

Often when a multi-platform development is initiated, one of the first things done is discarding the use of proprietary database extensions. This decision is made with the objective of reducing porting effort.
By removing stored procedures, one of the most powerful database optimization techniques is lost.

hipergate use stored procedures where they can contribute to better performance and maintainability at the cost of making the code harder to port to new database management systems. Using stored procedures contribute to improve product quality in several ways

1º) Stores procedures are the fastest method for executing precompiled batches of SQL sentences.

2º) Stored procedures reduce network traffic between web server and database server.

3º) For functions that perform a lookup at the database and return a single value, it is more efficient to use stored procedures return values than generating a recordset.

4º) Stored procedures code is easier to modify than compiled Java code.

How to call stored procedures from Java

Usually there is no need to call directly the stored procedures from JSP code, since most procedures are written specifically for accelerating a particular process and are called from inside compiled Java methods from hipergate standard classes.

Anyway, in this chapter it is shown how procedure `k_sp_autenticate` is called from method `autenticate()` of class `com.knowgate.ACL`.

This example is interesting because it shows differences that exist among database management systems.

```java
public static short autenticate
(JDCConnection oConn, String sUserId, String sAuthStr, int iFlags)
throws SQLException, UnsupportedOperationException {

  /* Search a User at table k_users and verify if password given as
   parameter is the same as that stored at k_users table and, if
   so, check that the user is active and that password has not
   expired */

  short iStatus;
  CallableStatement oCall;
  Statement oStmt;
  ResultSet oRSet;
  String sPassword;

  switch (oConn.getDataBaseProduct()) {

    case JDCConnection.DBMS_ORACLE:

    oCall = oConn.prepareCall("{ call k_sp_autenticate (?,?,?)}");

    oCall.setString(1, sUserId);
    oCall.setString(2, sAuthStr);
    oCall.registerOutParameter(3, java.sql.Types.DECIMAL);

    oCall.execute();

     /* In Oracle there is no difference between SMALLINT and
        NUMBER. So here the result is retrieved as a decimal
        number and then parsed and converted to a 16 bits integer.
        Usually the JDBC driver can perform this conversion
        without notice if a Short value is requested as return
        value for the procedure.
     */

    iStatus = Short.parseShort(oCall.getBigDecimal(1).toString());

    oCall.close();
// Always close you statements or you will soon run out of them

    break;

    case JDCConnection.DBMS_MSSQL:

    oCall = oConn.prepareCall("{ call k_sp_autenticate (?,?,?)}");

    oCall.setString(1, sUserId);
    oCall.setString(2, sAuthStr);
    oCall.registerOutParameter(3, java.sql.Types.SMALLINT);

    oCall.execute();

    iStatus = oCall.getShort(3);

    oCall.close();
    break;

    case JDCConnection.DBMS_POSTGRESQL:

     /* In PostgreSQL k_sp_autenticate is PL/pgSQL function. In
        this case a ResultSet is used for retrieving function
        result.
```

```
     */

    oStmt = oConn.createStatement();

    oRSet = oStmt.executeQuery("SELECT k_sp_autenticate('"+sUserId
+ "','" + sPassword + "')");

    oRSet.next();

    iStatus = oRSet.getShort(1);

    oRSet.close();
    oStmt.close();
    break;

    default:
      throw new UnsupportedOperationException("proc. not found");

  } // end switch

  return iStatus;

} // autenticate
```

Stored Procedures for Security and User Login

They can be found at **security.ddl** file on each DBMS folder under bajo **com/knowgate/hipergate/datamodel/procedures**.

k_get_domain_id

Get Domain numeric identifier from its name. Search is case sensitive.

`NmDomain VARCHAR`	Name of domain being searched.
`IdDomain INTEGER OUT`	Domain Id. or zero if not found.

k_get_workarea_id

Get Workarea GUID from its name. Search is case sensitive.

`NmWorkArea VARCHAR`	Name of Workarea being searched.
`IdDomain INTEGER`	Identifier to which Workarea belongs
`IdWorkArea CHAR OUT`	Workarea GUID or NULL if not found.

k_is_workarea_admin

Check whether a user belongs to Administrators Group of a Workarea

`IdWorkArea CHAR`	Workarea GUID.
`IdUser CHAR`	User GUID.
`IsAdmin INTEGER OUT`	1 if user belongs to Administrators Group, 0 if not.

k_is_workarea_ poweruser

Check whether a user belongs to Power Users Group of a Workarea

`IdWorkArea CHAR`	Workarea GUID.
`IdUser CHAR`	User GUID.
`IsPowUser INTEGER OUT`	1 if user belongs to Power Users Group, 0 if not.

k_is_workarea_ user

Check whether a user belongs to Users Group of a Workarea

`IdWorkArea CHAR`	Workarea GUID.
`IdUser CHAR`	User GUID.
`IsUser INTEGER OUT`	1 if user belongs to Users Group, 0 if not.

k_is_workarea_ guest

Check whether a user belongs to Guests Group of a Workarea

`IdWorkArea CHAR`	Workarea GUID.
`IdUser CHAR`	User GUID.
`IsGuest INTEGER OUT`	1 if user belongs to Guests Group, 0 if not.

k_get_user_from_email

Get a User GUID from his main e-mail address. E-mail addresses for users are unique across domains.

`TxMainEmail VARCHAR`	e-mail address from field `tx_main_email` of `k_users` table.
`Iduser CHAR OUT`	User GUID or NULL if e-mail was not found.

k_get_user_from_nick

Get User GUID from his nickname. Nicknames may be repeated from different users across domains.

`IdDomain INTEGER`	Identifier of Domain to which searched user must belong.
`TxNick VARCHAR`	User nickname.
`IdUser CHAR OUT`	User GUID or NULL if no user with such nickname was found at given domain.

k_get_group_id

Get Group GUID from its name. Search is case sensitive.

`IdDomain INTEGER`	Identifier of Domain to which group belongs.
`NmGroup VARCHAR`	Group Name.
`IdGroup CHAR OUT`	Group GUID or NULL if no group with such name was found at given domain.

k_sp_autenticate

Verify that a user/password pair for checking if user have access to the system. Password checking is case sensitive.

`IdDomain CHAR`	GUID of User to authenticate.
`PwdText VARCHAR`	Password.
`CoStatus SMALLINT OUT`	Result: 1 User and password are valid. -1 No user with such GUID exists. -2 Password is not correct. -3 User is deactivated (field `bo_active` is set to zero). -8 User login has expired (current date is greater than `dt_cancel`). -9 Password has expired.

k_sp_del_group

Delete a Users Group.

`IdGroup CHAR`	GUID of group to be deleted.

k_sp_del_user

Delete a user.

`IdUser CHAR`	GUID of User to be deleted.

☞ This stored procedured must not be used for directly deleting a User in a production environment. Many columns from several submodules have a foreign key reference to `k_users` table.

Java method `com.knowgate.acl.ACLUser.delete()` can be used to delete additional references on cascade; but even this Java method is unable to delete additional references not in the standard distro data model.

The preferable approach is never deleting user but simply deactivate them by setting `k_users.bo_active` to zero and `k_users.dt_cancel` to current date.

In case that it is really neccesary to delete a User, may be desirable to write an special purpose clean up procedure ran before actual user deletion.

Category Management Stored Procedures

k_sp_get_cat_id

Get a Category GUID from its name.

`NmCategory VARCHAR`	Category Name.
`IdCategory CHAR OUT`	Category GUID or NULL if not found.

k_sp_cat_descendant

Verifies if a Category is descendant of another one.

`IdCategory CHAR`	GUID of child Category.
`IdAncestor CHAR`	GUID of parent Category.
`BoChild SMALLINT OUT`	1 if Category `IdCategory` is descendant of `IdAncestor`, 0 if not.

This function does not exist on PostgreSQL.

k_sp_cat_level

Get depth level of a Category at the tree.

`IdCategory CHAR`	Category GUID.
`CatLevel INTEGER OUT`	Depth level (1 for root categories).

k_sp_del_category

Delete a Category.

`IdCategory CHAR` Category GUID.

☞ Because a Category can have an arbitrary number of associated objects through `k_x_cat_objs`, it is neccesary to use `com.knowgate.hipergate.Category` Java class for deleting categories. Calling `k_sp_del_category` directly may leave leaked objects such as forums or documents existing at the database but impossible to reach.

k_sp_del_category_r

Delete a Category and all its descendants.

`IdCategory CHAR` Category GUID.

k_sp_get_cat_path

Compose a path for a Category by concatenating `nm_category` field from all its ancestors separated by a slash '/' character. This procedure is used for creating file paths for categories.

`IdCategory CHAR` Category GUID.

`CatPath VARCHAR OUT` Category Path.

k_sp_cat_obj_position

`IdCategory CHAR` Category GUID.

`OdPosition INTEGER OUT` Position of object inside Category or NULL if object is not contained into category.

k_sp_cat_expand

Expand all descendants from a Category writing results at table `k_cat_expand`.
For performance reasons, in some cases it is convenient to have a pre-expanded list of all childs and grandchilds from a category.

This procedure deletes all previous descendants at `k_cat_expand` and rewrites them.

`IdCategory CHAR` GUID of category to be expanded.

☞ Implementation used for expansion varies from one database to another. In Oracle descendants are expanded using `SELECT … FROM k_cat_tree START WITH gu_parent_cat = ? CONNECT BY gu_parent_cat = PRIOR gu_child_cat` sentence. In Microsoft SQL Server a temporary table is used as a stack pile. In PostgreSQL recursive calls to `k_sp_cat_expand_node` function are used.

k_sp_cat_usr_perm

Get permissions of a User over a category, taking into account permissions granted directly and permissions granted by making the user member of a group that have permissions over the category.

In no explicit permission are given for the user over the category then permissions are taken from category parent or grandparents.

`IdUser CHAR`	User GUID.
`IdCategory CHAR`	Category GUID.
`AclMask INTEGER OUT`	Permissions bit mask of user for the category. See `bit_mask` field of `k_lu_permissions`.

k_sp_cat_del_grp

Remove permissions of a Group over a Category.

`IdCategory CHAR`	Category GUID.
`IdGroup CHAR`	Group GUID.
`Recurse SMALLINT`	Apply removal recursively to child categories.
`Objects SMALLINT`	Not used, must be zero.

k_sp_cat_del_usr

Remove permissions of a User over a category.

`IdCategory CHAR`	Category GUID.
`IdUser CHAR`	User GUID.
`Recurse SMALLINT`	Apply removal recursively to child categories.
`Objects SMALLINT`	Not used, must be zero.

k_sp_cat_set_grp

Grant permissions to a Group over a Category. If Group already had permissions over Category a duplicated primary key exception on `k_x_cat_group_acl` table will be raised.

`IdCategory CHAR`	Category GUID.
`IdGroup CHAR`	Group GUID.
`Recurse SMALLINT`	Apply grant recursively to child categories.
`Objects SMALLINT`	Not used, must be zero.

k_sp_cat_set_usr

Grant permissions to a User over a Category. If User already had permissions over Category a duplicated primary key exception on `k_x_cat_user_acl` table will be raised.

`IdCategory CHAR`	Category GUID.
`IdUser CHAR`	User GUID.
`Recurse SMALLINT`	Apply grant recursively to child categories.
`Objects SMALLINT`	Not used, must be zero.

k_sp_get_user_mailroot

Get User mailroot category. Mailroot category is always under User home category which is the category referenced from column `gu_category` of table `k_users`. Mailroot category is identified by following a naming convention: its `nm_category` column must be: *DOMAIN_nickname*_mail being DOMAIN the name of the Domain (`k_domains.nm_domain`) to which user belongs and nickname the value of `k_users.tx_nickname`.

☞ Because User's `tx_nickname` is used for identifying his mailroot category by name, changing nickname after user has created will cause loss of mailroot category.

`GuUser CHAR`	User GUID.
`GuCategory CHAR OUT`	GUID of mailroot Category or NULL is it is not found.

k_sp_get_user_mailfolder

Get User mail folder. Mail folders must be a first level child of User's mailroot category.

`GuUser CHAR`	User GUID.

`NmFolder VARCHAR`	Folder Name. It may be either as `nm_category` from `k_categories`. Or a shortened name like : `inbox`, `outbox`, `drafts`, `deleted`, `sent`, `spam`, `received` or `templates`.
`GuCategory CHAR OUT`	GUID of mail folder or NULL if not found.

Workgroup Stored Procedures

Can be found at file **addrbook.ddl** under **com/knowgate/hipergate/datamodel/procedures**.

k_ sp_del_ meeting

Delete Meeting.

`MeetingId CHAR`	Meeting GUID.

k_ sp_del_ fellow

Delete Employee. Meeting for employee will be erased on cascade.

`FellowId CHAR`	Employee GUID.

k_ sp_del_ room

Delete Room or Shared Resource. If resource is in use by any meeting a foreign key violation exception will be raised.

`RoomNm VARCHAR`	Room Name.
`WorkAreaId VARCHAR`	Workarea GUID.

Product Management Stored Procedures

Can be found at file **products.ddl** under bajo **com/knowgate/hipergate/datamodel/procedures**.

k_ sp_del_ product

Delete Product.

`ProductId CHAR` Product GUID.

☞ Products must be deleted using Java method `delete()` from class `com.knowgate.hipergate.Product` and not calling directly to `k_sp_del_product`. Database stored procedures cannot delete external disk files. If `k_sp_del_product` is executed directly zombie files will be left at hard drive.

Customer Relationship Management Stored Procedures

Can be found at file **crm.ddl** under **com/knowgate/hipergate/datamodel/procedures**.

k_ sp_del_sales_man

Delete Salesman.

`SalesManId CHAR` Salesman GUID.

☞ Salesmen are sub-registers of Domain Users. When a Salesman is deleted Companies assigned to him will be set free of any assignment. Deleting a Salesman does not delete the underlying Domain User. Orders placed by a deleted Salesman are not deleted when he disappears.

k_ sp_del_contact

Delete Contact.

`ContactId CHAR` Contact GUID.

☞ Contacts must be deleted by calling Java method `delete()` from class `com.knowgate.crm.Contact` and not directly calling `k_sp_del_contact` because database stored procedures cannot delete external disk files attached to the Contact.

k_ sp_del_company

Delete Company.

`CompanyId CHAR` Company GUID.

☞ Before deleting a Company it is neccesary to delete its Contacts. This is automatically done by Java class `com.knowgate.crm.Company`

k_ sp_del_oportunity

Delete Opportunity.

`OportunityId CHAR`	Opportunity GUID.

List Management Stored Procedures

Can be found at file **lists.ddl** under **com/knowgate/hipergate/datamodel/procedures**.

k_ sp_del_ list

Remove a list and all its members.

`ListId CHAR`	GUID of the List to be removed.

k_sp_email_blocked

Verifies whether or not an e-mail address is in the black list associated to a distribution list.

`GuList CHAR(32)`	GUID of the distribution list.
`TxEmail VARCHAR(100)`	e-mail to be verified.
`BoBlocked SMALLINT OUT`	1 if `TxEmail` is found in the black list associated to `GuList`, 0 otherwise.

k_sp_contact_blocked

Verifies whether or not a Contact is in the black list associated to a distribution list.

`GuList CHAR(32)`	GUID of the distribution list.
`GuContact CHAR(32)`	GUID of Contact to be verified.
`BoBlocked SMALLINT OUT`	1 if `GuContact` is found in the black list associated to `GuList`, 0 otherwise.

k_sp_company_blocked

Verifies whether or not a Company is in the black list associated to a distribution list.

`GuList CHAR(32)`	GUID of the distribution list.
`GuCompany CHAR(32)`	GUID of Contact to be verified.

`BoBlocked SMALLINT OUT` 1 if `GuCompany` is found in the black list associated to `GuList`, 0 otherwise.

k_sp_del_duplicates

Delete duplicated email addresses from a static or direct list.

ORACLE This procedure is not available for Oracle, use `DELETE FROM k_x_list_members WHERE gu_list=? AND ROWID NOT IN (SELECT MAX(ROWID) FROM k_x_list_members WHERE gu_list=? GROUP BY tx_email` instead.

`ListId CHAR`	Base Static or Direct List GUID.
`Deleted INTEGER OUT`	Count of deleted emails.

List Members Triggers

From version 2.0 the table `k_member_address` contains all addresses of Contacts and Companies. `k_member_address` has the same information as `k_contacts`, `k_companies` and `k_addresses` but in a single table which eliminates the need of JOINs and makes index creation easier.

`k_member_address` is automatically maintained up to date by triggers following the next rules :

1. When an address is added to table `k_addresses` with its `bo_active` set to 1 then the address is also added to `k_member_address`.

2. When an address is updated at `k_addresses` with field `bo_active=1`, a register is updated or added at `k_member_address`. When an address is updated at `k_addresses` with field `bo_active=0`, a register is updated or added at `k_member_address`.

3. When an address is associated to a company at `k_x_company_addr` table, the company information is updated at the corresponding addresses from `k_member_address`.

4. When an address is associated to a contact at `k_x_contact_addr` table, the contact information is updated at the corresponding addresses from `k_member_address`.

5. When an address is deleted from `k_addresses` it is also deleted from `k_member_address`.

6. When a company is deleted from `k_companies`, its `gu_company` field is set to `NULL` at `k_member_address`.

7. When a contact is deleted from `k_contacts`, its `gu_contact` field is set to `NULL` at `k_member_address`.

Cómo reconstruir la vista materializada k_member_address

In PostgreSQL PL/pgSQL function `k_sp_rebuild_member_address` may be used for refilling all the registers of materializad view `k_member_address` with data taken from `k_addresses`, `k_companies` y `k_contacts`.

Forums Stored Procedures

Can be found at file **forums.ddl** under **com/knowgate/hipergate/ datamodel/procedures**.

k_ sp_del_ newsgroup

Removes a Group of messages.

`IdNewsGroup CHAR`	GUID of the Group of Messages to remove.

k_ sp_del_ newsmsg

Removes a Message.

`IdNewsMsg CHAR`	GUID of the Message to remove.

☞ Messages containing attached files must be removed using the `delete()` method in the Java class `com.knowgate.forums.NewsMessage`. Calling `k_sp_del_newsmsg` directly on a message with binary files will dump a foreign key violation on the `k_products` table.

Projects and Bugs Management Stored Procedures

Can be found at file **projtrack.ddl** under **com/knowgate/hipergate/datamodel/procedures.**

k_sp_prj_expand

Expands all child for a project and adds them to the `k_project_expand` table.

`StartWith CHAR`	GUID of the Project to expand.

k_sp_del_project

Removes a Project, including its child Projects, Tasks and Bugs.

`ProjectId CHAR`	GUID of the Project to remove.

k_sp_del_duty

Removes a Task.

`DutyId CHAR`	GUID of the Task to remove.

k_sp_del_bug

Removes a Bug.

`BugId CHAR`	GUID of the Bug to remove.

k_sp_prj_cost

Returns the total cost for a Project, including its child's costs. Total cost is the sum of the cost of Duties plus costs stated explicitly at `k_project_costs` table.

`ProjectId CHAR`	Project's GUID.

Hipermail Stored Procedures

Can be found at file **hipermail.ddl** under **com/knowgate/hipergate/datamodel/procedures**. Also there are a couple of stored procedures for getting user's mail folders at **categories.ddl**.

k_sp_del_mime_msg

Delete a message.

`MimeMsgId CHAR`	GUID of message to be deleted.

☞ Messages must be deleted using Java method `DBMimeMessage.delete()` or calling `DBMimeMessage.setFlag(Flags.Flag.DELETED, true)` and then `DBFolder.expunge()`. Executing `k_sp_del_mime_msg` directly does not remove message source stored at MBOX files. Even if messages are fully stored at LONGVARBINARY columns, there may be references to external files from message drafts.

k_sp_get_mime_msg

Get message GUID from its Id. generated from the mail provider.

`MsgId VARCHAR`	Message Id.
`Guid CHAR OUT`	Message GUID.

k_sp_write_inet_addr

Insert a recipient and add references to User, Contact or Company as the e-mail address matches another already existing one at the same Workarea.

`DomainId INTEGER`	Domain integer Id.
`WorkAreaId CHAR`	Workarea GUID.
`MsgGuid CHAR`	Message GUID.
`MimeMsgId VARCHAR`	Message Id. generated by mail provider.
`RecipientTp VARCHAR`	Recipient Type { `from` | `to` | `cc` | `bcc` }.
`EMailTx VARCHAR`	Recipient's e-mail address.
`PersonalTx NVARCHAR`	Recipient's full name.

4 Java Classes

hipergate has a rich set of libraries that contain functionality and services for building custom extensions on top of the base product.

This section is dedicated to general concepts about Java libraries usage. More information may be found reading the JavaDoc.

Purpose of Java LIBRARIES

When starting to write a web based application it is easy to put everything inside Java Server Pages code.

JSP is easy to write and do not require manual recompiling each time a change is made.

Moreover the page division itself makes easier configuration management and version control than with pre-compiled monolithic libraries.

Unfortunately code written inside JSP is difficult to extend and virtually impossible to reuse. As application business rules grow more complex, JSP becomes more and more cumbersome and tricky to maintain eventually leading to true *spaghetti* code.

At hipergate base functionality is written in Java classes independent from product Java Server Pages. This gives greater modularity, maintainability and performance.

Generic packages vs. packages dependant of the datamodel

Classes of file hipergate.jar are grouped in packages that may be broadly classified in two groups: generic packages and packages dependant of the datamodel. Generic packages can work with any database and do not require an specific underlying table structure. Dependant packages are classes written on purpose for a particular data submodel.

Generic Packages implement services as: debug traces, database persistency, XSLT transformations, FTP file access, etc.

Generic Packages are :

- **`com.knowgate.debug`** : Debug Traces.
- **`com.knowgate.jdc`** : Database Connection Pool.
- **`com.knowgate.dataobjs`** : Database access Objects.
- **`com.knowgate.datacopy`** : Copy complex data structures.
- **`com.knowgate.dataxslt`** : XSLT transformations.

- **com.knowgate.dfs** : FTP and NFS file access.
- **com.knowgate.cache** : RAM memory cache.
- **com.knowgate.misc** : Miscellaneous subroutines.
- **com.knowgate.ole** : POI wrapper for reading OLE2 doc. properties.

Datamodel dependant packages are :

- **com.knowgate.acl** : User Authentification.
- **com.knowgate.addrbook** : Collaborative Tools.
- **com.knowgate.crm** : Sales Force Automation.
- **com.knowgate.dataxslt.db** : XML template indexes.
- **com.knowgate.forums** : Forums.
- **com.knowgate.hipergate** : Product Categorization, Thesauri and Shop
- **com.knowgate.hipergate.datamodel** : Datamodel Launcher
- **com.knowgate.http** : Binary Servlets.
- **com.knowgate.ldap** : Authenticate user using LDAP.
- **com.knowgate.projtrack** : Project Management and Bug Tracking.
- **com.knowgate.scheduler**: Task Scheduler.
- **com.knowgate.workareas**: Workareas.

Additional Packages compiled inside hipergate.jar

- **com.oreilly.servlet** : © 1999-2002 Jason Hunter & Matt Towers.
- **com.enterprisedt.net.ftp** : © 2000-2003 Enterprise Dist. Techs Ltd.
- **org.jical** : © 2002 Stuart Guthrie.
- **dom** : © 1999-2000 The Apache Software Foundation.

Additional Packages compiled outside hipergate.jar

- **org.w3c.tidy** : © 1998-2000 World Wide Web Consortium.
- **com.knowgate.jcifs** : © 2000 Michael B. Allen.

Debug Traces

Each hipergate.jar distribution is published in two different flavours: *debug* and *release*.

The debug version leaves detailed execution traces at /tmp/javatrc.txt (UNIX) or C:\javatrc.txt (Windows).

The static variable `trace` from class `com.knowgate.debug.DebugFile` controls whether debug traces must be generated. This is a final variable and cannot be changed at runtime.

Querying debug mode from the command line

You can know from the command line whether or not a debug version is installed. Type: `java com.knowgate.debug.DebugFile`. Either "Debug mode enabled" or "Debug mode disabled" will appear on the console.

Initialization Properties

Initialization properties are set at `hipergate.cnf`.

For accessing these properties class `com.knowgate.misc.Environment` is used.

By default `hipergate.cnf` must be at /etc (UNIX) or C:\WINNT (Windows). For placing properties file `hipergate.cnf` somewhere else, the operating system environment variable `KNOWGATE_PROFILES` must be set pointing to the directory containing `hipergate.cnf`.

For example :

`KNOWGATE_PROFILES=/opt/knowgate/`
(UNIX)
or
`SET KNOWGATE_PROFILES=C:\\knowgate\\`
(Windows)

One properties are read for the first time they are cached in memory by class `Environment` until method `Environment.refresh()` is called.

Initial loading of tables and data

The package `com.knowgate.hipergate.datamodel` contains the neccesary elements for creating the hipergate datamodel from scratch.

Class `ModelManager` can be used for creating and dropping the whole or parts of the datamodel.

All SQL senetences for loading the datamodel are contained as resources in subfolders of package `com.knowgate.hipergate.datamodel` inside `hipergate.jar`.

`ModelManager` can be invoked from the command line with :

```
java com.knowgate.hipergate.datamodel.ModelManager path
command module [verbose]
```

or

```
java com.knowgate.hipergate.datamodel.ModelManager path
command [domain|workarea] [domain_name| domain_name.
workarea_name] [verbose]
```

or

```
java com.knowgate.hipergate.datamodel.ModelManager path
clone workarea origin_domain_name.origin_workarea
target_domain_name. target_workarea [verbose]
```

☞ Command `com.knowgate.hipergate.datamodel.ModelManager` *`path`* `create all` may take several minutes to complete execution.

Where :

`path` : Full path `hipergate.cnf` file (or equivalent) that contain properties `driver`, `dburl`, `dbuser` y `dbpassword` for connecting to the database.

`command` : Must be `create` or `drop` or `clone` depending on whether creating or droping the domain or module coming next. Or `clone` for cloning a Workarea.

`module` : Name of module to be created or droped. Must be one of the following values : `all`, `kernel`, `lookups`, `security`, `jobs`, `thesauri`, `categories`, `products`, `teamwork`, `webbuilder`, `crm`, `lists`, `shop`, `projtrack`, `billing`.
If `all` is specified all modules will be created or droped.

☞ Creating all modules is different from creating the whole default database. When modules are created TEST, DEMO and REAL Workareas are not created as they are in database creqation. So, in fact, module creation is a subset of database creation.

☞ Modules have dependencies and foreign keys among them. If they are individually created or destroyed, it must be done in the order specified in the above list.

`domain` : Name of domain to be created or eliminated. The domain numeric identifier is automatically assigned and cannot be chosen by user.

`domain.workarea` : Name of domain *dot* Name of Workarea.

`verbose` : Print to standard output SQL sentences as they are executed.

Example 1, create complete datamodel with no data:

```
java com.knowgate.hipergate.datamodel.ModelManager
/opt/knowgate/hipergate.cnf create all verbose
```

Example 2, drop project management module:

```
java com.knowgate.hipergate.datamodel.ModelManager
/opt/knowgate/hipergate.cnf drop projtrack verbose
```

Example 3, Create a Domain:

```
java com.knowgate.hipergate.datamodel.ModelManager
/opt/knowgate/hipergate.cnf create domain YOURDOMAIN
```

Example 4, clone Workarea from domain MODEL to domain TEST1:

```
java com.knowgate.hipergate.datamodel.ModelManager
/opt/knowgate/hipergate.cnf clone workarea
MODEL.model_default TEST1.test1_workarea
```

Database Connection Pool

Package `com.knowgate.jdc` implements a databse connection pool and a wrapper for class `java.sql.Connection`.

It is recommended to use always the connection pool, not only because it improves performance but because it allows a centraliced management of open connections and detect more easily connection leaks.

It is necessary to pay attention to how transactions are commited or rolled-back on each page. Since a connection is not actually closed when calling `JDCConnection.close()` but just returned to the pool, a page may leave pending uncommited operations that block the next page. For avoiding this problem it is recommended either:

a) Use `Connection.setAutoCommit(true)` before performing any operation that writes anything into the database or

b) Use `Connection.setAutoCommit(false)` before performing any operation that writes into the database and then always do either y `Connection.commit()` or `Connection.rollback()` before exiting the page.

Maping of Java objects to database registers

There are a lot of object-relational bridges available at the market.

hipergate uses a propietary package (`com.knowgate.dataobjs`) for this task.

`dataobjs` is a database persistence package designed to be simple, fast and easy to use with the trade of some design restrictions :

- A persistent object can only be composed of simple properties that can be directly mapped to single database fields.
- Each object instance must correspond to a unique register at a given database table.
- Persistent object cannot contain references to other objects.
- Properties of persisted object must be called exactly equal as the database field where they are stored.
- Each object has one or several properties that act as object primary key.

With this approach it is easy to map Java object to database fields.

For example:

DDL

```
CREATE TABLE my_object_table (
  id_object  CHAR   (9),
  tx_object  VARCHAR(255),
  CONSTRAINT pk_object_table PRIMARY KEY (id_object)
)
```

Java

```
import com.knowgate.dataobjs.*;

public class MyObject extends DBPersist {

  public MyObject () {
    super ("my_object_table", "MyObject");
  }

}
```

Now, properties assigned to instances of class MyObject will be saved to the databse when `store()` method is called.

Just do :

```
com.knowgate.dataobjs.DBBind oDBB;
com.knowgate.jdc.JDCConnection oCon;
MyObject oObj;

// Load metadata taking connection from hipergate.cnf

oDBB = new DBBind();

try {
  // Get connection from pool set its name to “my_object_test”

  oCon = oDBB.getConnection("my_object_test");

  // Instantiate object inherited from DBPersist

  oObj = new MyObject();

  // Set properties called as database columns

  oObj.put ("id_object", "A12345678");
  oObj.put ("tx_object", "my first hipergate object");


  // Save register to my_object_table
  // It is not neccesary to worry about whether register
  // previously existed at database or not.
  // DBPersist handles insert or update automatically.

  oObj.store(oCon);
```

```
  oObj = null;
}
catch (SQLException) {
  /* ... Capturar la excepción */
}

oObj = new MyObject();

// Load object from database

boolean bLoaded = oObj.load (oCon, new Object[]{"A12345678"});

System.out.println("Texto: " + oObj.getString("tx_object"));

// Return connection to pool.

oCon.close("my_object_test");

oCon = null;
```

☞ `load()` and `store()` can only work if the table has a primary key constraint physically defined at the datamodel.

Table and Column names

Table and column names are centralized at `com.knowgate.dataobjs.DB`. It is possible to change table and column names by simply changing them at DB final static variables. But this only applies to the library itself as column names are also hardwired at HTML forms from the front-end.

DBMS metadata RAM cache

It is possible to connect the connection pool `com.knowgate.jdc.JDCConnectionPool` directly to the database. But the usual case is to use a `com.knowgate.dataobjs.DBBind` object that contains the pool inside. Database connection information is read from `hipergate.cnf` properties file.

Connections are typically obtained by calling `getConnection(…)` method of `DBBind`.

`DBBind` adds an additional layer that keeps database metadata information cached in RAM.

Metadata is used by `DBPersist` inherited classes for mapping Java variables to database columns.

Transactions

Transaction management is a responsibility of the client application, not of the Java library itself.

Transactions are managed by JSP pages using JDBC calls `Connection.setAutoCommit(false)` and then `Connection.commit()` or `Connection.rollback()`.

Auditing

Auditing is responsibility of client application.

hipergate has the class `DBAudit` and the table `k_auditing` for operation auditing purposes; but it is not recommended to use the same database both for OLTP and audit log purposes.

It is best to save audit information to a plain text file and later process it with a data mining tool.

Long fields management

Long fields may be written as plain Strings (for LONGVARCHAR y CLOB) or be associate to a disk file.

DDL

```
CREATE TABLE my_long_table (
  id_long  CHAR(9) ,
  tx_long  LONGVARCHAR,
  CONSTRAINT pk_long_table PRIMARY KEY (id_long)
)
```

Java

```
import com.knowgate.dataobjs.*;

public class MyLongObject extends DBPersist {
```

```
  public MyObject () {
    super ("my_long_table", "MyLongObject");
  }
```

_____

```
// Example of how to save a long field from a file

com.knowgate.dataobjs.DBBind oDBB;
com.knowgate.jdc.JDCConnection oCon;

oDBB = new DBBind();

oCon = oDBB.getConnection("my_long_object_test");

MyLongObject oObj = new MyLongObject ();

oObj.put ("id_long", "L12345678");
oObj.put ("tx_object", new File("/tmp/uploadme.txt"));
oObj.store (oCon);

oCon.close("my_long_object_test");

oCon = null;
```

_____

```
// Example of how to save a long field from an array

com.knowgate.dataobjs.DBBind oDBB;
com.knowgate.jdc.JDCConnection oCon;

oDBB = new DBBind();
oCon = oDBB.getConnection("my_long_object_test");
MyLongObject oObj = new MyLongObject ();
oObj.put ("id_long", "B12345678");
oObj.put ("tx_object", new char[]{'A','B','C','D'});
oObj.store (oCon);

oCon.close("my_long_object_test");
oCon = null;
```

XML data loading

`DBPersist` class has method `parseXML()` for loading fields from an XML file into a `DBPersist` object.

File C:\knowgate\UserXML.txt_____

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ACLUser>
<gu_user>32f4f56fda343a5898c15a021203dd82</gu_user>
<id_domain>1026</id_domain>
<nm_user>The 7th Guest</nm_user>
<tx_pwd>123456</tx_pwd>
```

```
<tx_main_email>guest7@domain.com</tx_main_email>
<tx_alt_email>admin@hipergate.com</tx_alt_email>
<dt_last_updated>Fri, 29 Aug 2003 13:30:00
GMT+0130</dt_last_updated>
<tx_comments><![CDATA[Sôme ñasti & ïnternational chars
stuff]]></tx_comments>
</ACLUser>
```

Java

```
import com.knowgate.acl.ACLUser;

ACLUser oUsr = new ACLUser();

oUsr.parseXML("file://C:\\knowgate\\UserXML.txt");

System.out.println("nm_user is " + oUsr.getString("nm_user"));
```

Only nodes named as database columns are loaded from XML file into `DBPersist`, the rest are ignored.

The load process will try to convert each XML element to the internal type of its corresponding column. So if dates or numbers are bad formatted, an exception will be raised during parsing.

Burst reads

The general policy of hipergate JSP pages is to minimize the time each database connection is in use. For each page, a connection is opened before sending HTML to the client, data is read and connection is returned to pool.

The main class used for quick and atomic database reads is `com.knowgate.dataobjs.DBSubset`.

`DBSubset` is a bi-dimensional array held in memory that contains a subset of registers from a database table read all at once.

```
import com.knowgate.jdc.JDCConnection;
import com.knowgate.acl.ACLUser;
import com.knowgate.dataobjs.*;

DBBind oDBB = new DBBind();

JDCConnection oCon = oDBB.getConnection("dbsubset_test");

// Example of parameterless DBSubset
```

```
DBSubset oCur = new DBSubset ("k_lu_currencies",
"alpha_code,tr_currency_en", "numeric_code NOT LIKE '9%'", 250);

int iCurrencyCount = oCur.load (oCon);

for (int c=0; c<iCurrencyCount; c++) {

   System.out.println(oCur.getString("alpha_code") + " " +
   oCur.getStringNull("tr_currency_en","no_translated_name"));

}

oCur = null;

// Example of DBSubset with parameters

DBSubset oCur = new DBSubset ("k_lu_currencies",
"alpha_code,tr_currency_en", "numeric_code NOT LIKE ? AND
char_code<>?", 250);

int iCurrencyCount = oCur.load (oCon, new Object[]{"9%","¤"});

for (int c=0; c<iCurrencyCount; c++) {

   System.out.println(oCur.getString("alpha_code") + " " +
   oCur.getStringNull("tr_currency_en","no_translated_name"));

}

oCur = null;


// Example with a JOIN of two tables with limited number of
results to 100

DBSubset oCur = new DBSubset ("k_users u, k_domains d",
"u.nm_user, d.nm_domain", "u.id_domain=d.id_domain", 100);


// The last parameter of constructor is NOT the number of
registers to be readed, but only a predistive indicator of how
many rows will be readed at a time. The maximum number of rows is
limited by calling setMaxRows().

oCur.setMaxRows(100);

int iUserCount = oCur.load (oCon);

oCur = null;

oCon.close("dbsubset_test ");

oCon = null;
```

☞ If MaxRows is not set, `DBSubset` will load all available files from the database.

Using class ImportExport for loading delimited text files

Class `com.knowgate.hipergate.datamodel.ImportExport` takes as input a delimited text file and a descriptor for that file and using both it loads data from the text file into one or more database tables.

`ImportExport` works by creating instances that implement `com.knowgate.hipergate.datamodel.ImportLoader` interface. The exact class to be instantiated depends on the instructions given at the file descriptor.

`ImportExport` has a single method called `perform()` which receives the file descriptor as parameter. The general syntax for the descriptor is:

```
[APPEND|UPDATE|APPENDUPDATE]
[CONTACTS|COMPANIES|USERS|PRODUCTS|FELLOWS] CONNECT user TO
"jdbc connection string" IDENTIFIED BY password SCHEMA
"schemaname"  WORKAREA "workarea name" INPUTFILE
"/tmp/filename.txt" CHARSET [ASCII|ISO8859_1| UTF8|…]
ROWDELIM [CR|LF|CRLF|caracter] COLDELIM [TAB|caracter]
BADFILE "/tmp/badfile.txt" DISCARDFILE "/tmp/badfile.txt"
[RECOVERABLE] [PRESERVESPACE] (column definition, column
definition, …)

column definition:= column name [CHAR | VARCHAR | DATE "date
format"| SMALLINT | INTEGER | FLOAT | DOUBLE | NUMERIC]
```

Date format must be one of the accepted by `SimpleDateFormat`.

Explanation of each reserved word:

APPEND, **UPDATE** or **APPENDUPDATE**: Data insertion mode. If it is APPEND then registers that do not exist at the database will be inserted, the ones that already exist will raise a duplicated primary key exception.

CONTACTS, COMPANIES, USERS, FELLOWS or **PRODUCTS**: Name of high-level entity that is being loaded. Currently it must be CONTACTS, COMPANIES, USERS, FELLOWS or PRODUCTS. Each of which will instantiate:

- `com.knowgate.crm.ContactLoader` for writing into tables `k_companies`, `k_contacts` and `k_addresses`.
- `com.knowgate.crm.CompanyLoader` for writing into tables `k_companies` and `k_addresses`.
- `com.knowgate.acl.UserLoader` for loading table `k_users`.
- `com.knowgate.hipergate.FellowLoader` for loading `k_users` and `k_fellows`

- `com.knowgate.hipergate.ProductLoader` for loading `k_products`.

Companies are matched by the value of column nm_legal, if the input file contains a company with the same value in k_companies. Nm_legal as a previously existing one at the database then they are supposed to be the same company.

Individuals are matched by sn_passport.

Addresses are matched by ix_address.

Users and fellows are matched by domain+nickname.

Products are matched by gu_product.

CONNECT ***user*** **TO** ***jdbc connection string*** **IDENTIFIED BY** ***password*** **SCHEMA** ***schemaname***: Complete specification of the database to connect to.

WORKAREA ***"workarea"***: Name or GUID of the Workarea where the input data will be written.

CATEGORY ***"category name"***: Name (`k_categories.nm_category`) or GUID of category where products will be inserted. This parameter is only allowed if loading products.

INPUTFILE ***"path to input file"***: Full path to input file.

BADFILE ***"path to outputfile"***: Full path to file where errors that occur during data processing will be logged.

DISCARDFILE ***"path to outputfile"***: Full path to file where lines that could not be inserted World be written.

CHARSET ***charset encoding***: Character encoding for inputfile. Must be one of the Java supported encodings listed at: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html.

ROWDELIM ***delimiter***: Row delimiter. May be a quote character or one of the keywords CR, LF or CRLF.

COLDELIM ***delimiter***: Column delimiter. May be a quote character or the keyword TAB.

RECOVERABLE: Signals that the whole data loading must be done within the same atomic transaction. Ether all or none of the input files are loaded. This option may overflow the rollback segments of the DBMS if the data set is too large.

UNRECOVERABLE: Inserted data Hill be committed alter each row. This is the default value

PRESERVESPACE: Signals that blank space to the right must be preserved..

INSERTLOOKUPS: Signals that lookup tables must be checked and updates if new values not already present are supplied at the input file.

WITHOUT DUPLICATED [NAMES|EMAILS]: This clause is only allowed when loading contacts. It avoids that contacts are loaded with duplicated name+surname or duplicated e-mail.

Contacts loading example:

File descriptor

```
APPEND CONTACTS
CONNECT knowgate TO
"jdbc:postgresql://192.168.1.10:10801/hgoltp8t"
IDENTIFIED BY knowgate WORKAREA test_default
INPUTFILE "C:\\Temp\\Contacts.txt" CHARSET ISO8859_1
ROWDELIM CRLF COLDELIM "|"
BADFILE "C:\\Temp\\Contacts_bad.txt"
DISCARDFILE "C:\\Temp\\Contacts_discard.txt"
(nm_legal VARCHAR,
id_company_ref VARCHAR,
tx_name VARCHAR,
tx_surname VARCHAR,
sn_passport VARCHAR,
nm_street VARCHAR,
nu_street VARCHAR,
zipcode VARCHAR,
mn_city VARCHAR,
work_phone VARCHAR)
```

Input File

```
FOAL LTD.|1.10000018|PERE|FOCHS ALVAREZ|B60614559|BALLESTER|5|08023|BARCELONA|956300022
FOAL LTD.|1.10000018|JORDI|SETIEN LLUC|B60614559|BALLESTER|5|08023|BARCELONA|556300669
FOAL LTD.|1.10000018|MERITXELL|VIDAL RUIZ|B60614559|BALLESTER|5|08023|BARCELONA|557893841
BETIS CORP.|1.10000060|IGNACIO|SANCHEZ MEJIAS|28452380T|ITALIA|7|41012|SEVILLA|556300065
MADRID DEVEL|1.10000204|ALDO|MARQUEZ SANTANA|02523827Z|CORTINA|1|28010|MADRID|556300210
INCHAURISA SL|1.10000326|ANTONIO|LOPEZ RIGUE|B82612656|LECHUGA|7|45600|TALAVERA |656305084
INCHAURISA SL|1.10000326|JESUS|PEREZ PEREZ|B82612656|LECHUGA|7|45600|TALAVERA |559507033
PONCE LTD.|1.10001052|CARMELO|MARTIN PONCE|22662241L|SAN MARCEL|1|46017|VALENCIA|525482155
PONCE LTD.|1.10001052|SUSANA|MARTIN LEON|22662241L|SAN MARCEL|1|46017|VALENCIA|556304004
MARTIN LTD.|1.10001062|ISAIAS|SASTRE MARTIN|03450263X|HOYA|14|40003|SEGOVIA|656304013
HERO CORP.|1.10001079|ELISEO|VIDAL PEREZ|34979130V|JOSE DE ARO|57|46022|VALENCIA|656304031
PRODUCCIONES SA|1.10001092|MARIA|SAN ROMAN|A78473584|CLAVEL|17|28250|TORRELODONES|456304045
PRODUCCIONES SA|1.10001092|ANA|ESTEVIL GIL|A78473584|CLAVEL|17|28250|TORRELODONES|985468979
GENIZAROS CORP.|1.10001939|PEDRO|GENIZ CANO|28734030S|ALTA|10|41980|LA ALGABA|856303506
GENIZAROS CORP.|1.10001939|JOSE|GENIZ CANO|28734030S|ALTA|10|41980|LA ALGABA|865950333
MORAMORA LTD.|1.10002035|JESUS|MORALES MORA|25337120K| ORTIZ|17|29300|ARCHIDONA|615473089
MORAMORA LTD.|1.10002035|ANDRES|LOZANO GIL|25337120K|ORTIZ|17|29300|ARCHIDONA|856303603
MORAMORA LTD.|1.10002035|FEDERICO|UMANO|25337120K|ORTIZ|17|29300|ARCHIDONA|857810220
```

Java Code:

```
ImportExport  oImpExp = new ImportExport();
oImpExp.perform("APPENDUPDATE CONTACTS CONNECT knowgate TO
\"jdbc:postgresql://192.168.1.10:10801/hgoltp8t\" IDENTIFIED
BY knowgate WORKAREA test_default INPUTFILE
\"C:\\\\Temp\\\\Contacts.txt\" CHARSET ISO8859_1 ROWDELIM
CRLF COLDELIM \"|\" BADFILE
\"C:\\\\Temp\\\\Contacts_bad.txt\" DISCARDFILE
\"C:\\\\Temp\\\\Contacts_discard.txt\" (nm_legal VARCHAR,
id_company_ref VARCHAR, tx_name VARCHAR, tx_surname VARCHAR,
sn_passport VARCHAR, nm_street VARCHAR, nu_street VARCHAR,
zipcode VARCHAR, mn_city VARCHAR, work_phone VARCHAR)");
```

Users loading example :

Java Code:

```
ImportExport  oImpExp = new ImportExport();
oImp.perform("APPENDUPDATE USERS CONNECT user_name TO \"
jdbc:oracle:thin:@192.168.1.24:1521:orcl\" IDENTIFIED BY
password WORKAREA test_default INPUTFILE
\"C:\\\\usuarios.txt\" CHARSET ISO8859_1 ROWDELIM CRLF
COLDELIM \";\" BADFILE \"C:\\\\Users_bad.txt\" DISCARDFILE
\"C:\\\\Users_discard.txt\" (id_domain INTEGER, nm_acl_group
VARCHAR, tx_pwd VARCHAR, tx_nickname VARCHAR, ignore NULL,
tx_main_email VARCHAR, nm_user VARCHAR, tx_surname1 VARCHAR,
de_title VARCHAR)");
```

Input File :

```
2050;TEST / Users;187987;omestre;ignore;oscar.mestre@hg.com;OSCAR;MESTRE;Delegado
2050;TEST / Users;140551;fbaca;ignore;paco.bacardit@hg.com;PACO;BACARDIT;Jefe de Área
2050;TEST / Users;175910;jfdez;ignore;jesus.fdez-jaso@hg.com;JESUS;FERNANDEZ;Delegado
2050;TEST / Users;176110;jmartin;ignore;jesus.martin@hg.com;JESUS;MARTIN;Delegado
2050;TEST / Users;205557;rblanco;ignore;Ruben.Blanco@hg.com;RUBEN;BLANCO;Delegado
2050;TEST / Users;204620;pibox;ignore;pi.boxaderas@hg.com;PEDRO I.;BOXADERAS;Delegado
```

```
2050;TEST / Users;205715;jfuset;ignore;Jordi.Fuset@hg.com;JORDI FUSET;Delegado;4
2050;TEST / Users;203212;ahernandez;ignore;a.herdez@hg.com;ALEX;HERNANDEZ;Delegado
2050;TEST / Users;144674;ejimenez;ignore;emilio.jimenez@hg.com;EMILIO;JIMENEZ;Delegado
2050;TEST / Users;203341;soller;ignore;sergio.oller@hg.com;SERGIO;OLLER;Delegado
2050;TEST / Users;200997;esanchez;ignore;e.sanchez@hg.com;ENRIQUE;SANCHEZ;Delegado
2050;TEST / Users;191847;psoriano;ignore;psoriano@hg.com;PEDRO;SORIANO;Delegado
2050;TEST / Users;185862;pufano;ignore;pedro.ufano@hg.com;PEDRO;UFANO;Delegado
2050;TEST / Users;205727;jverdugo;ignore;Jordi.VERDUGO@hg.com;JORDI;VERDUGO;Delegado
2050;TEST / Users;193015;jviles;ignore;juan.viles@hg.com;JUAN;VILES;Delegado;4
```

Products loading example:

Input File :

```
The Soul Of A New Machine|316491977|For a time after the first pieces of Route 495
were laid down across central Massachusetts, in the middle 1960s, the main hazard to
drivers…|STD|14.95|9.72|840|0|1|01/06/2001|2|HTML|Tracy
Kidder|1|316491977|320|http://www.amazon.com/gp/product/0316491977/qid=1146703168/sr=2
-1/ref=pd_bbs_b_2_1/002-4649625-8263234?s=books&v=glance&n=283155
The Mythical Man-Month|201835959|The classic book on the human elements of software
engineering.|STD|34.99|34.99|840|0|1|02/08/1995|2|HTML|Frederick P.
Brooks|1|201835959|322|http://www.amazon.com/gp/product/0201835959/qid=1146703814/sr=2
-1/ref=pd_bbs_b_2_1/002-4649625-8263234?s=books&v=glance&n=283155
Peopleware|932633439|Productive Projects and
Teams|STD|33.95|33.95|840|0|1|01/02/1999|2|HTML|Tom
DeMarco|1|932633439|245|http://www.amazon.com/gp/product/0932633439/qid=1146703965/sr=
2-1/ref=pd_bbs_b_2_1/002-4649625-8263234?s=books&v=glance&n=283155
Death March|013143635X|The #1 guide to surviving doomed
projects||34.99|22.04|840|0|1|07/12/2003|2|HTML|Edward
Yourdon|1|013143635X|304|http://www.amazon.com/gp/product/013143635X/qid=1146704097/sr
=2-1/ref=pd_bbs_b_2_1/002-4649625-8263234?s=books&v=glance&n=283155
```

File Descriptor :

```
APPENDUPDATE PRODUCTS CONNECT knowgate TO
"jdbc:postgresql://localhost:5432/hipergate" IDENTIFIED BY
knowgate WORKAREA test_default CATEGORY BOOKS~00001
INPUTFILE "C:\\Temp\\Products.txt" CHARSET ISO8859_1
ROWDELIM CRLF COLDELIM "|" BADFILE
"C:\\Temp\\Contacts_bad.txt" DISCARDFILE
"C:\\Temp\\Contacts_discard.txt" (nm_product VARCHAR,id_ref
VARCHAR,de_product VARCHAR,id_fare VARCHAR,pr_list
DECIMAL,pr_sale DECIMAL,id_currency VARCHAR,pct_tax_rate
FLOAT,is_tax_included SMALLINT,dt_acknowledge DATE
DD/MM/yyyy,id_cont_type INTEGER,id_prod_type VARCHAR,author
VARCHAR,days_to_deliver SMALLINT,isbn VARCHAR,pages
INTEGER,url_addr VARCHAR)
```

☞ `BOOKS~00001` is the unique name of the category where products must be inserted as set at column `nm_category` of table `k_categories`. The target category GUID may also be used instead of the name for the **CATEGORY** parameter value.

Java Code :

```
ImportExport  oImpExp = new ImportExport();
```

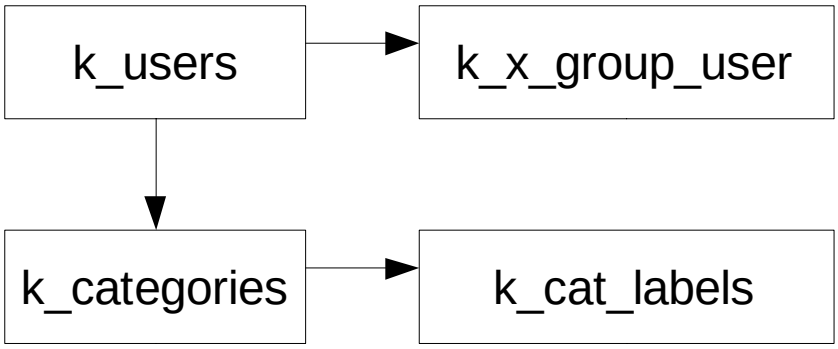
```
oImp.perform("APPENDUPDATE PRODUCTS CONNECT user_name TO
\"jdbc:postgresql://127.0.0.1:5432/hipergate\" IDENTIFIED BY
knowgate WORKAREA test_default CATEGORY BOOKS~00001
INPUTFILE \"C:\\\\Temp\\\\Products.txt\" CHARSET ISO8859_1
ROWDELIM CRLF COLDELIM \"|\" BADFILE
\"C:\\\\Temp\\\\Contacts_bad.txt\" DISCARDFILE
\"C:\\\\Temp\\\\Contacts_discard.txt\" (nm_product
VARCHAR,id_ref VARCHAR,de_product VARCHAR,id_fare
VARCHAR,pr_list DECIMAL,pr_sale DECIMAL,id_currency
VARCHAR,pct_tax_rate FLOAT,is_tax_included
SMALLINT,dt_acknowledge DATE DD/MM/yyyy,id_cont_type
INTEGER,id_prod_type VARCHAR,author VARCHAR,days_to_deliver
SMALLINT,isbn VARCHAR,pages INTEGER,url_addr VARCHAR)");
```

How the user and employee loader works

Users and employees are two tightly coupled entities at hipergate data model.

Users are loaded at table `k_users` and their group membership is established at `k_x_group_user`. Also, the class `com.knowgate.acl.UserLoader` (which is the one that actually performs user loading) makes use of the script `com/knowgate/hipergate/datamodel/scripts/user_categories_create.js` through class `com.knowgate.hipergate.datamodel.ModelManager` for the creation of default categories for each user needed by the Virtual Library and the WebMail.
When update mode is turned on, two users are considered to be the same if they have the same domain and nickname.

Map of tables written when loading a user from a plain text file.



Input file:

```
TEST / Users|paul|12345|S|paul.kless@hipergate.com|Paul|Klee||Abstract Paintings Inc.
TEST / Users|pablo|5552|S|pablo.picasso@hipergate.com|Pablo|Ruíz|Picasso|Cubist Print
TEST / Users|botero|98765|S|fer.botero@hipergate.com|Fernando|Botero||Medellín Draw
```

Java Code:

```
ImportExport  oImpExp = new ImportExport();
oImp.perform("APPEND USERS CONNECT knowgate TO \"
jdbc:mysql://127.0.0.1/hipergate\" IDENTIFIED BY knowgate
WORKAREA test_default INPUTFILE \"/tmp/Users.txt\" CHARSET
ISO8859_1 ROWDELIM CRLF COLDELIM \"|\" BADFILE
\"/tmp/Users_bad.txt\" DISCARDFILE
\"/tmp/Users_discard.txt\" (nm_acl_group VARCHAR,
tx_nickname VARCHAR,tx_pwd VARCHAR,tp_account
VARCHAR,tx_main_email VARCHAR,nm_user VARCHAR,tx_surname1
VARCHAR, tx_surname2 VARCHAR, nm_company VARCHAR)");
```

The first parameter of the input file is the name of the permissions group at table `k_acl_groups` to which the new user is to be made member, it is also possible to set the group GUID directly. When loading from a text file, it is only possible to specify one group per user.

The default Workarea for all users being loaded can be set by using the reserved keyword `WORKAREA` or a different default Workarea can be set for each user independently by specifying either its name (`nm_workarea`) or its GUID (`gu_workarea`)

The numeric identifier for the domain (`id_domain`) can be inferred from the Workarea, or explicitly set, in which case it must be consistent with the Workarea that is also given.

Employees are loaded at table `k_fellows` and its lookup tables `k_fellows_lookup` and `k_lu_fellow_titles`.

Although it is not required for preserving any referential integrity at the data model, for each employee loaded a corresponding user is also created. This is done automatically by `com.knowgate.addrbook.FellowLoader`

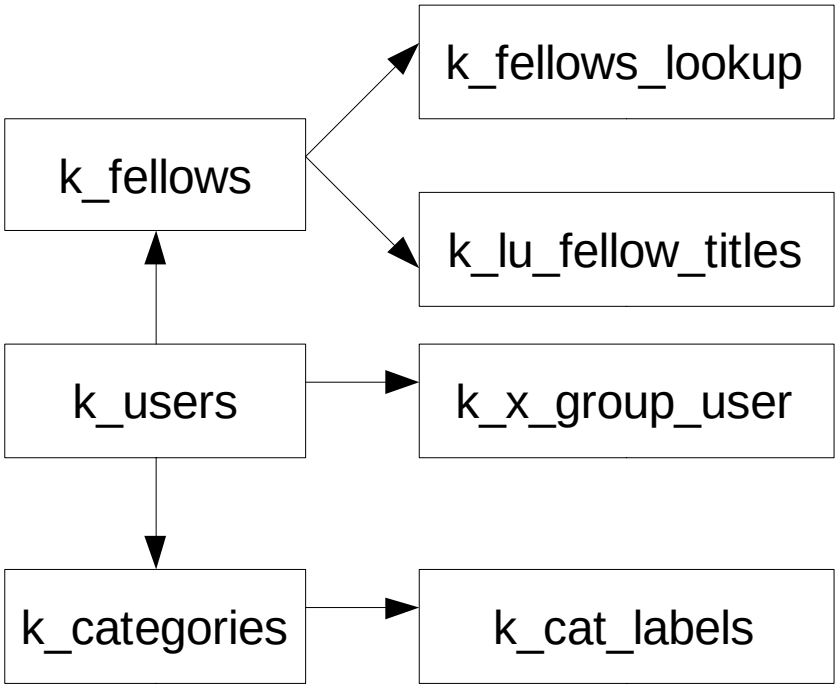
Some columns at `k_fellows` table are redundant with others at `k_users` and, although distinct values may be loaded for these redundant columns, it is recommended to keep them synchronized this way:

k_fellows	**k_users**
tx_name	nm_user
tx_surname	tx_surname1+" "+tx_surname2
tx_company	nm_company
tx_email	tx_main_email

If one the previous pairs is not explicitly specified then it takes the value from the other. This means that if you put a column named `tx_email` at the input file but not a `tx_main_email` column, then both `tx_email` and `tx_main_email` are stored at the database with the same value.

Tables `k_fellows_lookup` and `k_lu_fellow_titles` Hill only be written if modifier `INSERTLOOKUPS` is set.

Map of tables written when loading a fellow from a plain text file.



Input file:

```
TEST / Users|mark|12345|S|mark@hipergate.com|Mark|Kolomer||ACME|SALES|MANAGER
TEST / Users|lua|5552|S|lua@hipergate.com|Lua|Mel|Yi|ACME|SALES|ASSISTANT
TEST / Users|anna|98765|S|anna@hipergate.com|Anna|Nova||ACME|ACCOUNTING|CONTROLLER
```

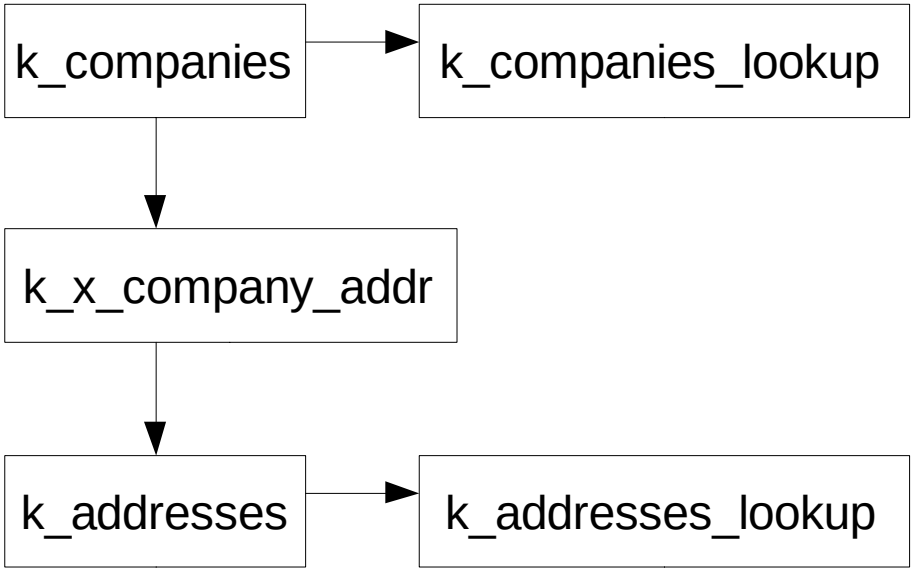
Java Code:

```
ImportExport  oImpExp = new ImportExport();
oImp.perform("APPEND FELLOWS CONNECT knowgate TO \"
jdbc:mysql://127.0.0.1/hipergate\" IDENTIFIED BY knowgate
WORKAREA test_default INSERTLOOKUPS INPUTFILE
\"/tmp/Fellows.txt\" CHARSET ISO8859_1 ROWDELIM CRLF
COLDELIM \"|\" BADFILE \"/tmp/Fellows_bad.txt\" DISCARDFILE
\"/tmp/Fellows_discard.txt\" (nm_acl_group VARCHAR,
tx_nickname VARCHAR,tx_pwd VARCHAR,tp_account
VARCHAR,tx_main_email VARCHAR,nm_user VARCHAR,tx_surname1
VARCHAR, tx_surname2 VARCHAR, nm_company VARCHAR, tx_dept
VARCHAR, de_title VARCHAR)");
```

How the company loader works

Companies are loaded by using class `com.knowgate.crm.CompanyLoader`. That class writes into `k_companies` and, depending on the values of `WRITE_LOOKUPS` and `WRITE_ADDRESSES` passed to the `store()` method it also writes to { `k_companies_lookup` , `k_addresses_lookup` } and { `k_addreses`, `k_x_company_addr` }
If the call for loading companies is made from `perform()` method from class `ImportExport`, then the default behaviour is not to load any lookups (unless the modifier `INSERTLOOKUPS` is set) and do load addresses.

Map of tables written when loading a company from a plain text file.



☞ It is not allowed that the company legal name (column `nm_legal` of table `k_companies`) is duplicated under the same Workarea.

How the contact loader works

Contacts are loaded by classes `com.knowgate.crm.ContactLoader`. That class writes to `k_contacts` table. If at method `store()` flag `WRITE_COMPANIES` is set, then `k_companies` table is also written. And if flag `WRITE_ADDRESSES` is set, then `k_addresses` and `k_x_contact_addr` tables are also written. The default behaviour from

`perform()` method of class `ImportExport` is to load both companies and addresses when loading a contact, but lookup values are not loaded unless `INSERTLOOKUPS` modifier is set.

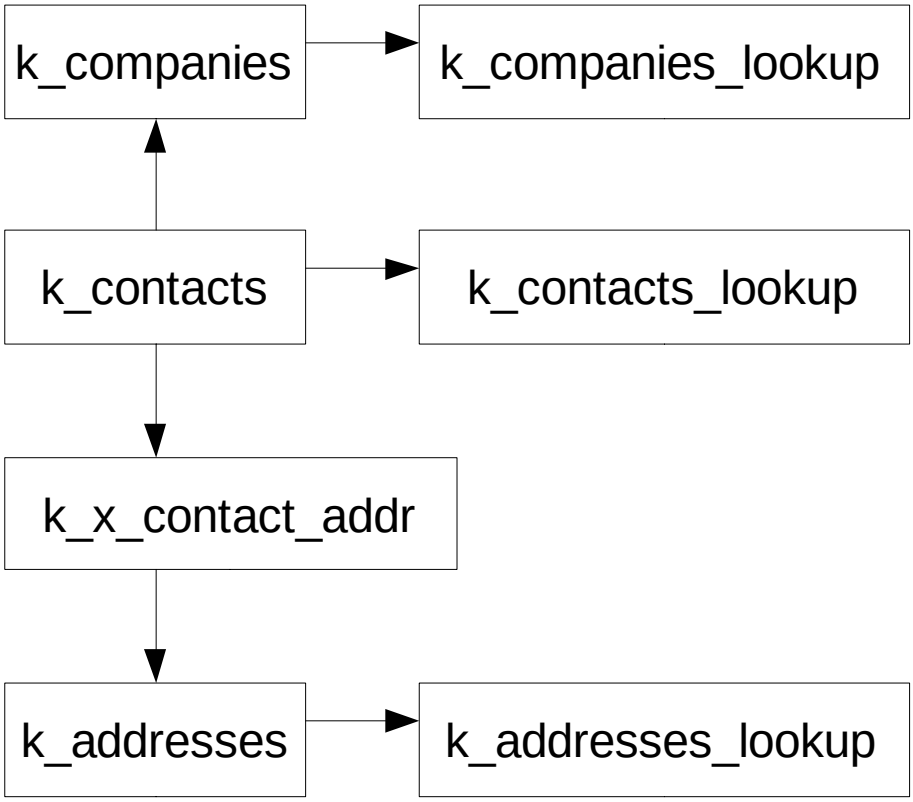
When update mode is used, two contacts are considered to be the same if they have the same `sn_passport`.

To avoid duplicated contacts there are two methods that can be passed as parameters to `store()` method:

- `NO_DUPLICATED_NAMES` : Avoids name plus surname duplicates (`tx_name+tx_surname` en `k_contacts`).
- `NO_DUPLICATED_MAILS` : Avoid e-mail duplicates.

From method `perform()` of class `ImportExport` the clause for avoiding duplicates are `WITHOUT DUPLICATED NAMES` or `WITHOUT DUPLICATED EMAILS`.

Map of tables written when loading a contact from a plain text file.



How the product loader works

Loading products into hipergate catalog is a rather complex task. Product data is kept at seven tables :

1. **k_products**: Holds basic product information: name, standard price, etc.
2. **k_addresses**: This is a sub-record from k_products. A product may have an optional associated address.
3. **k_prod_fares**: Since v3.0 a product may have several fares (different sale process for different customers). Each fare is identified by its name and there may be an unlimited number of fares per product.
4. **k_prod_locats**: A product may be present at different locations. If it is a physical good it may be stocked at several warehouses. If it is a file it may be downloadable from mirrored sites.
5. **k_prod_attr**: This table contains some common attributes for products: size, weight, etc.
6. **k_prod_keywords**: Products keywords (for searching). Keywords are stored at a long varchar column, so there is one keyword record for each product.
7. **k_x_cat_objs**: This table holds which products belong to which categories.

When inserting or updating a Product, all these tables must be taken into account. That's what ProductLoader class from com.knowgate.hipergate Java package does.

Restrictions to product loading

ProductLoader class loads products from a delimited text file, there are strong restrictions about what can be loaded. It is only possible to load one fare and one location for each product.

Loading data from delimited text files

Class `com.knowgate.dataobjs.DBSubset` allows loading data from plain text files in the database with the combined use of `parseCSV()` and `store()` methods.

☞ You can also use class com.knowgate.hipergate.datamodel.ModelManager if you want to load a text file that exactly matches all the columns of a given table.

In next example two web pages are used: a static HTML POST and a JSP that takes the uploaded files and writes it to `k_companies` table.

Example file companies.csv

```
ABC,7f000001f8ac895053100000a64b23ce,NOVOMEDIA S.A.,ACTIVE,CLIENTS
ACS,7f000001f8ac895053100000a64b23ce,ACS S.A.,ACTIVE,CLIENTS
AD PEPPER,7f000001f8ac895053100000a64b23ce,AD PEPPER CORP.,ACTIVE,CLIENTS
ADQUIRA,7f000001f8ac895053100000a64b23ce,ADQUIRA ESPAÑA S.A.,ACTIVE,CLIENTS
AECE,7f000001f8ac895053100000a64b23ce,ASOC. ESP. COMERCIO,ACTIVE,CLIENTS
```

☞ `7f000001f8ac895053100000a64b23ce` is the GUID of the Workarea where data is to be inserted.

☞ Lookup values "`ACTIVE`" y "`CLIENTS`" must have been previously inserted at `k_companies_lookup` before loading companies.csv file.

Page csvpost.html

```
<HTML>
  <BODY>
    <FORM METHOD="post" ENCTYPE="multipart/form-data" ACTION="csvload.jsp">
      <INPUT TYPE="text" NAME="descriptor" SIZE="100" VALUE=
        "nm_legal,gu_workarea,nm_commercial,id_status,tp_company">
      <BR>
      <INPUT TYPE="file" NAME="csvdata">
      <BR>
      <INPUT TYPE="submit">
    </FORM>
  </BODY>
</HTML>
```

Page csvload.jsp

```
<%@ page
import="java.util.Enumeration,java.sql.SQLException,com.oreilly.servlet.Mul
tipartRequest,com.knowgate.jdc.JDCConnection,com.knowgate.dataobjs.*"
language="java" %>

<%@ include file="../methods/dbbind.jsp" %>

<%

  SQLException[] aExceptions = null;

  MultipartRequest oReq = new MultipartRequest(request, "/tmp");

  Enumeration oFileNames = oReq.getFileNames();

  DBSubset oDBS = new DBSubset(DB.k_companies, oReq.getParameter
("descriptor"), "", 0);

  oDBS.parseCSV ("/tmp/" + oReq.getOriginalFileName
(oFileNames.nextElement().toString()), "ISO-8859-1");

  JDCConnection oCon = GlobalDBBind.getConnection("csvload");
```

```
  oCon.setAutoCommit (false);

  aExceptions = oDBS.store (oCon,
Class.forName("com.knowgate.crm.Company"), false);

  if (oDBS.eof())
    oCon.commit();
  else
    oCon.rollback();

  oCon.close("csvload");

%>
<HTML>
  <BODY>
    <% if (oDBS.eof())

          out.write (String.valueOf(aExceptions.length) + " registers
                     successfully inserted");

       else {

         for (int e=0; e<aExceptions.length; e++) {

            if (null!=aExceptions[e])

              out.write("Line " + String.valueOf(e) + " " +
                        aExceptions[e].getMessage() + "<BR>");

         } // next (e)
       } // fi ()
    %>
  </BODY>
</HTML>
```

Example of a complex data loading

The next example shows a combined loading of contacts with their corresponding addresses from a delimited text file.

The file descriptor is passed as parameter in an INPUT field from page csvpost.html –see class `com.knowgate.misc.CSVParser` at JavaDoc-.

Variables `sGuWorkArea` and `sIdUser` of page `csvload.jsp` must be change for matching the desired Workarea and User.

Before loading the file `contacts.csv`, the lookup fields `de_title` for `k_contacts_lookup` and `tp_street`, `id_state` for `k_addresses_lookup` must be loaded.

At table `k_contacts_lookup` field `gu_owner` must be loaded with the GUID of the Workarea. Field `id_section` must be set to '`de_title`' and each `vl_lookup` field must be set with values {'`BUSINESS DEVELOPMENT MANGER`', '`CIO`', '`PRODUCT MANAGER`'} that are those values used at file `contacts.csv`.

At table `k_addresses_lookup`, `gu_owner` must be loaded equally loaded with `sGuWorkArea` and `id_section` must be loaded with `'tp_street'` and `vl_lookup = {'CALLE','AVDA.'}` and, else, `id_section='es'`, `vl_lookup='MAD'`.

Companies with legal name: ABC, ACS, AD PEPPER y ADQUIRA must exist at table `k_companies` before loading contacts.

Method `com.knowgate.crm.Contact.store()` assigns automatically a GUID for contact `gu_company` if a Company with the given legal name exists.

Example file contacts.csv

```
MR.;JOSÉ FRANCISCO;PEREZ;PRODUCT MANAGER;ABC;ABC;INTERNET;;CALLE;IGNACIO
LUCA DE TENA;7;;28027;MADRID;MAD;SPAIN;es;jfperez@abc.es;;91) 339-9000;;

MR.;JOAQUÍN;FERNÁNDEZ MARCOS;CIO;ACS;ACS;;Maite;AVDA.;ALFONSO
XII;98;;28036;MADRID;MAD;SPAIN;es;jfdez@acs.com;rcisneros@acs-poc.com;91)
343-9200;;

MR.;CHARLES;DEVILLE;BUSINESS DEVELOPMENT MANAGER;AD PEPPER;AD
PEPPER;;;CALLE;ORENSE;32
2ºD;;28020;MADRID;MAD;SPAIN;es;cdeville@adpepper.com;www.adpepper.com;91)
417-7450;68) 750-5785;

MR.;JACQUES;CHIRAULT;CIO;ADQUIRA;ADQUIRA;;;CALLE;GOYA;4 - 4ª
Pl.;;28001;MADRID;MAD;SPAIN;es;jchirault@adquira.com;;91) 436-3358;;
```

Page csvpost.html

```
<HTML>
  <BODY>
    <FORM METHOD="post" ENCTYPE="multipart/form-data" ACTION="csvload.jsp">
      <INPUT TYPE="text" NAME="descriptor" SIZE="100" VALUE="
        tx_salutation;tx_name;tx_surname;de_title;nm_commercial;nm_legal;tx
        _dept;contact_person;tp_street;nm_street;nu_street;tx_addr1;zipcode
        ;mn_city;id_state;nm_country;id_country;tx_email;url_addr;work_phon
        e;direct_phone;fax_phone">
      <BR>
      <INPUT TYPE="file" NAME="csvdata">
      <BR>
      <INPUT TYPE="submit">
    </FORM>
  </BODY>
</HTML>
```

Page csvload.jsp

```
<%@ page
import="java.util.LinkedList,java.util.ListIterator,java.util.Enumeration,j
ava.sql.SQLException,java.sql.PreparedStatement,com.oreilly.servlet.Multipa
rtRequest,com.knowgate.jdc.JDCConnection,com.knowgate.dataobjs.*,com.knowga
te.misc.CSVParser,com.knowgate.crm.Contact,com.knowgate.hipergate.Address"
language="java" %>
```

```
<%@ include file="../methods/dbbind.jsp" %>

<%

  // ********************************************
  // GUID of Workarea where data is to be inserted

  final String sGuWorkArea = "7f000001f8ac895053100000a64b23ce";
  final String sIdUser = "7f000001f8ac895158100001a0a10d3c";

  int iCol = 0;
  int iRow = 0;
  DBColumn oCol;

  String sContGUID, sAddrGUID, sField;

  Contact oCont = new Contact();
  Address oAddr = new Address();

  // ******************************************************************
  // Create a list of columns for both k_contacts and k_addresses tables

  LinkedList oContCols = oCont.getTable().getColumns();
  LinkedList oAddrCols = oAddr.getTable().getColumns();

  ListIterator oCols;

  // ***************
  // Upload CSV file

  MultipartRequest oReq = new MultipartRequest(request, "/tmp");

  Enumeration oFileNames = oReq.getFileNames();

  // *******************
  // Parse uploaded File

  CSVParser oParser = new CSVParser("ISO-8859-1");

  oParser.parseFile ("/tmp/" + oReq.getOriginalFileName(
oFileNames.nextElement().toString()), oReq.getParameter("descriptor"));

  // **********************************
  // Connect to Database using a DBBind

  JDCConnection oCon = GlobalDBBind.getConnection("contactload");

  try {

    oCon.setAutoCommit (false);

    PreparedStatement oStm = oCon.prepareStatement("INSERT INTO " +
DB.k_x_contact_addr + "(" + DB.gu_contact + "," + DB.gu_address + ") VALUES
(?,?)");

    for (iRow=0; iRow<oParser.getLineCount(); iRow++) {

      // *************************
      // Store Contact Information

      oCols = oContCols.listIterator();

      oCont.clear();

      while (oCols.hasNext()) {

        oCol = (DBColumn) oCols.next();
        iCol = oParser.getColumnPosition(oCol.getName());

        if (iCol>=0) {
             sField = oParser.getField(iCol, iRow);
```

```
        if (sField.length()>0)
          oCont.put (oCol.getName(), sField, oCol.getSqlType());

      } // fi (iCol>=0)

  } // wend

  iCol = oParser.getColumnPosition(DB.nm_legal);

  if (iCol>=0) {
    sField = oParser.getField(iCol, iRow);

      if (sField.length()>0)
       oCont.put(DB.nm_legal, sField);
  }

  oCont.put(DB.gu_workarea, sGuWorkArea);
  oCont.put(DB.gu_writer, sIdUser);

  oCont.put(DB.bo_private, (short) 0);
  oCont.put(DB.nu_notes, 0);
  oCont.put(DB.nu_attachs, 0);

  oCont.store(oCon);

  // GUID for Contact is automatically generated upon store
  sContGUID = oCont.getString(DB.gu_contact);

  // *************************
  // Store Address for Contact

  oCols = oAddrCols.listIterator();

  oAddr.clear();

  while (oCols.hasNext()) {

    oCol = (DBColumn) oCols.next();
    iCol = oParser.getColumnPosition(oCol.getName());

    if (iCol>=0) {
        sField = oParser.getField(iCol, iRow);

        if (sField.length()>0)
          oAddr.put (oCol.getName(), sField, oCol.getSqlType());

      } // fi (iCol>=0)

  } // wend

  iCol = oParser.getColumnPosition(DB.nm_commercial);
  if (iCol>=0) {
    sField = oParser.getField(iCol, iRow);

      if (sField.length()>0)
       oAddr.put(DB.nm_company, sField);
  }

  oAddr.put(DB.gu_workarea, sGuWorkArea);
  oAddr.put(DB.gu_user, sIdUser);
  oAddr.put(DB.bo_active, (short) 1);
  oAddr.put(DB.ix_address, 1);

  oAddr.store(oCon);

  // GUID for Address is automatically generated upon store
  sAddrGUID = oAddr.getString(DB.gu_address);

  // *************************
  // Link Address with Contact

  oStm.setString(1, sContGUID);
```

```
      oStm.setString(2, sAddrGUID);
      oStm.executeUpdate();

    } // next (iRow)

    oStm.close();

    oCon.commit();

    oCon.close("contactload");

  }
  catch (SQLException sqle) {

    oCon.rollback();
    oCon.close("contactload");
    oCon = null;

    out.write("Error at line " + String.valueOf(iRow+1) + " " +
sqle.getMessage());
  }

  if (null==oCon) return;
  oCon = null;

%>
<HTML>
  <BODY>
    <% out.write(String.valueOf(oParser.getLineCount()) + " contacts
successfully inserted"); %>
  </BODY>
</HTML>
```

Dumping database tables to text files

The class `com.knowgate.dataobjs.DBSubset` allows writing database tables directly to either delimited text or XML.

The `print()` method is used for generating delimited text files.

```
DBSubset oDBS = new DBSubset ("tabla", "campo1,campo2,campo3",
"1=1", 100);
oDBS.setColumnDelimiter("\t");
oDBS.setRowDelimiter("\n");
Connection oConn = DriverManager.getConnection("jdbc:…", "user",
"authstr");
FileOutputStream oFileOut = new FileOutputStream ("/tmp/out.txt");
oDBS.print (oConn, oFileOut);
oFileOut.close();
oConn.close();
```

The `toXML()` method is used for generating XML Strings.

```
Connection oConn = DriverManager.getConnection("jdbc:…", "user",
"authstr");
DBSubset oDBS = new DBSubset ("tabla", "campo1,campo2,campo3",
"1=1", 100);
oDBS.load(oConn);
String sXML = oDBS.toString("",null);
oConn.close();
```

Shortcuts for accesing lookup tables

In a typical OLTP application, a significant percentage of database accesses is done againist lookup tables that rarely change.

Lookup values are usually a few rows shown in a combobox or a popup window.

Class `com.knowgate.hipergate.DBLanguages` has some useful methods for quickly retrieving lookup values and reduce database accesses.

Security and User Roles

How to create a new Domain

Domains can be created using method `createDomain()` from class `com.knowgate.hipergate.datamodel.ModelManager`.

Creating a Domain is a lengthy and delicate process. For simplifying it, new domains are created by making a copy of an already existing one, the MODEL Domain -preloaded by the standard setup-.

During the copy process a unique numeric identifier is automatically assigned to the new domain. Also new GUIDs are created for Workareas, Users and Groups.

Data to be copied for each domain is defined at `domain_clon.xml` (one per DBMS) inside folder `com/knowgate/hipergate/datamodel/scripts/dbms/`. These XML files are interpreted and executed by `com/knowgate/datacopy/DataStruct`.

The Java source code for domain copy is a BeanShell script located called `domain_create.js` at `com/knowgate/hipergate/datamodel/scripts/` As it is an external non-compiled Java Source, it is possible to modify the domain creation process without recompiling the whole sources.

Domains may be created from Java following the next example:

```
import com.knowgate.hipergate.datamodel.ModelManager;

ModelManager oMan = new ModelManager();

oMan.connect("org.postgresql.Driver", "jdbc:postgresql:database",
“”, "usr", "pwd");

int iNewDomainId = oMan.createDomain("newdomain");

oMan.disconnect();

if (0==iNewDomainId) { /* Error */ }
```

Or domain may also be created from the command line using :

```
java com.knowgate.hipergate.datamodel.ModelManager
hipergate.cnf create domain domain_name
```

See: Scripts Java Bean Shell

How to delete a Domain

Deleting a domain also deletes all the data contained in it.

For deleting a domain from Java do :

```
import com.knowgate.hipergate.datamodel.ModelManager;

ModelManager oMan = new ModelManager();

oMan.connect("org.postgresql.Driver", "jdbc:postgresql:database",
"usr", "pwd");

int iNewDomainId = oMan.dropDomain("domain_name");

oMan.disconnect();

if (0==iNewDomainId) { /* Error */ }
```

Or from the command line :

```
java com.knowgate.hipergate.datamodel.ModelManager
hipergate.cnf drop domain domain_name
```

How to create an empty Workarea

Workareas are created using class `com.knowgate.Workareas.Workarea`.

For creating a Workarea programmatically it is neccesary :

1. Insert a register in `k_workareas` associated to a Domain.
2. Create directories for the Workarea.
3. Set permissions of User Groups for each Application.

☞ This creation routine does not insert any lookup values for the Workarea.

```
import java.sql.Statement;

import com.knowgate.workareas.*;

import com.knowgate.acl.ACLDomain;
import com.knowgate.jdc.JDCConnection;
import com.knowgate.dataobjs.DB;
import com.knowgate.dataobjs.DBBind;
import com.knowgate.misc.Environment;

// Get Database Binding
DBBind oDBB = new DBBind();

// Create Empty WorkArea
WorkArea oWrkA = new WorkArea();

// Get Connection from Pool
JDCConnection oCon = oDBB.getConnection("sample_workarea");

// Retrieve domain integer identifier from name
int iDomainId = ACLDomain.getIdFromName(oCon, "TEST1");

// Load Domain data
ACLDomain oDom = new Domain(oCon, iDomainId);

// Fill WorkArea fields
oWrkA.put (DB.nm_workarea, "workarea_name");
oWrkA.put (DB.id_domain, iDomainId);
oWrkA.put (DB.gu_owner, oDom.getString(DB.gu_owner));
oWrkA.put (DB.bo_active, (short)1);

// Store Workarea and get new GUID
String sWrkGUID = oWrkA.store(oCon1);

// Get Initialization Properties from hipergate.cnf

Properties oEnv = Environment.getProfile("hipergate");

// Create directory for Workarea under /web branch.
// Get Workareas put property from hipergate.cnf and
// create a subdirectory with the Workarea GUID

FileSystemWorkArea oFS = new FileSystemWorkArea (oEnv);

oFS.mkworkpath (sWrkGUID);
```

```
// Get /storage branch root directory.

String sStorageRoot = Environment.getProfilePath("storage");

String sSep = System.getProperty("file.separator");

// Compose path to Workarea subdirectory under /storage.
// Example: /opt/knowgate/storage/domains/1027/Workareas/…/apps

String sWrkBase = "file://" + sStorageRoot + "domains" + sSep +
String.valueOf(iDomainId) + sSep + "Workareas" + sSep + sWrkGUID +
sSep + "apps";

// Create directories for Workarea under /storage branch.

oFS.mkdirs (sWrkBase);

oFS.mkdirs (sWrkBase +  sSep + "MailWire");
oFS.mkdirs (sWrkBase +  sSep + "Sales");
oFS.mkdirs (sWrkBase +  sSep + "VirtualDisk");
oFS.mkdirs (sWrkBase +  sSep + "WebBuilder");

// Give permissions to domain administrators group over
Configuration Application.

final String Configuration = "30";

Statement oStm = oCon.createStatement();

oStm.executeUpdate("INSERT INTO k_x_app_workarea (id_app,
gu_workarea, gu_admins) VALUES(" + Configuration + ", '" +
sWrkGUID + "', '" + oDom.getString(DB.gu_admins) + "')");

oStm.close();

oCon.close("sample_workarea");
```

☞ When a new Workarea is created, remember to also create its directories.

How to duplicate a Workarea

In many case it is needed to create a Workarea with data in it rather than creating an empty Workarea. This can be achieved by copying an already existing Workarea.

For copying a Workarea type from the command line :

```
java com.knowgate.hipergate.datamodel.ModelManager
hipergate.cnf clone  origin_domain.origin_workarea
target_domain. target_workarea [verbose]
```

How to delete a Workarea

From Java Code :

```
import com.knowgate.workareas.WorkArea;
import com.knowgate.dataobjs.DBBind;
import com.knowgate.misc.Environment;

// Get Database Binding
DBBind oDBB = new DBBind();

// Get Connection from Pool
JDCConnection oCon = oDBB.getConnection("delete_workarea");

// Delete Workarea and its directories
WorkArea.delete (oCon, Environment.getProfile("hipergate"));

oCon.close("delete _workarea");
```

From the command line :

```
java com.knowgate.hipergate.datamodel.ModelManager
hipergate.cnf drop workarea domain.workarea [verbose]
```

How to create a User

1. Insert a register at table `k_users` with filelds `id_domain` and `gu_workarea` properly set.
2. Add User to the desired User Groups.
3. Create a *Home* Category for User.
4. Associate *Home* Category with User.
5. Set permissions of User over his *Home* Category.
6. Set permissions of Domain Administrators over User *Home* Category.
7. (Optional) Add User GUID into k_bugs_lookup and k_duties_lookup tables so that he can be assigned to Incidences and Duties. There is an example of how to do this at /vdisk/usernew_store.jsp page.

```
import com.knowgate.acl.*;
import com.knowgate.dataobjs.DB;
import com.knowgate.dataobjs.DBBind;
import com.knowgate.jdc.JDCConnection;
import com.knowgate.hipergate.Category;

Integer iOne = new Integer (1);

Short iTrue = new Short ((short)1);

// Create Empty User
```

```
ACLUser oUsr = new ACLUser();

// Get Database Binding
DBBind oDBB = new DBBind();

// Get Connection from Pool
JDCConnection oCon = oDBB.getConnection("create_user");

oUsr.put(DB.id_domain, 1027);
oUsr.put(DB.gu_workarea, "01234567890123456789012345678 9AB");
oUsr.put(DB.tx_nickname, "usernick");
oUsr.put(DB.tx_pwd, "donttell");
oUsr.put(DB.tx_main_email, "usernick");

// Step 1. Store New User
oUsr.store(oCon);

String sUsrGUID = oUsr.getString (DB.gu_user);

// **********************

// Load Domain data
ACLDomain oDom = new Domain(oCon, 1027);

// Get Administrator Group for Domain
ACLGroup oAdmins = new ACLGroup (oDom.getString(DB.gu_admins));

// Step 2. Add User to Administrators Group
oAdmins.addACLUser(oCon, sUsrGUID);

// ****************************************************

// Concatenate Domanin name and User Name,
// this concatenation will be used has the Home category Name.
String sDomainNick = oDom.getString(DB.nm_domain) + "_" +
"usernick";

// Get GUID of Category with name is DOMAIN_USERS,
// this is always the home categories parent for every domain.
String sParentId = Category.getIdFromName(oCon, oDom.getString
(DB.nm_domain) + "_" + "USERS");

// Step 3. Create User Home Category
String sHomeId = Category.create (oCon, new Object[] { sParentId,
sUsrGUID, sDomainNick, iTrue, iOne, "mydesktopc_16x16.gif",
"mydesktopc_16x16.gif" });

// Create Labels for Home Category
CategoryLabel.create (oCon, new Object[] { sHomeId, "es",
"usernick", null });

CategoryLabel.create (oCon, new Object[] { sHomeId, "en",
"usernick", null });

Category oCat = new Category (sHomeId);

// *******************************************

// Step. 4. Set reference to user Home Category
oUsr.replace (DB.gu_category, sHomeId);
oUsr.store (oCon);

// ***********************************
```

```
// Step 5. Set permission for new User
oCat.setUserPermissions (oCon, sUsrGUID,
ACL.PERMISSION_LIST|ACL.PERMISSION_READ|ACL.PERMISSION_ADD|ACL.PER
MISSION_DELETE|ACL.PERMISSION_MODIFY|ACL.PERMISSION_GRANT, (short)
1, (short) 0);

// ****************************************

// Step 6. Set permissions for Domain Owner
oCat.setUserPermissions (oCon, oDom.getString(DB.gu_owner),
ACL.PERMISSION_FULL_CONTROL, (short) 1, (short) 0);

// Set permissions for Domain Administrators
oCat.setGroupPermissions (oCon, oDom.getString(DB.gu_admins),
ACL.PERMISSION_FULL_CONTROL, (short) 1, (short) 0);

oCon.close("create_user");
```

Alternative script for creating default categories for a user :

```
import bsh.*;

import java.sql.*;


import com.knowgate.acl.*;
import com.knowgate.dataobjs.DB;
import com.knowgate.dataobjs.DBBind;
import com.knowgate.jdc.JDCConnection;
import com.knowgate.hipergate.datamodel.ModelManager;

final int iDomainId = 1027;
final String sWorkAreaId = "01234567890123456789012345678 9AB";

Integer iOne = new Integer (1);

Short iTrue = new Short ((short)1);

// Get Database Binding
DBBind oDBB = new DBBind();

// Get Connection from Pool
JDCConnection oCon = oDBB.getConnection("create_user2");

// Create User
String sUsrGUID = ACLUser.create (oCon, new Object[] { new
Integer(iDomainId), "usernick", "userpwd", "usernick", iTrue,
iTrue, iTrue, "usermail@domain.com", null, "John", "Smith", null,
"high school name", "Franklin", "ACME Inc.", "WebMaster",
sWorkAreaId });

// Get Power Users Group for Domain

Statement oStm = oCon.createStatement();

ResultSet oRst = oStm.executeQuery("SELECT " + DB.gu_acl_group + "
FROM " + DB.k_acl_groups + " WHERE " + DB.id_domain + "=" +
String.valueOf(iDomainId) + " AND " + DB.nm_acl_group + " LIKE
'Power%'");

oRst.next();
```

```
String sGrpGUID = oRst.getString(1);

oRst.close();
oStm.close();

// Create Power Users Group Object

oPowUsers = new ACLGroup(oCon, sGrpGUID);

// Add User to Power Users Group
oPowUsers.addACLUser(oCon, sUsrGUID);

Interpreter oInterpreter = new Interpreter();

// Create default categories for user with a BeanShell Script

String sCode = ModelManager.getResourceAsString
("scripts/user_categories_create.js");

// Set script input parameters

oInterpreter.set("UserId ", sUsrGUID);
oInterpreter.set("DefaultConnection", oCon);

// Run script

oInterpreter.source(sCode);

// Get script return values

Integer iCodError = (Integer) oInterpreter.get ("ErrorCode");
String sErrMsg = (String) oInterpreter.get ("ErrorMessage");
Object oRetVal = oInterpreter.get ("ReturnValue");
```

Queries By Form (QBF)

Queries By Form are a simple way of composing SQL queries against a single table or view.

At a QBF the User is requested to specify values for fields at a table or view. Registers that match the specified criteria are searched upon query execution.

At the standard model, QBFs allow up to 3 search fields connected by logical operators (OR, AND).

QBF XML definition files are stored at directory `/storage/qbf`.

A query is an instance of a QBF XML definition file bound to a set of search parameters. Queries are stored at `k_queries` table.

Example on a QBF XML definition file :

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<qbf>
  <title_es>Consulta de Tareas</title_es>
  <title_en>Duties Query</title_en>
  <method>post</method>
  <action>duty_list.jsp?selected=4&amp;subselected=1</action>
   <baseobject>v_duty_resource b</baseobject>
   <basefilter>(b.gu_owner='${cookie.workarea}')</basefilter>

   <fields>
     <field>
       <name>nm_duty</name>
       <label_es>Nombre</label_es>
       <label_en>Name</label_en>
       <type>varchar</type>
     </field>
     <field>
       <name>de_duty</name>
       <label_es>Descripcion</label_es>
       <type>varchar</type>
     </field>
     <field>
       <name>nm_project</name>
       <label_es>Proyecto</label_es>
       <label_en>Project</label_en>
       <type>lookup</type>
       <form>proj_tree_f.jsp?nm_table=void</form>
     </field>
     <field>
       <name>od_priority</name>
       <label_es>Prioridad</label_es>
       <label_en>Priority</label_en>
       <type>lookup</type>
       <form>lookup_f.jsp?nm_table=k_duties_lookup</form>
     </field>
     <field>
       <name>dt_start</name>
       <label_es>Fecha Inicio</label_es>
       <label_en>Stara Date</label_en>
       <type>date</type>
     </field>

     <columns>
       <column default="yes">
         <name>nm_duty</name>
         <label_es>Nombre</label_es>
       </column>
       <column default="yes">
         <name>de_duty</name>
         <label_es>Descripcion</label_es>
       </column>
       <column default="yes">
         <name>od_priority</name>
         <label_es>Prioridad</label_es>
       </column>
       <column default="yes">
         <name>tx_status</name>
         <label_es>Estado</label_es>
       </column>
       <column default="yes">
         <name>dt_start</name>
         <label_es>Fecha Inicio</label_es>
       </column>
     </columns>
```

```
    <sortable>
      <by>
        <name>nm_duty</name>
        <label_es>Nombre</label_es>
        <label_en>Name</label_en>
      </by>
    </sortable>
  </qbf>
```

How to create a query step by step.

1º) Create a new XML with tag <qbf>. Set a character set (UTF-8, ISO-8859-1).

2º) Choose a title for supported languages: es, en, fr, de, etc. Add tags <title_*xx*>.

3º) Choose a method for passing parameters to the HTML results page either "post" or "get".

4º) Set the URL of the results page <action>. Ampersand characters from the URL must be escaped as & entities. <action> page is only for HTM output. Page `/common/qbf.jsp` may also invoke `com.knowgate.http.HttpQueryServlet` servlet for sending delimited text files to the client.

5º) Set a base object and its alias. Base object must be a single table or view..

6º) Set base filter. Base filters are used for showing data for only one Workarea. When the user executes a query, the base filter is automatically added to it. The base filter contains two types of parameters ${cookie.*nombre*} and ${param.*nombre*} that are substituted at runtime with actual cookies and HTTP request parameters.

7º) Define columns for filtering. For each columns specify:

- 7.1) Actual name of column at database.
- 7.2) Translated label for each supported language.
- 7.3) Type { varchar | smallint | integer | flota | date | lookup }
- 7.4) Only for lookup type
  - 7.4.1) URL for lookup form.

8º) Define columns visualizable as output and whether they must be initially selected or not. Selecting columns only affects delimited text and Excel output and not HTML output.

9º) Specify columns suitable for ordering. Query may only sort by one of then.

10º) Save file at `/storage/qbf`.

11º) Parametrized instances for new query can now be created from page `/common/qbf.jsp`.

12º) Query can be directly executed using class `com.knowgate.hipergate.QueryByForm`

class `com.knowgate.http.HtppQueryServlet` contains an example of how to do so.

File Access

Class `com.knowgate.dfs.FileSystem` contains methods for unified local and remote (FTP) file management.

Storage files vs. Web Files

hipergate files are stored at two different directory branches: `/storage` and
`/web` (see Setup Instructions). `/web` must be local to the web server –or, at least, a network share-. `/storage` may be accessed locally or using FTP.

Basically files that cannot be accessed directly via HTTP GET requests are placed under `/storage` branch and files that can be retrieved via HTTP GET are placed under `/web/workareas` branch. `/storage` files can still be downloaded from a client browser using binary servlets from package `com.knowgate.http`.

Reading text files in a single step operation

Sometimes it is convenient to read a plain text file into a character array with a single step operation.
It is possible to use `com.knowgate.dfs.FileSystem.readfile()` for this purpose.

Converting ASCII files to Unicode

Class `com.knowgate.dfs.FileSystem` has method `convert()` for converting ASC-II files to Unicode.

XSLT Transformations

The content production module works by applying an XSL stylesheet to an XML file.

The model is founded in three key elements :

1. The template –also called XSL stylesheet-. Usually an XSL file for an XHTML output.
2. An XML file containing metadata about information allowed by the template.
3. Another XML file containing actual data entered by user.

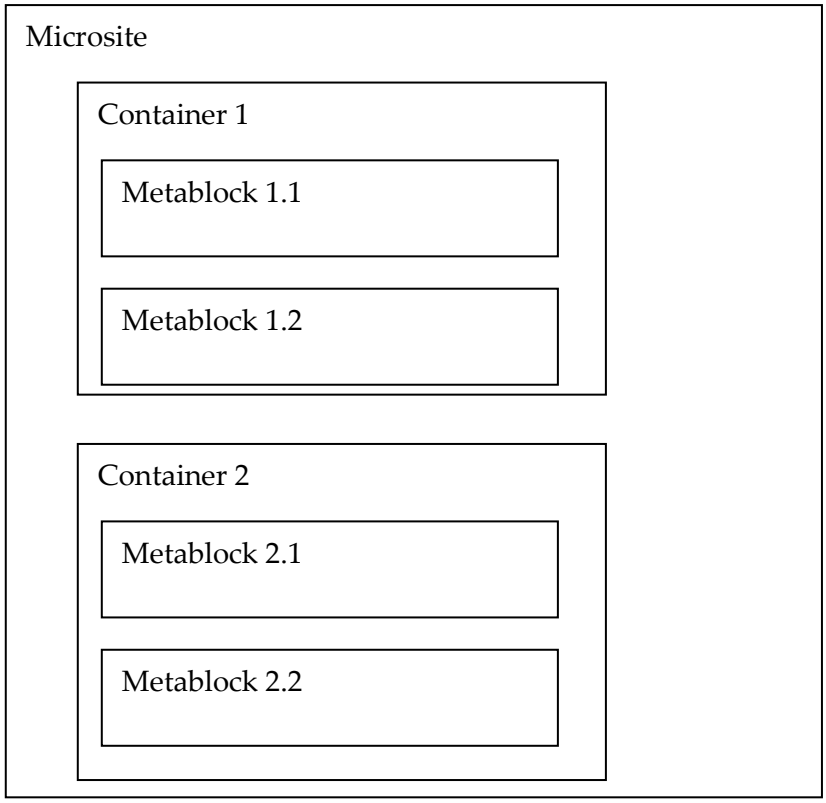
Metadata

Microsite: Metadata defining the structure of a page set. Microsites have *Contenedores* each container represents an XHTML page template.

Contenedor: Each XHTML page definition.

Metablock: Specification of what block types are allowed inside each Container.

☞ Microsites are stored in XML files that must comply with microsite.xsd schema.



Data

PageSet: A Microsite instance with actual user data.

Page: A Container instance with actual user data.

Block: A Metablock instance with actual user data.

☞ PageSets are stored in XML files that must comply with pageset.xsd schema.

☞ It is not neccesary to define metadata for paragraphs nor images because they are all the same and its structure is hardwired into Java code.

Simple Job Executor

The simple job executor is a single Java class (`com.knowgate.scheduler.SingleThreadExecutor`) designed for sequential processing of Atoms composing a Job.

Simple job executor is single thread and uses the database as direct support for job progress tracking.

Simple job executor is a small, robust and easy code piece, although it is not very fast or very scalable.

`SingleThreadExecutor` is ideal for processing small information volumes, for example for sending 100 to 500 e-mails or publishing a few files via FTP.

Mono-thread execution from the command line

It is possible to start a simple job executor from the command line for processing a single Job.

Two circumstances may occur :

a) the Job to execute is already created at `k_jobs` table or
b) the Job has to be created from an XML definition file.

Job Scheduler

The Job Scheduler is located at `com.knowgate.scheduler` package. Job Scheduler has a much richer internal structure than Simple Job Executor. Simple Job Executor is mono-thread and uses the database directly updating a row per processed atom. The Job Scheduler uses a RAM queue and a thread pool for better performance. The database is updated in batches. Job Scheduler has much better performance and scalability than Simple Job Executor, but is also much more complex.

The Job Scheduler is composed of the following elements :

1º) Jobs : Stored at `k_jobs` table.

2º) Commands : Each Job is the execution of a Command (commands are listed at `k_lu_job_commands`). The command tells the Job what to do with its Atoms. Each command behavior is implemented in a class from package `com.knowgate.scheduler.jobs`. Each command is a derived class from abstract base class `com.knowgate.scheduler.job`.

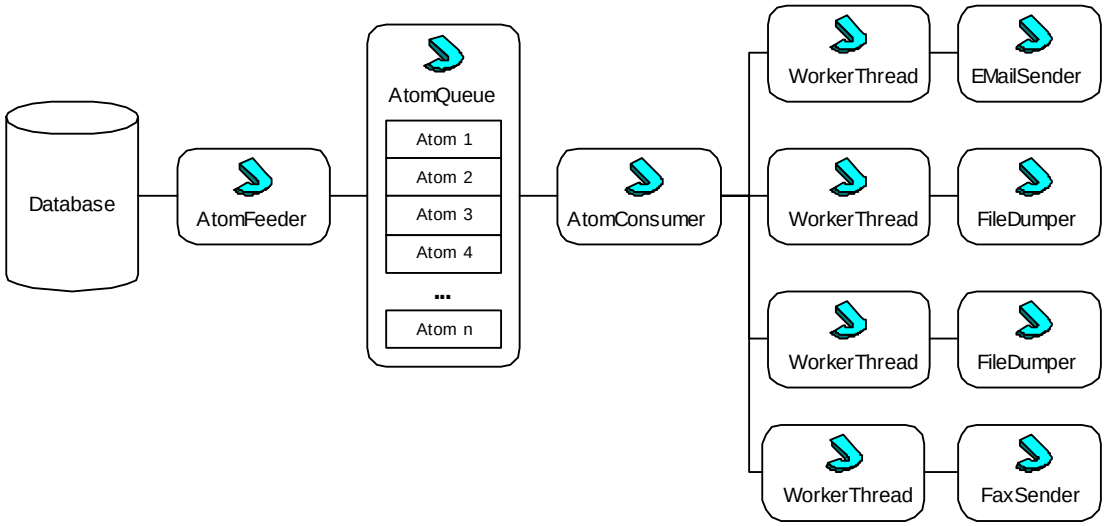
3º) Atoms : Each Job is divided into atomic units. The command is applying atom by atom as the Job execution proceeds. Pending atoms are stored at `k_job_atoms`. Already executed atoms are moved to `k_job_atoms_archived`. Atoms are the transactional unit for the Job. Each Atom is materialized in RAM as a Java object instance build taking data from `k_job_atoms`.

4º) Atom Execution Queue : The execution queue is a Java class that keeps in memory a limited number of Jobs pending of execution.
The queue exists for the sake of execution performance: it is faster to grab atoms in batch mode than get them one by one with SQL queries but, also, it is not feasible to load all atoms to be processed in RAM at a time as their number can be very large.

5º) Atom Queue Feeder : Is a Java object for feeding the queue with atoms pending of execution. Each Atom is represented itself as another Java object instance.

6º) Atom Consumer : Extracts atoms from the queue in a thread-safe way.

7º) Worker Threads: A pool of threads designed to execute job commands. Each thread gets the first available atom using the Atom Consumer object, then looks at the command to be executed for that atom and delegates behaviour to the proper Job subclass.



Command line execution

The Job Scheduler can be started from the command line using:
`com.knowgate.scheduler.SchedulerDaemon` *`path_cnf`* `[verbose]`

Where *`path_cnf`* is an absolute path to the initialization properties file, for example `/etc/hipergate.cnf`

If the `verbose` parameter is set then the progress of each worker thread will be shown through the console.

On start up the worker thread controller daemon will perform the following actions :

1st) Instantiate an execution queue in RAM memory with its Feeder and Consumer objects.

2nd) Connect to the database specified at `dburl` property from initialization file.

3rd) Create the pool of worker threads.

4th) Search at `k_jobs` table jobs pending of execution.

5th) Load a batch of atoms from pending jobs into the execution queue.

6th) Launch worker threads and start consuming atoms from the queue.

The worker threads can give information about their internal state at runtime by using callbacks.
For listening to internal worker threads events make a subclass of `WorkerThreadCallback` and implement abstract method `call()`.

Operations notified by worker threads are:

Código de Operación	Descripción
WT_EXCEPTION	An exception was raised inside the worker thread.
WT_JOB_INSTANTIATE	A subclass of Job was created.
WT_JOB_FINISH	Job execution finished.
WT_ATOMCONSUMER_NOMORE	No more atoms found at the queue.
WT_ATOM_GET	An Atom was extracted from the queue.
WT_ATOM_CONSUME	An Atom previously extracted from the queue was consumed.

An example of a `WorkerThreadCallback` subclass from the JobController Java Swing project :

```
package com.knowgate.jobcontroller;

import com.knowgate.scheduler.WorkerThreadCallback;

import javax.swing.JTextArea;

public class ThreadNotify extends WorkerThreadCallback {

  private final int BufferSize = 131072;

  private int iWritten;
  private JTextArea oTextArea;
  private StringBuffer oProgress;

  public ThreadNotify(String sCallbackName, JTextArea oTxtArea) {
    super (sCallbackName);

    StringBuffer oProgress = new StringBuffer(BufferSize);
    oTextArea = oTxtArea;
    iWritten = 0;
  }

  public synchronized void call (String sThreadId, int iOpCode, String
sMessage, Exception oXcpt, Object oParam) {

    if (iWritten+sMessage.length()>BufferSize) {
      oTextArea.setText(oProgress.substring(32767));
      oProgress.setLength(0);
      oProgress.append(oTextArea.getText());
      oProgress.append('\n');
      iWritten = oProgress.length();
    }

    oTextArea.append(sMessage + "\n");
    iWritten += sMessage.length() + 1;
  } // call

} // ThreadNotify
```

Create and register ThreadNotify as a callback :

```
import com.knowgate.scheduler. SchedulerDaemon;
import javax.swing.JTextArea;

JTextArea jProgressText = new JTextArea();
ThreadNotify oNotifText = new ThreadNotify("progress", jProgressText);
oNotifText.call("MainFrame", 0, "Creating SchedulerDaemon...", null, null);
SchedulerDaemon oSD = new SchedulerDaemon("/etc/hipergate.cnf");
oSD.registerCallback(oNotifText);
oNotifText.call("MainFrame", 0, "Starting SchedulerDaemon...", null, null);
oSD.start();
```

Setting up the number of executor threads

The maximum number of worker threads is set at `maxschedulerthreads` parameter from file hipergate.cnf.

Log Archive

For each Job a log archive is created at `/storage/jobs/`*`guid_workarea/guid_job`*`.txt`

Job Subclasses

There are 4 standard Job subclasses :

`DumyJob` : A do-nothing subclass of `Job` just for testing purposes.

`FileDumper` : The simplest `Job` subclass. Designed mostly as an example. Takes input files, replace personalization tags for each member of a distribution list and saves the resulting replaced files at the job log directory (`/storage/jobs/`*`guid_workarea/guid_job/`*).

`MimeSender` : Send custom e-mails.

`FTPPublisher` : Send files to a remote host using FTP.

For each Job subclass there is an associated command at table `k_lu_job_commands` :

class name	id_command
`com.knowgate.scheduler.jobs.DummyJob`	`DUMY`
`com.knowgate.scheduler.jobs.FileDumper`	`SAVE`
`com.knowgate.scheduler.jobs.MimeSender`	`SEND`
`com.knowgate.scheduler.jobs.FTPPublisher`	`FTP`

Environment properties for Job subclasses

Jobs can read environment properties from file hipergate.cnf or from any other properties collection specified when called `Job.instantiate()` method.

Properties from hipergate.cnf typically used by Jobs

`driver`	Driver JDBC
`dburl`	URL for database connection string
`dbuser`	Database user
`dbpwd`	User password
`workareasput`	Path for reading files previously generated by webbuilder
`maxschedulerthreads`	Maximum number of executor threads
`mail.transport.protocol`	Mail transport protocol
`mail.host`	Mail host
`mail.user`	User for mail host

Parameters for each Job subclass

Each Job subclass can have its own additional parameters from field `k_jobs.tx_parameters`. Parameters are stored delimited by commas, with format “name:value,name:value, name:value”.

Valid parameters are :

bo_attachimages Can be “1” or “0”. If “1” it means that images must be attached to messaged sent, they must be converted from `<IMG SRC="http://">` to `<IMG SRC="cid:">`.

bo_path	Can be “1” or “0”. If “1” it means that directories that do not exist at target file structure must be created dynamically upon FTP file transfer.
gu_list	GUID of distribution list where to send message copies.
gu_pageset	GUID of PageSet to be sent by e-mail or fax.
gu_workarea	GUID of Workarea to which PageSet to be sent belongs.
nm_file	Name of file to be copied to remote host.
nm_pageset	Name of PageSet to be sent by e-mail or fax.
nm_server	Name of remote host.
path	Path at remote host where files are to be copied.
tx_from	e-mail message sender.
tx_nickname	Nick of user used for login via FTP to remote host.
tx_pwd	Password of user used for login via FTP to remote host.
tx_sender	Full name of e-mail sender.
tx_subject	Message subject.

Sending e-mails from the command line

Have two different classes for sending e-mails from the command line:

```
com.knowgate.hipergate.SendMail
com.knowgate.scheduler.jobs.MimeSender
```

SendMail is a class suitable for sending a few messages to e-mails address stored at a plain text file external to hipergate database. SendMail does not allow personalizing the e-mail message for each recipient, or adding Web Beacons.

MimeSender is a bit more complex class used for sending e-mails using a job to members of a distribution list inside hipergate database.

MimeSender allows personalizing the e-mail for each recipient and adding Web Beacons to message HTML part.

How to send mail batches using SendMail

The class `com.knowgate.hipermail.SendMail` may be used for sending e-mails from the command line, with or without creating a job.

This class takes as command line parameter the path to a properties file (.cnf) that contains all the required information for the mailing.

The syntax for the command line is :

```
#java –cp htmlparser-1.6.jar:httpclient-4.0.jar:
httpcore-4.0.jar: httpmime-4.0.jar: jakarta-oro-
2.0.8.jar:javamail-1.4.0.jar
com.knowgate.hipermail.SendMail /etc/sendmail.cnf
```

or, for Windows:

```
C:\JRE\java –cp htmlparser-1.6.jar;httpclient-4.0.jar;
httpcore-4.0.jar; httpmime-4.0.jar; jakarta-oro-
2.0.8.jar;javamail-1.4.0.jar
com.knowgate.hipermail.SendMail C:\Temp\sendmail.cnf
```

If e-mails are sent using a job, then a second .cnf file is required with connection parameters of the database:

```
C:\JRE\java –cp htmlparser-1.6.jar;httpclient-4.0.jar;
httpcore-4.0.jar; httpmime-4.0.jar; jakarta-oro-
2.0.8.jar;javamail-1.4.0.jar
com.knowgate.hipermail.SendMail C:\Temp\sendmail.cnf
C:\Windows\hipergate.cnf
```

File sendmail.cnf must contain the following properties:

Property	Description
`mail.transport.protocol`	smtp
`mail.user`	User for SMTP service
`mail.password`	Password for SMTP service
`mail.smtp.host`	Mail host
`mail.smtp.socketFactory.class`	Only for SSL. Set this property to javax.net.ssl.SSLSocketFactory
`mail.smtp.socketFactory.port`	Only for SSL.

`recipients`	Path to file containing the recipients list (one recipient per line).
`encoding`	Character set. ISO-8859-1, UTF-8…
`userdir`	Path to directory that contains files composing the message
`textplain`	(Optional) File name of file containing the plain text part of the message. This file must be at userdir.
`texthtml`	(Optional) File name of file containing the HTML part of the message. This file must be at userdir.
`subject`	Subject
`from`	From e-mail address
`displayname`	From Display name
`replyto`	(Optional) Reply-To address
`recipienttype`	Recipient type: to, cc o bcc
`attachments`	(Optional) Attachments delimited by semi-colons.
`job`	(Optional) Job Title. If this property is set, then a second command line argument is required with properties specifing the connection parameters of the database.
`messageid`	(Optional) Unique identifier for message.

SendMail class and jobs

If a job title is set at the properties file, then a second command line parameter is required containing database connection information. A new Job will be created at k_jobs table with its corresponding atoms, one atom for each distinct e-mail recipient.

A Job can be re-started if it was unexpectedly stopped. In that case it will continue sending e-mail from the last processed one, skipping any previously sent e-mail. The job is identified by its title (column tl_job at k_jobs table).

How to send mail batches using MimeSender

hipergate allows sending personalized mail batches from the command line by using `com.knowgate.scheduler.jobs.MimeSender` class.
The command line utility creates a persistent job on the fly for sending the e-mails and starts sending them immediately.

For sending e-mails with MimeSender you need to have previously:

1. A mail account configured for a user of hipermail module.
2. A fulfilled list from hipergate contacts manager.
3. A template for mail message body (either plain text or HTML) and attached its files.
4. A .cnf properties file containing the required parameters for connecting to the database and identifying the target list.

Sample configuration file for bulk mail from the command line

```
# hipergate MimeSender bulk mailer configuration file

# Database
driver=org.postgresql.Driver
dburl=jdbc\:postgresql\://127.0.0.1\:5432/postgres
schema=
dbpassword=postgres
dbuser=postgres
poolsize=5
maxconnections=10
connectiontimeout=20000
connectionreaperdelay=31536000000

# File System
fileserver=localhost
fileprotocol=file\://
fileuser=
temp=C\:\\Windows\\Temp

# The e-mail templates must be under \mailing\mail.list
# subdirectory of storage directory
storage=C\:\\ARCHIV~1\\Tomcat\\storage

# Mail System

# mail.account must match a value from k_user_mail.tl_account
mail.account=account_name

# mail.list must match a value from k_lists.de_list
mail.list=List Description

mail.job.title=Test eMailing 2009-01-01
```

This file is much like the standard hipergate.cnf file but has less properties and an additional section # Mail System.

You may name this configuration file as you wish, but it must be placed at the same environment properties directory where hipergate.cnf is. The default environment properties directory is /etc for Linux and C:\Windows for Windows.

The `mail.account` must be the name of an account at webmail module of hipergate (column `tl_account` at `k_user_mail` table). For creating a new account inside the application go to Collaborative Tools – WebMail and click on the link at the right for creating new accounts.

The `mail.list` must be the description of the target recipients list (column `de_list` at `k_lists` table).

The mail.job.title may be any unique name that you want to give to the mail batch.

Mail contents template

The template for the mail to be sent must be put under storage/mailing/list_description directory. For the above example this directory will be more precisely:
`C\:\\ARCHIV~1\\Tomcat\\storage\\mailing\\List Description`

The e-mail may be either plain/text of HTML. But if it is HTML an alternative plain text part will be also added to the message body by extracting texts from the HTML part.

For sending the e-mails as plain text put a file named body.txt under storage/mailing/list_description directory and for sending an HTML email put a file named body.htm.

This is a sample HTML e-mail :

```
<HTML LANG=""en">
    <HEAD>
      <META CONTENT="text/html; charset=utf-8" HTTP-EQUIV="content-type" />
    <TITLE>hipergate 4.0</TITLE>
  </HEAD>
  <BODY>
    <IMG SRC="C:\ARCHIV~1\Tomcat\storage\mailing\List
Description\hipergate190x50.gif" WIDTH="190" HEIGHT="50" BORDER="0"
ALT="hipergate logo" />
     <BR/>
```

```
    Hello World!
    <BR/>
    This message is for {#Data.Name} {#Data.Surname}

    <!--WEBBEACON
SRC="http://www.yourserver.com/hipergate/hipermail/web_beacon.jsp?gu_job={#
Job.Guid}&pg_atom={#Job.Atom}&gu_company={#Data.Company_Guid}&gu_contact={#
Data.Contact_Guid}&tx_email={#Address.EMail}"-->
  </BODY>
</HTML>
```

All the images referenced at the HTML source code will be resolved and attached to the final message.

Personalizing custom mails

Mails can be personalized by inserting some predefined emailing tags that are replaced by values of columns at some tables from the database. In the example {#Data.Name} refers to recipient's name (column tx_name from k_contacts table) and {#Data.Surame} refers to tx_surname of the same table.

Tracking opened mails with Web Beacons in MimeSender

hipergate recognizes a special comment `<!--WEBBEACON SRC="…"-->` and replaces it by a 1x1 pixel image with the same SRC.

You can use this feature for placing any kind of Web beacon that you want inside each mail.

hipergate offers its own web beacon handler at /hipermail/web_beacon.jsp This page returns a 1x1 transparent GIF image and writes a line at `k_job_atoms_tracking` table.

This web beacon should receive as parameters the GUID of the Job that sent the email, the atom number, the GUID of Company and/or Contact and the e- address to which the mail was sent. This is done by inserting the following parameters at query string of web_beacon.jsp

gu_job={#Job.Guid}	Job GUID
pg_atom={#Job.Atom}	Atom Number
gu_company={#Data.Company_Guid}	Company GUID
gu_contact={#Data.Contact_Guid}	Contact
tx_email={#Address.EMail}	E-Mail Address

At runtime each {#...} tag is replaced by its proper value just before sending each e-mail.

MimeSender command line

The required components for running MimeSender are:

- The JDBC driver foryour database
- Sun JavaMail
- Apache Commons HttpClient
- Jakarta ORO
- HtmlParser

Thus a full command line for invoking MimeSender may be like:

```
java -cp postgresql.jar;jakarta-oro.jar;commons-
httpclient.jar;htmlparser.jar;javamail.jar;C:\Tomcat\webapps
\hipergate\WEB-INF\classes\
com.knowgate.scheduler.jobs.MimeSender mimesend.cnf
```

How does the mailing process work

The steps taken by the mailing routine at MimeSender.main() are:

1. Look up at `k_jobs` for a Job titled like `mail.job.title` property of the `.cnf` file specified as the commend line parameter.
    1.1. If no Job with the specified title already exists, create a new one and set its state to running (3) for preventing any other Job scheduler thread from trying to run it.
    1.2. If it is a new Job, retrieve e-mail addresses from the List specified at `mail.list` property of the `.cnf` file and create an Atom at `k_job_atoms` table for each e-mail address from the List by copying column values from `k_member_address` table into `k_job_atoms` table. Each Atom is initially written in running state (3).
2. Open an SMTP session using the connection parameter of the Account specified at `mail.account` property of the `.cnf` file.
3. Iterate through Job Atoms.
    3.1. For each Atom (e-mail recipient) at the message body replace {#...} tags by the actual values for Name, Surname, etc. from `k_job_atoms` table.
    3.2. Resolve and load any image references as inline attachments of the mime message.

3.3. If message body is HTML, create an alternative plain/text part by automatically extracting texts from the HTML.
3.4. Replace the <!--WEBBEACON --> comment by a 1x1 pixel image.
3.5. Send the resulting personalized mail using JavaMail.
3.5.1. If mail was successfully sent, move Atom from `k_job_atoms` table to `k_job_atoms_archived` table.
3.5.2. If there was any error whilst sending the e-mail, set Atom status to interrupted (4) and write a brief description of the error at `tx_log` column of `k_job_atoms` table.
3.6. Write progress information to system stdout.

4. When no more Atoms are available, set Job status to finished (0).

Job logs

Jobs execution logs are written to storage/jobs/guid_of_the_job.txt file.

How to write your own e-mailing routines

Sometimes it can be necessary to write your own custom algorithms for sending e-mails based on hipergate.

For sending personalized e-mails it is necessary :

1º) Have the recipients list in a computer suitable format: database, felt file, etc.

2º) Decide message media and format: e-mail HTML, e-mail text, or FAX

3º) Decide what parts of the message will be personalized: subject, recipient's name, text, etc. For each personalizable item define its custom tag.

4º) If there are files or images, decide how they will be attached or linked to the message.

5º) Choose a massive e-mailer program.

Obtaining recipients lists

The easiest way of obtaining recipients lists from hipergate is to use the web based query interface and export results to Excel or comma delimited files.

If the data extraction is done using Java code, the most practical way is using `print()` method of class `com.knowgate.dataobjs. DBSubset. print()` dumps all registers from `DBSubset` to a `Stream` using delimiter specified at `setColumnDelimiter()` and `setRowDelimiter()`.

If the recipients list is at a text file instead of at the database, it is possible to use `com.knowgate.misc.CSVParser` for reading the list into a bi-dimensional array. `CSVParser` can handle delimited text files with a register per line.

Personalization of messages

The process of personalizing mails must be fast enough for allowing sending a big volume of messages with a minimum memory and CPU consumption.

The class used for message personaliztion is `com.knowgate.dataxslt. FastStreamReplacer`. `FastStreamReplacer` takes an InputStream and a HashMap as input parameters. In a single step process, all keys with format "`{#name}`" from the HashMap are searched and replaced. Regular expressions are not allowed. Output is returned in a String.

Hipergate uses its own personalization tags for messages. Each tag corresponds to a field from the database which is placed instead of the tag just before sending the message. The tags can be in English or Spanish.

Class `com.knowgate.scheduler.Atom` has the tags table hardwired inside its Java code. During the process of personalization and sending, a subclass of `Job` gets the tag mapping from `Atom` and pass it as parameter to `FastStreamReplacer` for replacing the tag with actual values from the database.

Predefined tags hardwired into `Atom` Java class.

Database field	English tag	Spanish tag
Not a database field, it is replaced with today's date in short format `yyyy-MM-dd`	`System.Date`	`Sistema.Fecha`
`tx_name`	`Data.Name`	`Datos.Nombre`
`tx_surname`	`Data.Surname`	`Datos.Apellidos`
`tx_salutation`	`Data.Salutation`	`Datos.Saludo`
`nm_commercial`	`Data.Legal_Name`	`Datos.Razon_Social`
`tx_email`	`Address.EMail`	`Direccion.EMail`
`tp_street`	`Address.Street_Type`	`Direccion.Tipo_Via`
`nm_street`	`Address.Street_Name`	`Direccion.Nombre_Via`
`nu_street`	`Address.Street_Num`	`Direccion.Numero_Via`
`tx_addr1`	`Address.Line1`	`Direccion.Linea1`
`tx_addr2`	`Address.Line2`	`Direccion.Linea2`
`nm_country`	`Address.Country`	`Direccion.Pais`
`nm_state`	`Address.State`	`Direccion.Provincia`
`mn_city`	`Address.City`	`Direccion.Ciudad`
`Zipcode`	`Address.Zipcode`	`Direccion.Codigo_Postal`
`fax_phone`	`Address.Fax_Phone`	`Direccion.Telf_Fax`
`work_phone`	`Address. Proffesional_Phone`	`Direccion. Telf_Profesional`

Example of how to use FastStreamReplacer

```
import java.io.StringBufferInputStream;
import java.util.HashMap;
import com.knowgate.dataxslt.FastStreamReplacer;

FastStreamReplacer oReplacer = new FastStreamReplacer();

StringBufferInputStream oInput = new StringBufferInputStream
("Hello {#Data.Name} {#Data.Surname} today is {#System.Date}");

HashMap oMap = new HashMap();

oMap.put("Data.Name", "Paul");
oMap.put("Data.Surname", "Smith");

String sOutput = oReplacer.replace (oInput, oMap);

System.out.println (sOutput);
System.out.println (String.valueOf(oReplacer.lastReplacements()) +
" items replaced");
```

Images attached to messages

The e-mails sent with images require that those images are attached with the message body. In HTML the tag `<IMG SRC="cid:...">` is used for signaling a reference to an attached file. This means that before sending the message all http:// or file:// references must be changed to cid:

Class `com.knowgate.scheduler.jobs.EmailSender` uses `org.acmsl.htmlparser.html.HTMLProcessor` for finding all `<IMG>` tags and replaces `SRC` with a `cid:` reference.

Example of a class that uses SendMail for sending e-mails

```
import java.util.ArrayList;
import java.util.Properties;
import java.io.FileReader;
import com.knowgate.dfs.FileSystem;
import com.knowgate.hipermail.SendMail;
import com.knowgate.misc.Gadgets;
import com.knowgate.misc.CSVParser;

public class SMail {

  public static void main(String[] args) throws Exception {
    ArrayList oErrs = new ArrayList();

    // Read file with CSVParser class
    CSVParser oMails = new CSVParser("ISO-8859-1");
    oMails.parseFile("/tmp/emails.txt","email");
    int nLines = oMails.getLineCount();

    // Read mail host connection properties
    FileReader oRdr = new FileReader("/etc/sendmail.cnf");
    Properties oProps = new Properties();
    oProps.load(oRdr);
    oRdr.close();

    // Read plain text and HTML message parts
    FileSystem oFs = new FileSystem();
    String sText = oFs.readfilestr("file:///tmp/body.txt", "ISO-
8859-1");
    String sHtml = oFs.readfilestr("file:///tmp/body.html","ISO-
8859-1");

    for (int l=0; l<nLines; l++) {
      oErrs.clear();

      if (sSubject.length()>0) {

        try {

            oErrs = SendMail.send(oProps, "/tmp/", sBody, sText,
"ISO-8859-1", null, "Mail Subject", "from@hipergate.com", "From
Display Name", "noreply@hipergate.com", new String[]
{oMails.getField(0,l)}, null, null, null);
```

```
            System.out.println(oMails.getField(3,l)+" OK");

        } catch (Exception e) {
          System.out.println("ERROR "+oMails.getField(0,l)+"
"+e.getClass().getName()+" "+e.getMessage());
        }
      } // fi
    } // next
  }
} // SMail
```

How to use the shopping basket

The shopping basket is the Java class
`com.knowgate.hipergate.ShoppingBasket`

The shopping basket is not used from hipergate, instead it is designed to be used from a generic e-commerce site front-end.

The shopping basket has three Basic elements:

1. Customer Identifier
2. Global properties of the basket
3. Order lines

Global properties are a set of objects container incide the basket and retrieved by name.

Usually, every order line has the same attributes as the other ones: line number, sale price, taxes, etc.
There is no convention on how line attributes must be named.

How to load the basket from JavaScript

The easiest way of loading the shopping basket from JavaScript is composing a string in a hidden HTML element where all attributes for a line are written delimited by commas (or other) and use the method:
`ShoppingBasket.addLine(String, String)`
for each line.

This is, a client side page with:

```
<FORM>
<INPUT TYPE=”hidden” NAME =”id_customer” VALUE=”12345”>
<INPUT TYPE=”hidden” NAME =”tp_comprador” VALUE=”SOHO”>
<INPUT TYPE=”hidden” NAME =”line1” VALUE=”price=10, amount =1,product=xxx”>
<INPUT TYPE=”hidden” NAME =”line2” VALUE=” price =10,amount=1,product=yyy”>
</FORM>
```

And then at server side:

```
ShoppingBasket oBasket = new ShoppingBasket();
oBasket.setCustomer(request.getParameter("id_customer"));
oBasket.setProperty("type", request.getParameter("tp_comprador");
oBasket.addLine(request.getParameter("line1"),",");
oBasket.addLine(request.getParameter("line2"),",");
```

The shopping basket can the be kept at Session objectc of the servlet runner.

Search and sum functions

Shopping baskets allow searching for a line which has attributes of a given value. Also summing the values of a given attribute for all lines. Only attributes of type `java.math.BigDecimal` can be added.

Multicurrency support

Multiple currencies support is provided by classes `com.knowgate.hipergate.DBCurrency` and `com.knowgate.math.Money`

The Money class is an extension of `java.math.BigDecimal` that adds the currency in which the `BigDecimal` amount is expressed (`CurrencyCode`).

For performing currency conversions, it is possible to manually set a conversion rate, o let the library get one on real-time by calling ConversionRate web service from webserviceX.NET The call to the web service may take several seconds, so it is a good idea to cache locally the conversion rate.

The class DBCurrency is used for accessing tables `k_lu_currencies` and `k_lu_currencies_history`. Column `nu_conversion` is there to hold a fixed conversion rate for a base currency from foreign currencies. The base currency is not stored anywhere in the database and is dependent on the client application.

SMS interfaces

From version 5.0 hipergate supports sending short mobile messages (SMS). In order to send and SMS, hipergate need to connect to an SMS carrier.

There is no default carrier or account, when deploying hipergate an account must be contracted separately with the SMS carrier.
Each SMS carrier needs its own SMS interface. These carrier interfaces are implemented as subclasses of `com.knowgate.sms.SMSPush` abstract class.
The standard built has interfaces written for Sybase 365 and Altiria.
For implementing an interface for anew carrier just create a subclass and override methods connect() close() and push() from SMSPush abstract class.
You may look at `com.knowgate.sms. SMSPushSybase365` as an example implementation.

5 Java BeanShell Scripts

Java BeanShell is used for extending hipergate back-end functionality without recompiling core Java classes.

hipergate scripting technology is based upon 3 components :

Jakarta Bean Scripting Framework
BeanShell
Mozilla Rhino Shell

Parameter passing conventions between Java and BeanShell

BeanShell can take and return Java object as input and output parameters.

The following parameter names are used by convention:

DefaultConnection: Input. JDCConnection to database. In operations that involve reading from a data source and writing to another this is the Origin data source.

AlternativeConnection: Input. JDCConnection to database. In operations that involve reading from a data source and writting to another this is the Target data source.

ErrorMessage: Output. String. Text of the error message or "" if no error was raised.

ReturnValue: Output. Object. Script return value.

Example from `com.knowgate.hipergate.datamodel.ModelManager`

```
import bsh.*;
import com.knowgate.hipergate.datamodel.ModelManager;


// Create a new domain cloned from MODEL (1025)
public int createDomain(JDCConnection oConn, String sDomainNm)
throws EvalError, IOException, FileNotFoundException, SQLException
{

    Interpreter oInterpreter = new Interpreter();

    // Get script source code from .JAR
     String sCode = ModelManager.getResourceAsString
                                ("scripts/domain_create.js");

    // Set script input parameters
    oInterpreter.set("DomainNm", sDomainNm);
    oInterpreter.set("DefaultConnection", oConn);
    oInterpreter.set("AlternativeConnection", oConn);

    // Interpret script
    oInterpreter.source(sCode);

    Integer iCodError = (Integer) oInterpreter.get("ErrorCode");
    String sErrMsg = (String) oInterpreter.get("ErrorMessage");
    Object oRetVal = oInterpreter.get("ReturnValue");

    if (null!=oRetVal)
      return Integer.parseInt(oRetVal.toString());
    else
      return 0;
  }
```

6 JSP Pages

General structure of pages

Pages are grouped by directories according to the functional modules to which they belong.

/addrbook	Collaborative Tools
/common	Common pages used by several modules.
/custom	Pages from your own custom features.
/crm	Customers Relationships Management.
/dynapi	DynAPI Cross-Browser DHTML Library.
/examples	Examples.
/forums	Newsgroups (discussion forums).
/includes	HTML fragments for composing other pages.
/javascript	Client side JavaScript.
/jobs	Job Scheduler control interface.
/mailwire	Newsletters.
/methods	Static Java methods for inclusion in JSP.
/projtrack	Project Management.
/register	User registration.
/shop	Virtual Shop.
/skins	Application skins.
/vdisk	Virtual Disk and Corporate Library.
/wab	Import Windows Address Book.
/webbuilder	Contents Production.

Servlet container character encoding

hipergate versions 1.x use ISO-8859-1 (Latin1) character set for all pages.

From version the character was changed to UTF-8.

For enforcing the servlet container to use Unicode, the following sentence was included in file `/methods/dbbind.jsp` :

```
<% request.setCharacterEncoding("UTF-8"); %>
```

In case that the servlet container does not support `setCharacterEncoding()` the best is to change character encoding at method `loadrequest()` of files /methods/reqload.jspf and /methods/multipartreqload.jspf

```
public String recode (String badDecoded)
throws java.io.UnsupportedEncodingException {
    byte[] goodOriginal = badDecoded.getBytes("ISO-8859-1");
    return new String(goodOriginal, "UTF-8");
}

String nm_company = recode (request.getParameter("nm_company"));
```

Programming Conventions

Page headers

- Only clases used are must be imported.
- Pages have no session beans.

```
<%@ page
import="java.io.IOException,java.net.URLDecoder,java.sql.SQLExcept
ion,com.knowgate.jdc.JDCConnection,com.knowgate.acl.*"
language="java" session="false"
contentType="text/html;charset=UTF-8" %>
```

Connection Management

- Database connections must always be obtained from the pool. When obtaining a connection, a unique name must be assigned to it so that the statistics collector can later identify it.

- Connections must be closed explicitly even in the case that an exception is raised. Not closing connections properly will cause connection leaks and may end up blocking the whole database.
- All transactions must always be either committed or rolled back explicitly, failure to do so may block data pages or even entire tables indefinitely.
- JSP pages must get a database connection at the beginning of their execution and return it to the pool as soon as possible.
- It is recommended that no single database access lasts longer than 20 seconds.

Page Types

There are basically 4 page types :

- Menu Pages
- Listing Pages
- Edition Forms
- Save and Delete Pages

The control flow starts at the menu pages, then goes to listing pages, from there to edition pages and last to save/delete pages.

Application Beans

GlobalDBBind Data Access

`GlobalDBBind` is an instance of class `com.knowgate.dataobjs.DBBind` that acts as a singleton for caching database structure metadata. `GlobalDBBind` keeps the data model structure in RAM. This choice is used by `DBPersist` inherited objects for transparent persistence of Java classes to the RDBMS. `GlobalDBBind` also maintains a reference to the database connection pool.

☞ `DBBind` keeps the datamodel cached in memory. Each time the datamodel is changed, the application server must be restarted for `DBBind` to be able to see the changes.

Connecting to multiple databases at a time

`GlobalDBBind` reads its connection parameters from `hipergate.cnf` file by default from. As each instance of `DBBind` keeps a copy of the datamodel in memory, each `DBBind` can only be connected to one database at a time.

The easiest way of connecting to several databases without passing initialization parameters to `DBBind` is writing derived classes from `DBBind` that read their connection parameters from other files. The standard hipergate configuration has 4 precompiled classes for this purpose : `DBPortal`, `DBTest`, `DBDemo` and `DBReal`. Reading each one from files `portal.cnf`, `test.cnf`, `demo.cnf` y `real.cnf`. These four classes are simple inheritances from `DBBind` with no functionality except that they read from an initialization file other than `hipergate.cnf` by default.

GlobalCacheClient Distributed Cache

`GlobalCacheClient` is an instance of class `com.knowgate.cache.DistributedCachePeer`. At the client, the cache is a hash table (`java.util.HashMap`) that associates text strings (tokens) with Java objects. The cache has a limit set to 400 entries by default. The size of cached objects is not taken into account when limiting cache usage.

The local cache can work in parallel with local caches from other servers. For working in web farms a *cache coordinator* is needed. This cache coordinator is disabled by default; it may be enabled touching `appserver.cnf` file.

User Sessions

Cookies

Hipergate does not maintain server-side sessions. Information about logged users is kept by the client browser in session cookies.

All information is passed from one form to another using HTTP POST or GET methods. There is no centralized flow control.

It is not possible to have more than one session opened at the same time on a given client browser.

Session cookies set by `/common/login_chk.jsp` page are :

`domainid` Numeric Identifier of Domain to which user is logged.

`domainnm` Domain Name.

`skin` Name of active skin directory. This directory contains the CSS and the images that determine the application's look'n feel.

`userid` GUID of the connected user.

`authstr` User password. May be encrypted or clear text depending on how the login_chk.jsp page is configured. By default it is clear text.

`appmask` Bit mask of visible applications. Each application is identified by a bit between positions 1 and 31. The appmask is computed once and cached at the server for avoiding that a malicious cookie could make visible hidden applications.

`idaccount` Account Number (table `k_accounts`) for ASPs.

`workarea` GUID of active Workarea.

Cached information

For performance reasons, the role of users is cached into server's memory. In a certain way this is equivalent to maintaining certain session information about logged users with the difference that there is an absolute limit for resources that can be consumed at server-side by user sessions information.

Objects cached per user :

Token	Type	Description

[*UserId*,trial]	Boolean	**true** for trial accounts
[*UserId*,owner]	Boolean	**true** if is administrator of domain
[*UserId*,admin]	Boolean	**true** if belongs to domain administrator's group
[*UserId*,powuser]	Boolean	**true** if belongs to domain power users' group
[*UserId*,user]	Boolean	**true** if belongs to domain users' group
[*UserId*,guest]	Boolean	**true** if belongs to domain guests' group
[*UserId*,options]	String	HTML text of menu options
[*UserId*,suboptions]	String	HTML text of menu suboptions
[*UserId*,authstr]	String	User password
[*UserId*,mailbox]	String	Nombre del buzón de correo

User authentication procedure

Initial authentication

User authentication is done by page `/common/login_chk.jsp` :

1.a) If authentication information is a pair {e-mail , password} the user GUID and default Workarea must be previously found for the given e-mail. This information is extracted from fields `tx_main_email` y `gu_workarea` of table `k_users` by using `com.KnowGate.acl.ACLUser.getIdFromEmail()` method.

1.b) If authentication information is the set {nick , password, domain, workarea} only the user GUID must be found.

2) Once the user GUID has been retrieved (if it exists), method `com.knowgate.acl.ACL.autenticate()` is called for verifying that the user is active, his password is valid, and his account has not expired.

3) In ASP mode, verify that user billing account is active and has not expired.

4) Having verified the user's credentials and account status, the session cookies are written and the user role is cached at server -side.

5) From version 2.2, all login attempts are saved to table k_login_audit.

6) Control flow is redirected to `/common/desktop.jsp` page.

☞ User logins can be activated or deactivated without touching any user parameter. If field `k_users.bo_active` is set to zero the authentication process will consider the users as deactivated and refuse the connection request.

Re-authentication per page

Because the server does not maintain states, it is necessary to authenticate each HTTP request. This is done by calling method `autenticateSession (GlobalDBBind, request, response)` from `/includes/authusrs.jsp`

How to connect a User to different Workareas

The authentication process normally takes user e-mail and password for connecting to the application.

☞ Users belong to domains, not to Workareas, thus, when a new user is created it is usually assigned to a default Workarea (field `k_users.gu_workarea`).

Standard Authentication

The standard authentication from page `/common/login_chk.jsp` only takes user e-mail and password as input parameters.

The table `k_users` has a unique index on `tx_main_email` column that allows retrieving a user's GUID given his e-mail.

Also, the field `gu_workarea` of table `k_users` sets the Workarea to which user will be connected when logged by his e-mail.

Extended Authentication

In some cases it may be convenient that the same user can connect to several Workareas. This is achieved changing the pair {email, password} by {nickname, domain name, workarea name, password}. Page `login_chk.jsp` can process either set of logon parameters.

In practical terms, for connecting a user to a Workarea other than his default, replace page `login.html` with a form that has the following inputs :

```
<FORM METHOD="post" ACTION="login_chk.jsp">
  <INPUT TYPE="text" MAXLENGTH="32" NAME="nickname">
  <INPUT TYPE="text" MAXLENGTH="50" NAME="pwd_text">
  <INPUT TYPE="text" MAXLENGTH="30" NAME="nm_domain">
  <INPUT TYPE="text" MAXLENGTH="50" NAME="nm_workarea">
</FORM>
```

How to replace the native security model

Authentication logic is contained in class `com.knowgate.acl.ACL`, the include `authusrs.jsp`, and the page `login_chk.jsp`. By replacing these code fragments it is possible that the application uses an alternative authentication system (such as LDAP).

Before fully replacing the native security system, it must be taken into account that access rights for Categories at the Virtual Disk are tightly

coupled with the native security model. If just `com.knowgate.acl.ACL`, `authusrs.jsp` and `login_chk.jsp` are changed, the Virtual Disk will not work.

Anonymous users

The easiest way to allow anonymous access to hipergate back-end is by mapping all anonymous users to a single Guest account from a given Workarea.

For example, the following page automatically logs in a user and sends him to hipergate's main page.

```
<HTML>
  <HEAD>
    <SCRIPT LANGUAGE="javascript" SRC="/javascript/cookies.js">
    </SCRIPT>
    <SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
    <!--
    var dtInf = new Date(2099, 11, 30, 0, 0, 0, 0);

    setCookie ("skin",     "xp"  , dtInf);
    setCookie ("domainid", "2050", dtInf);
    setCookie ("domainnm", "DEMO", dtInf);

    setCookie ("userid",   "7f000001fa045a44b410000084bc7be2");
    setCookie ("authstr",  "demo");
    setCookie ("workarea", "7f000001fa0186e8b710000188942678");

    //-->
    </SCRIPT>
    <META HTTP-EQUIV="Refresh" CONTENT="0;
          URL=/common/desktop.jsp">
  </HEAD>
</HTML>
```

`skin` must be always xp.

Values for cookies `domainid` and `domainnm` must be taken from table `k_domains`.

Values for cookies `userid`, `authusr` and `Workarea` must be taken from table `k_users`.

Once inside the application, anonymous users may be identified by their Guest role by using function isDomainGuest() from file /methods/authusrs.jspf.

Most hipergate JSP pages automatically work in read-only mode when a Guest user is logged in, but it is best to check this point for avoiding potential security failures.

If a log-in facility must be given to anonymous users at any time, the best is to add a log-in box at /common/tabmenu.jspf. There are a few example code lines commented at that file.

```
<!--
    <td>
    <% if (isDomainGuest (GlobalDBBind, request, response) { %>

      <form method="post" action="/common/login_chk.jsp">

        e-mail: 
        <input type="text" name="nickname" class="box"
               size="30" maxlength="100"
               value="administrator@hipergate-test.com">

            
        password: 

        <input type="text" name="pwd_text" class="box"
               maxlength="30" size="30" value="TEST">

            
        <input type="submit" value="Login">

      </form>

    <% } %>
    </td>
-->
```

Static methods libraries for JSP pages

These methods can be found at directory `/methods` and are included in JSP pages using directives like

```
<%@ include file="../methods/authusrs.jspf" %>
```

authusrs.jsp

`autenticateCookie`

```
short autenticateCookie (
DBBind dbb, HttpServletRequest req, HttpServletResponse res)
throws ClassNotFoundException,InstantiationException,
ServletException
```

Checks that `userid` and `authstr` cookies correspond to a valid user/password pair. Password verification is tried first from the local cache. If there is a cache miss then the password is retrieved from the database. The `authstr` cookie must match the password value stored at local cache. If either password is not cached or both values are not the same method `com.knowgate.acl. ACL.autenticate()` is called for deciding whether or not password is valid.

Returns: `0` if user/password is valid or a bitwise OR combination of the following values if user/password is not valid.

```
ACL.USER_NOT_FOUND
ACL.INVALID_PASSWORD
ACL.ACCOUNT_DEACTIVATED
ACL.SESSION_EXPIRED
ACL.DOMAIN_NOT_FOUND
ACL.WORKAREA_NOT_FOUND
ACL.ACCOUNT_CANCELLED
ACL.PASSWORD_EXPIRED
ACL.INTERNAL_ERROR
```

`autenticateSession`

```
short autenticateSession (
DBBind dbb, HttpServletRequest req, HttpServletResponse res)
throws ClassNotFoundException,InstantiationException,
ServletException,IOException
```

Check that `userid` and `authstr` cookies correspond to a valid user/password pair. If not, redirects to `/common/errmsg.jsp` page.

Returns: `0` if user/password is valid or a bitwise OR combination of the following values if user/password is not valid.

```
ACL.USER_NOT_FOUND
ACL.INVALID_PASSWORD
ACL.ACCOUNT_DEACTIVATED
ACL.SESSION_EXPIRED
ACL.DOMAIN_NOT_FOUND
ACL.WORKAREA_NOT_FOUND
ACL.ACCOUNT_CANCELLED
ACL.PASSWORD_EXPIRED
ACL.INTERNAL_ERROR
```

cookies.jsp

`getNavigatorLanguage`

```
String getNavigatorLanguage (HttpServletRequest req)
```

Returns: "es" if client browser primary language is Spanish. "en" for any other language (v1.1).

getCookie

```
String getCookie (HttpServletRequest req, String sName,
String sDefault)
```

Get a cookie or a default value for it if not found.

nullif.jsp

nullif

```
String nullif (String sParam)
```

Return `sParam` if `sParam` is not **`null`** or `""` if `sParam` is **`null`**

nullif

```
String nullif (String sParam, String sDefaultVal)
```

Return `sParam` if `sParam` is not **`null`** or `sDefaultVal` if `sParam` is **`null`**

reqload.jsp

loadRequest

```
void loadRequest (ServletRequest r, DBPersist p)
throws NumberFormatException, java.text.ParseException
```

Load all keys from a `DBPersist` present as parameters of a `ServletRequest`. The routine iterates through `DBPersist` columns and finds if it exists a not null and not empty (`""`) parameter at the `ServletRequest` with that column name. Each parameter is converted from String to the type needed by the `DBPersist`.

Top tabbed menu

The menu is at `/common/tabmenu.jsp`

The application does not use frames on a regular basis. The top menu must be included is every page manually.

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1" %>
```

```
<jsp:useBean id="GlobalDBBind" scope="application"
class="com.knowgate.dataobjs.DBBind"/>

<jsp:useBean id="GlobalCacheClient" scope="application"
class="com.knowgate.cache.DistributedCachePeer"/>

<%@ include file="../methods/cookies.jsp" %>

<BODY CLASS="htmlbody">
    <%@ include file="../common/tabmenu.jsp" %>
…
</BODY>
```

The menu code requires methods from `cookies.jsp` and beans `GlobalDBBind` and `GlobalCacheClient`.

How to show the menu option currently selected

The menu is divided in two levels. Selected and subselected options must be passed as HTTP GET parameters called `selected` and `subselected` . The index is zero based. If `selected` < 0 no menu option will be selected.

Applications Mask

For each User and Workarea there is an application bit mask that is computed by looking at the Groups to which user belongs having those groups permissions assigned to application roles.

The application mask is a 32 bit integer with a bit per application.

Typically each application is a top menu option.

Menu options cache

Menu options enabled for each user are cached by `GlobalCacheClient` for better performance. As the menu HTML changes slightly depending on the selected option, it is not possible to cache just a single menu per user but all possible options must be cached as they are requested.

Domain Administrator special privileges

No matter what you specify at application roles, the domain administrators (identified by gu_owner field at k_domains table) always have access

granted to the Configuration section. This is enforced for avoiding an accidental removal of all administrators' privileges, leaving the domain impossible to administrate.

Personalizing the home page

From v2.1 hipergate includes the possibility of personalizing the application home page for each user.

hipergate personalizes the home page by using a kind of pseudo-portlets (Java classes implementing the `javax.portlet.GenericPortlet` interface). hipergate subclasses are not true portlets because hipergate is not a portlet container so what these classes do is using `HttpServletResponse` as an emulation of a true portlet container.

For each portlet the following elements are involved on the rendering process:

- The container page
- The Java class implementing the portlet interface
- The XSL stylesheet used for rendering portlet HTML contents
- A row at k_x_portlet_user that determines porlet position and state
- A set of properties to be passed as parameters to the portlet
- A cache file for each portlet instance per user, page and Workarea

How page composition is done

1. For each user and zone, a DBSubset is retrieved from GlobalCacheClient DistributedCachePeer application bean. If there is no cached entry then the information is loaded from k_x_portlet_user table at the database.
2. For each zone, the list of portlets that must be show there for the current user and Workarea is iterated.
3. Collect environment properties needed for `GenericPortlet.render()` method into an `HipergateRenderRequest` object instance. The call `render()` passing `HipergateRenderRequest` and `HipergateRenderRequest` as parameters.
4. From inside render() method, determine if the `dt_modified` timestamp of `k_x_portlet_user` is before or after the date of creation of the cache file for the portlet content located under storage/domains/*domain_id*/workareas/ *workarea_guid*/cache/*user_id* If cache is no longer valid, get the portlet XSL stylesheet from the

StylesheetCache merge it with portlet contents, update cache file and write results to `RenderResponse` object.

Portlet caching

Because portlets are called each time a page is loaded and a page can contain several portlets, it is important to implement them in an efficient way. hipergate portlets rely on 3 different caching mechanisms for improving portlet performance:

1st) Information about which portlet must be painted on each page and their state is cached into GlobalCacheClient DistributedCachePeer application bean.
2nd) The portlet output is cached for each page, user and Workarea in a file under storage directory. If the portlet underlying data has not changed then the cached file is served and no database access is done.
3rd) Each portlet need an XSL stylesheet for rendering its output as XHTML. Loading and parsing an XSL stylesheet may be a costly task, so stylesheets are kept into an internal cache and reused on each call.

Selecting lookup values

If the standard convention is used for creation lookup tables then it will be possible to reuse the same JSP page set for maintaining all lookups of the application.

This convention consists in creating a *base table* (for example, `k_base`) and another table `k_base_lookup` (same name as the base table but with `_lookup` suffix). `k_base_looukp` table contains all lookup values for all fields `k_base` for each Workarea.

Lookup values are always of CHARACTER VARYING type.

For each lookup it must be specified:

1º) Section name –typically the name of the column at `k_base`–.
2º) Internal value for lookup (`k_base_looukp.vl_lookup`).
3º) Displayed labels for each language.

How to call page lookup_f.jsp

Page lookup_f.jsp is opened in a new window from the maintenance form of each table.

☞ It is possible to open all lookup forms in the same window or in a new window. In Internet Explorer there is a great performance difference between opening in a named window and opening in an unnamed (new) window being the second must faster. For this reason it is recommended to open each lookup in a new window.

GET Params.

`nm_table`	Name of lookup tables.
`id_language`	Language for displaying labels.
`id_section`	Section, usually the name of the column at base table.
`nm_control`	Name of HTML INPUT control at table maintenance form that will hold the lookup translated label.
`nm_coding`	Hidden HTML INPUT at table maintenance form that will hold the internal lookup value.
`tp_control`	Type of control at base table form. `1` - <INPUT TYPE="text"> `2` - <SELECT>

Loading lookups from SQL scripts

Lookup values can be loaded using SQL scripts instead of HTML forms.

For doing so, it is necessary to identify the lookup table where register are to be inserted and set:

- the Workarea
- lookup numeric identifier [1..n]
- section (base column)
- internal lookup value
- translated labels

For example:

```
INSERT INTO k_companies_lookup
(gu_owner,id_section,pg_lookup,vl_lookup,tr_es,tr_en) VALUES
('$gu_workarea$','id_sector',1,'A','Agricultura',
'Agriculture');
```

```
INSERT INTO k_companies_lookup
(gu_owner,id_section,pg_lookup,vl_lookup,tr_es,tr_en) VALUES
('$gu_workarea$','id_sector',2,'I','Industria',
'Industrial');
```

Fields to be filled are :

`gu_owner`: GUID of Workarea to which registers will belong.

`id_section`: Name of section (base column).

`pg_lookup`: Lookup numeric identifier. It is a unique number per Workarea, althoughit may be repeated for different Workareas.

`vl_lookup`: Actual internal value that will be stored at base table id_section.

`tr_es`: Spanish Label.

`tr_en`: English Label.

User Defined Attributes

hipergate allows adding user defined attributes to standard entities. These attributes are not physically created as columns in the datamodel but stored as metadata at special tables.

At JSP application layer, user defined attributes are managed by file **`/methods/customattrs.jsp`** this module must be included in JSP pages using `<%@ include file="../methods/customattrs.jsp" %>`

For painting user defined attributes in a form, method `paintAttributes()` does as follows:

1. Read metadata definition that corresponds to current table and workarea from `k_lu_meta_attrs`.
2. Metadata information is cached locally with key *table#language*[*workarea*] –for example `k_contacts_attrs#en[01234567890123456789012345678 9AB]`. Next time `paintAttributes()` method is called metatadat will be readed from the cache and not from the database.

3. If the logged user belongs to the Workarea administrators' group then paint *add attribute* and *remove attribute* links.
4. Attributes names are listed separated by commas at HTML control: `<INPUT TYPE="hidden" NAME=" custom_attributes">`. This hidden list of named is used by `storeAttributes()` method for knowing what attributes exist.
5. Finally each attribute is written with its corresponding value.
6. Method `paintAttributesHidden()` is the same as `paintAttributes()` except that it paints attributes in hidden HTML controls instead of in visible HTML controls. This feature is used for transferring data between 2 forms of the same page.

Sending e-mails in response to user actions

The class `com.oreilly.servlet.MailMessage` is a convenient wrapper on top of JavaMail for sending e-mails through SMTP.

The following code example can be added to any page storing user data.

```
MailMessage msg = new MailMessage("mail.mydomain.com");
msg.from("Sender Display Name");
msg.to("recipient@hisdomain.com");
msg.setHeader("Return-Path", "noreply@mydomain.com");
msg.setHeader("MIME-Version","1.0");
msg.setHeader("Content-Type","text/plain;charset=\"utf-8\"");
msg.setHeader("Content-Transfer-Encoding","8bit");
msg.setSubject("Mail message subject");
msg.getPrintStream().println("Plain text message");
msg.sendAndClose();
```

Example Pages

The directory `/examples` contains pages that may be used as base examples for witing new ones.

listing.jsp

This page is an example of how to generate a screen listing that shows rows read from the database. It is possible to specify a filter for retrieving only the rows that meet a given criteria

Steo by Step what `listing.jsp` does is :

1. Get language of client browser.

2. Get selected skin reading its cookie.
3. Get current Domain and Workarea reading their cookies.
4. Read screen resolution parameter, if not found assume 800x600.

   ☞ It is not possible to read client screen resolution from JSP code without passing it as an http GET parameter.

5. Read clauses for filtering registers. This clauses may be of two types :
   5.1 A WHERE clause either freely composed or obtained from a QBF.
   5.2 Pair {Field, Value} for finding rows with a field matching an specific pattern. The searched value is concatenated as `'%Value%'` and searched with a `LIKE` operator in the designed column.
6. Read maximum rows to retrieve and row offset.

   ☞ Since the relational model does not guarantee any order for retrieving rows, scrolling registers by reloading the same page but skipping a given number of rows is not really a consistent operation unless an order by is specified for results. The same sentence executed twice could, potentially, return the same rows in different order.

7. Read order by column for results.
8. Get a connection from the pool. The name `"instancelisting"` for the connection must be changed so that the statistics collector can identify the origin page of the connection request to the pool.
9. If page received HTTP GET parameter `where` then apply filter to SQL sentence of data retrieval.
10. If page received HTTP GET parameters `field` and `find` then retrieve only rows matching the pattern `find` specified for field.
11. If there is no filter then retrieve rows up to the maximum specified.
12. Definir la llamada a la página para crear una nueva instancia. Hay que cambiar el archivo `"instance_edit.jsp"` por el nombre del formulario de creación pertinente.
13. Define how to call the JSP page for deleting instances. The file `"instance_edit_delete.jsp"` must be replaced with the name of the actual page. During the page loading process, the primary key for read rows is stored in a JavaScript array. When `deleteInstances()` method is called it verifies which checkboxes are checked and appends the primary key of row for that checkbox to hidden input control `checkeditems`; this control holds the primary keys of rows to be deleted separated by commas and it is posted to the delete page.
14. Define the call to instance edition page. File `"instance_edit.jsp"` must be replaced by the actual edition form.
15. Define JavaScript function for reloading the page ordering by a given column.

16. Define JavaScript function for selecting/deselecting all instances.
17. Define JavaScript function for looking for a given pattern in a column.
18. Define call to clone page. For cloning an instance it is neccesary to have previously written an XML file with the definition of the instance as a *complex data structure*. These XML definition files are placed at `/storage/datacopy` directory. Input Parameters for `/common/clone.jsp` page are :

`id_domain`	Numeric Identifier of current Domain.
`nm_domain`	Name of current Domain.
`datastruct`	Name (without extension) of XML file that contains the definition of the entity to be cloned.
`gu_instance`	GUID of entity to be cloned.
`opcode`	4 letter operation code for auditing purposes.
`classid`	Numeric identifier of class of cloned object (from `k_classes`).

    Once the `clone.jsp` page is opened, a timer is created with an active wait llop executing every 100 miliseconds. When the clone process is finished the popup window closes automatically and the listing page is refreshed with a filter looking for the cloned instance.
19. Paint the top menu with search boxes.
20. Paint Previous and Next links.
21. Paint rows readed from the database.
22. Set right mouse button context menu for entities.

simpleform.jsp

This page is an example of a simple maintenance form for a register of the database. Maintenance forms may either create new registers or update already existing ones. The design pattern is that each maintenance form makes a posts edited data to a page with the same name but ended with `_store.jsp`.

Step by step, the actions performed by `simpleform.jsp` are:

1. Verify user credentials from cookies.
2. Add no-cache page headers.
3. Get environment and the primary key of the object to be editing (if it already exists).
4. Create an instance of a subclass of DBPersist used for loading values.

5. Get a connection from the pool. The name of the connection must be changed from the default so that the statistics collector can identify the origin of each connection request and detect leaks.
6. Get lookup values using method `DBLanguages.getHTMLSelectLookUp()`
7. JavaScript function for showing month calendar popup. This function receives as parameter the name of the HTML control where to return the date selected at the popup.
8. JavaScript function for opening lookup selection form.
9. JavaScript function for validating data at client side prior to sending it to server.
10. Common actions menu.
11. Example of mandatory fields.
12. Example of optional fields.
13. Example of selecting lookup values.
14. Example of selecting dates from calendar.

Loading dates from HTML forms to the database

By convention all dates have format YYYY-MM-DD [HH24:MI:SS] no matter what language is used.

There are 4 standard procedures for loading dates into the databases which shall be called : *Cast*, *Split*, *Parse* and *Escape*.

1ª) Cast. A type conversion for producing a SQL sentence only valid for a particular RDBMS.

```
import java.sql.Statement;

import com.knowgate.jdc.JDCConnection;
import com.knowgate.dataobjs.DBBind;

// Get Database Binding
DBBind oDBB = new DBBind();

// Get Connection from Pool
JDCConnection oCon = oDBB.getConnection("cast_date");

String sDateTime = "2003-09-18 02:38:00";
String sDateDB;

swhich (oCon.getDataBaseProduct()) {

  case JDCConnection.DBMS_ORACLE:
    sDateDB = "TO_DATE('" + sDateTime + "','YYYY-MM-DD
HH24:MI:SS')";
    break;

  case JDCConnection.DBMS_POSTGRESQL:
    sDateDB = "TIMESTAMP '" + sDateTime + "'";
```

```
    break;

  default: // ODBC escape syntax
   sDateDB = "{ts '" + sDateTime + "'}";
}

Statement oStm = oCon.createStatement();

oStm.executeUpdate ("UPDATE k_users SET dt_modified=" + sDateDB);

oStm.close();

oCon.close("cast_date");
```

2ª) Split. Convert a short date input into a Java Date and then bind it as a JDBC input parameter.

```
import java.sql.PreparedStatement;
import java.sql.Timestamp;

import java.util.Date;

import com.knowgate.jdc.JDCConnection;
import com.knowgate.misc.Gadgets;

// Get Database Binding
DBBind oDBB = new DBBind();

// Get Connection from Pool
JDCConnection oCon = oDBB.getConnection("split_date");

String aDt [] = Gadgets.split("2003-09-18", "-");

Date oDt = new Date (Integer.parseInt(aDt[0]),
Integer.parseInt(aDt[1]), Integer.parseInt(aDt[2]));

Timestamp oTs = new Timestamp (oDt.getTime());

PrepareStatement oStm = oCon.prepareStatement("UPDATE k_users SET
dt_modified=?");

oStm.setTimestamp (1, oTs);

oStm.executeUpdate ();

oStm.close();

oCon.close("split_date");
```

3ª) Parse. Convert a datetime input into a Java TimeStamp and then bind it as a JDBC input parameter.

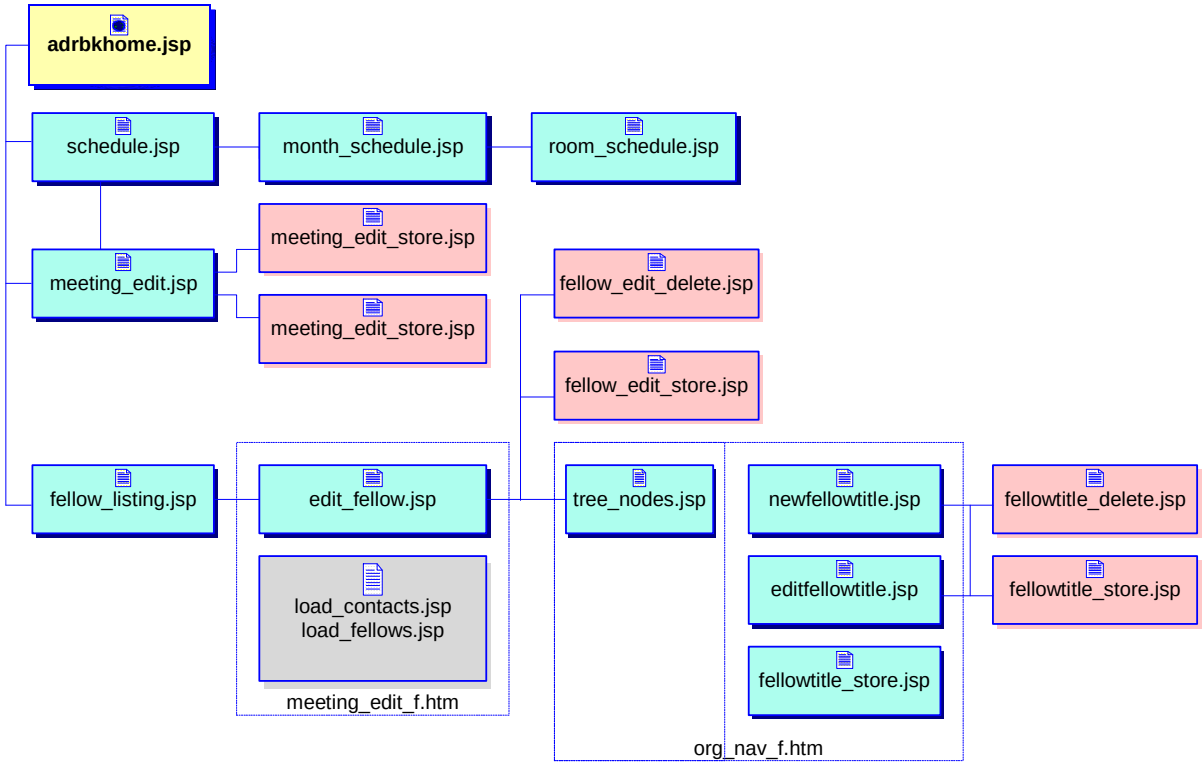
```
import java.text. SimpleDateFormat;

import java.sql.PreparedStatement;
import java.sql.Timestamp;
```

```
import com.knowgate.jdc.JDCConnection;
import com.knowgate.misc.Gadgets;

// Get Database Binding
DBBind oDBB = new DBBind();

// Get Connection from Pool
JDCConnection oCon = oDBB.getConnection("parse_date");

SimpleDateFormat oFmt = new SimpleDateFormat ("yyyy-MM-dd
hh:mm:ss");

String sDt = "2003-09-18 02:38:00";

Timestamp oTs = new Timestamp(oFmt.parse(sDt).getTime());

PrepareStatement oStm = oCon.prepareStatement("UPDATE k_users SET
dt_modified=?");

oStm.setTimestamp (1, oTs);

oStm.executeUpdate ();

oStm.close();

oCon.close("parse_date");
```

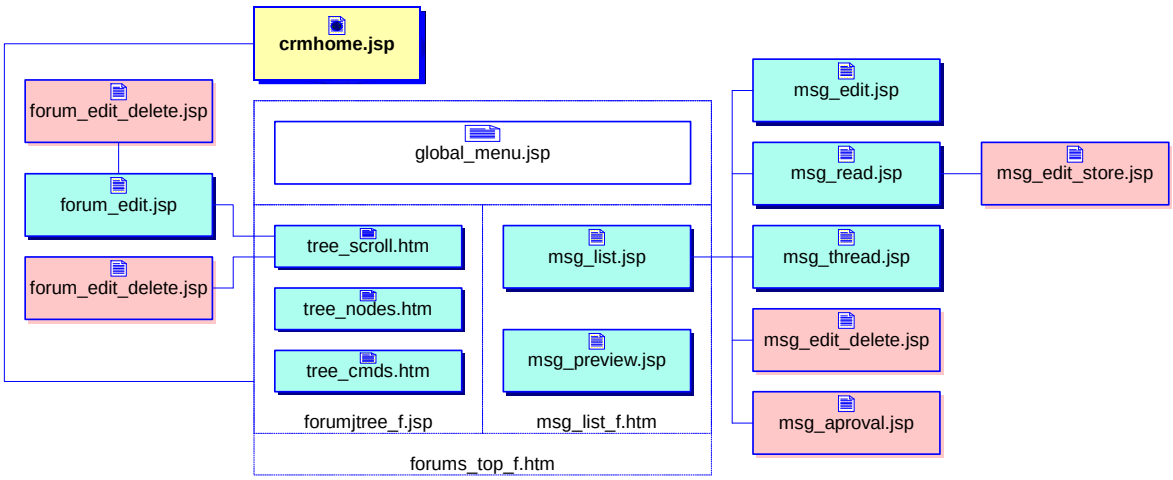
4ª) Escape. Use `escape` method from class `DBBind` with parameter "`ts`" for a Timestamp or "`d`" for a short date.

```
import java.sql.Statement;

import java.util.Date;


import com.knowgate.jdc.JDCConnection;
import com.knowgate.dataobjs.DBBind;

// Get Database Binding
DBBind oDBB = new DBBind();

// Get Connection from Pool
JDCConnection oCon = oDBB.getConnection("escape_date");

// Get different escape or cast date strings depending on which
database management system the DDBind is connected.

String sDt = DBBind.escape(new Date(),"ts");

Statement oStm = oCon.createStatement();

oStm.executeUpdate ("UPDATE k_users SET dt_modified=" + sDt);

oStm.close();

oCon.close("escape_date");
```

Collaborative Tools Module



Forums Module



How forum messages are stored

Forum messages bodies are stored in clear text at `tx_msg` field from `k_newsmsgs` table. Message contents may be either plain text or HTML but is the responsibility on the presentation layer to handle HTML tags.

Message body can be as large as the database CLOB or LONGVARCHAR type allows it.

Message attachments are stored outside the database under /storage directory branch and are referenced from `k_prod_locats` table.
If a message has attachments then its `gu_product` column is not null. This column points to a single Product object from `k_products` table that have one ProductLocation for each attached file.
hipergate default front-end limits the number of attachments to four per message, but the database can hold an unlimited amount of them.

NewsMessage attachments are downloaded using com.knowgate.http.HttpBinaryServlet, see the JavaDoc API for more information about HttpBinaryServlet.

Creating RDF Site Summary (RSS) documents for a forum

hipergate includes a couple of example of how to generate RSS documents from forums messages.

`/forums/msg_rss10.jsp`	Generate RSS 1.0
`/forums/msg_rss20.jsp`	Generate RSS 2.0

These pages are only a foundation for generating XML docs from messages. They must be customized for each client application.

Input Parameters

Each page takes the following input parameters by HTML GET or POST:

`gu_newsgrp`	NewsGroup GUID.
`nm_newsgrp`	NewsGroup Name.
`id_language`	Language Identifier (2 characters code).
`nu_messages`	Maximum number of messages to show.

Control Parameters

Inside JSP code there are 2 variables that control output format:

`ENCODING`	Sets how `<description>` tags will be codified. It may be :	
	`ENCODE_NONE`	Text is dumped in XML as it goes out of the database.
	`ENCODE_HTML`	Text is passed through function `Gadgets.HTMLEncode()` non ASCII-7 characters are converted into HTML entities.
	`ENCODE_CDATA`	Text is enclosed by tags `<![CDATA[...]]>`
	The default value is `ENCODE_HTML`.	
`MAX_MSG_DESC_LEN`	Maximum length of text inside tags `<description>`. The default value is 200 characters.	

Creating a blog or a static posts page set

It is possible to create an HTML static copy of all posts from a NewsGroup by using class com.knowgate.forums.NewsGroupJournal. NewsGroupJournal is an XML binding class for storage/xslt/schemas/journal-def-jixb.xml file. At the binary distribution of hipergate NewsGroupJournal is preprocessed with JiBX by executing from the command line:

```
java -cp bcel.jar;jibx-bind-1.1.5.jar;jibx-extras.jar;xpp3.jar
org.jibx.binding.Compile journal-def-jixb.xml
```

How the posts of a NewsGroup will be written in XHTML is defined at an XML file stored at `tx_journal` column of `k_newsgroups` table. This XML file stored as a CLOB is like:

```
<journal guid="guid_of_newsgroup" language="en">
  <basehref>http://localhost:8000/hipergate</basehref>
  <blogpath>C:\PROGRA~1\Tomcat\webapps\hipergate</blogpath>
  <outputpath>C:\PROGRA~1\Tomcat\webapps\hipergate</outputpath>
  <templates>
    <template name="blog" filter="main">
           <inputfile>xsl\blog.xsl</inputfile>
    </template>
```

```
    <template name="month" filter="monthly">
           <inputfile>xsl\month.xsl</inputfile>
    </template>
    <template name="day" filter="daily">
           <inputfile>xsl\day.xsl</inputfile>
    </template>
    <template name="one" filter="single">
           <inputfile>xsl\single.xsl</inputfile>
    </template>
    <template name="tag" filter="bytag">
           <inputfile>xsl\bytag.xsl</inputfile>
    </template>
    <template name="rss" filter="rss2">
           <inputfile>xsl\rss2.xsl</inputfile>
    </template>
  </templates>
</journal>
```

This file describes that the NewsGroup will be written as static XHTML to output directory `C:\PROGRA~1\Tomcat\webapps\hipergate\archive` using six different XSL stylesheets one for each page filtering type. These templates must be at `C:\PROGRA~1\Tomcat\webapps\hipergate\xsl`

There are also seven predefined page filtering types :

1. main
2. monthly
3. daily
4. single
5. bytag
6. rss2

The XSL stylesheets for each page filtering type work on XML data files like this one.

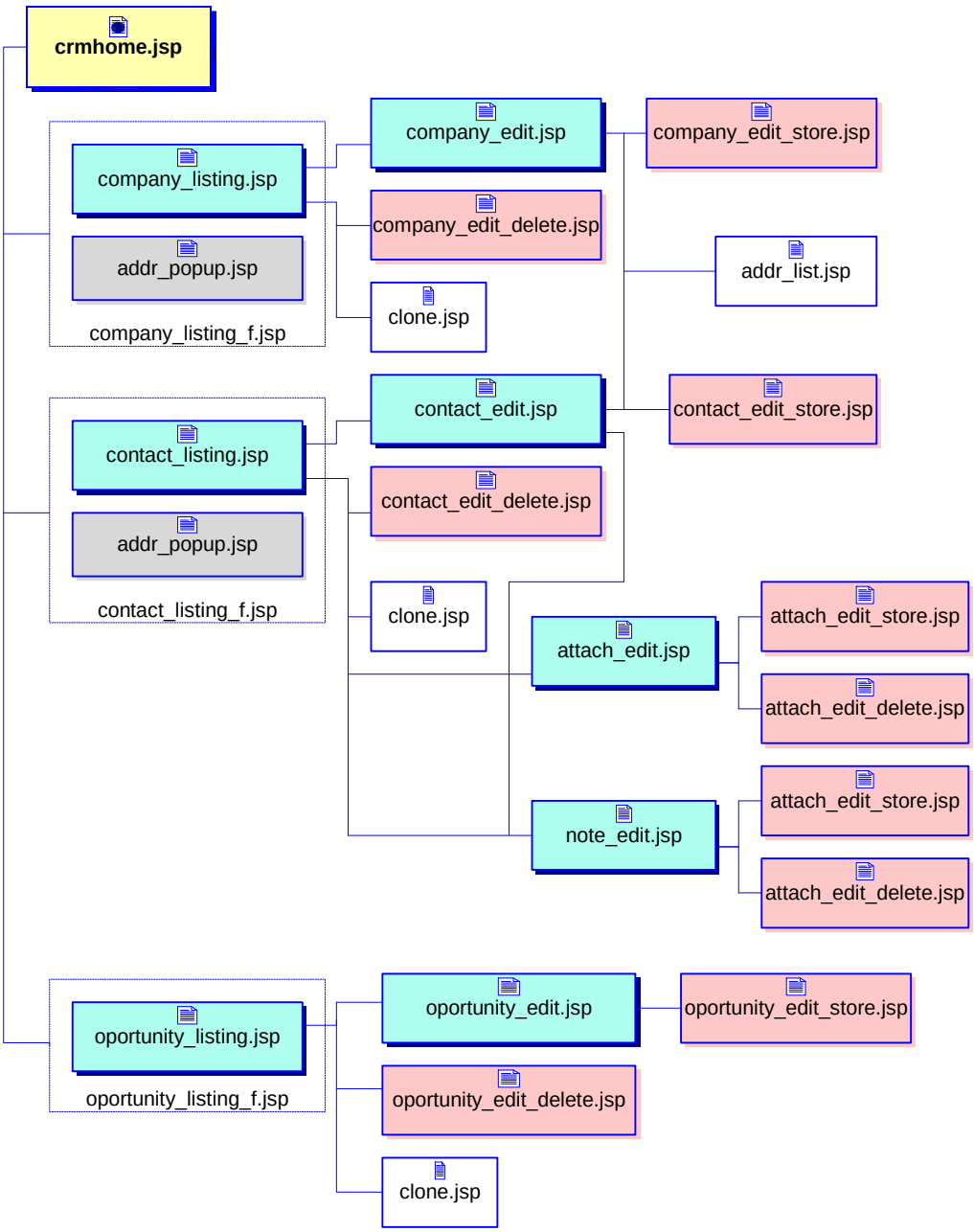
```
<?xml version="1.0" encoding="UTF-8"?>
<Journal guid="c1a8032213193241336100021ba07890">
  <NewsMessages offset="0" eof="true" count="2">
    <NewsMessage>
      <gu_category>c0a8012213193241336100021ba0a3b1</gu_category>
      <gu_msg>c0a80122131952ef1db10002ce29fad4</gu_msg>
      <gu_product>c0a80122131952ef02110002beaece84</gu_product>
      <nm_author><![CDATA[Administrator DEMO]]></nm_author>
      <tx_subject><![CDATA[Subject of sample Post #1]]></tx_subject>
      <dt_published>2011-08-04T12:00:00</dt_published>
      <tx_email><![CDATA[administrator@hipergate-demo.com]]></tx_email>
      <nu_thread_msgs>1</nu_thread_msgs>
      <gu_thread_msg>c0a80122131952ef1db10002ce29fad4</gu_thread_msg>
      <gu_parent_msg></gu_parent_msg>
      <nu_votes></nu_votes>
       <tx_permalink><![CDATA[c0a80122131952ef1db10002ce29fad4]]></tx_permalink>
      <tx_msg><![CDATA[Text of sample post #1]]></tx_msg>
      <NewsMessageTag></newsmessagetags></NewsMessageTag>
    </NewsMessage>
    <NewsMessage>
      <gu_category>c0a8012213193241336100021ba0a3b1</gu_category>
      <gu_msg>c0a801221319544316610003 7dc96b03</gu_msg>
      <gu_product>c0a801221319544314310003 6dd642a9</gu_product>
      <nm_author><![CDATA[Administrator DEMO]]></nm_author>
      <tx_subject><![CDATA[Subject of sample post #2]]></tx_subject>
```

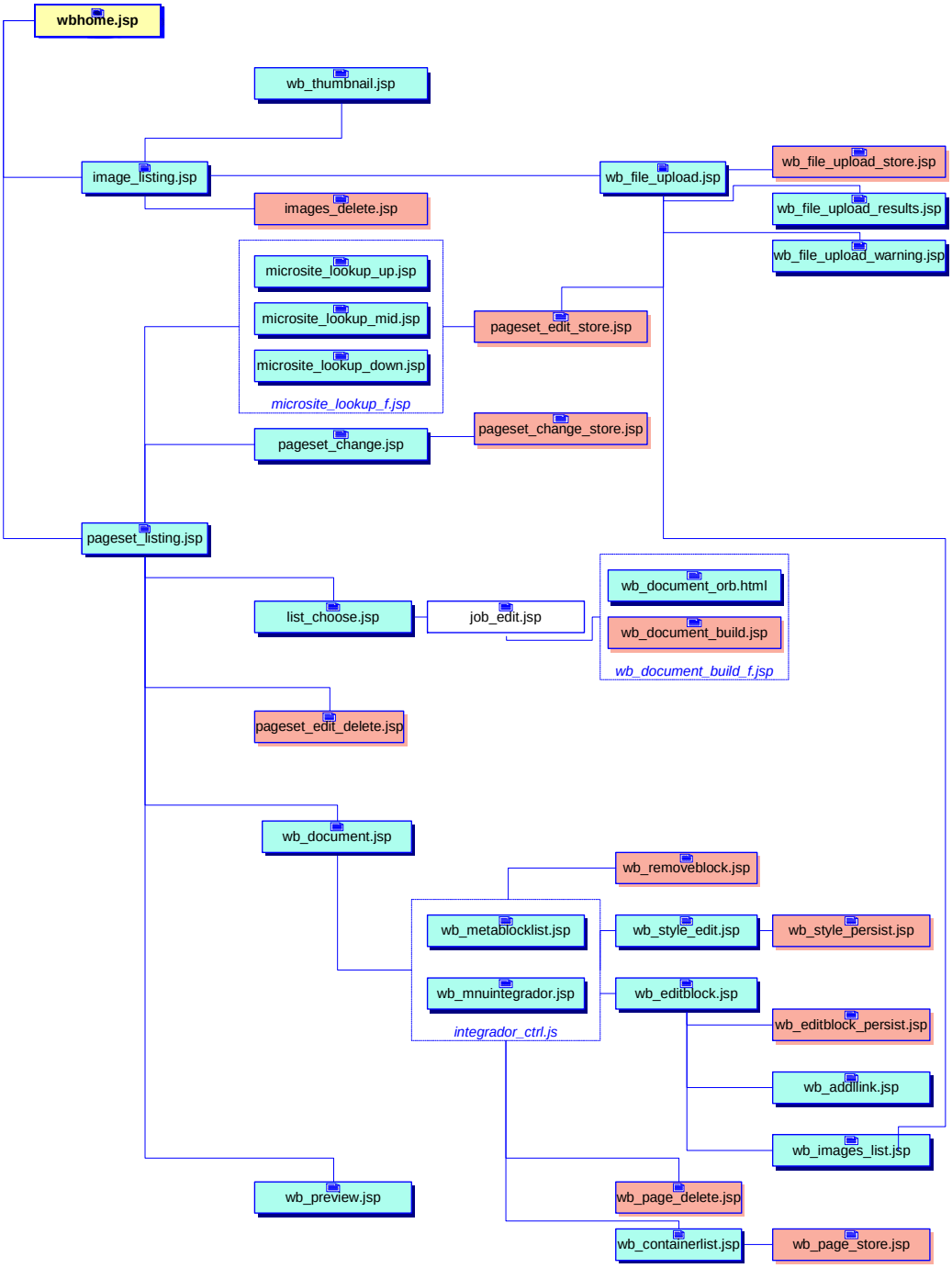
```
      <dt_published>2011-08-04T12:00:00</dt_published>
      <tx_email><![CDATA[administrator@hipergate-demo.com]]></tx_email>
      <nu_thread_msgs>1</nu_thread_msgs>
      <gu_thread_msg>c0a801221319544316610003 7dc96b03</gu_thread_msg>
      <gu_parent_msg></gu_parent_msg>
      <nu_votes></nu_votes>
      <tx_permalink><![CDATA[c0a801221319544316610003 7dc96b03]]></tx_permalink>
      <tx_msg><![CDATA[Text of sample post #2]]></tx_msg>
      <NewsMessageTag>
        <newsmessagetags>
          <newsmessagetag>
            <gu_tag>c0a801221319537a674100030cf9d0cf</gu_tag>
            <gu_newsgrp>c0a801221319324133610002 1ba0a3b1</gu_newsgrp>
            <dt_created>2011-08-04T04:34:37</dt_created>
            <tl_tag><![CDATA[hashtagtext1]]></tl_tag>
            <de_tag></de_tag>
            <nu_msgs>1</nu_msgs>
            <bo_incoming_ping></bo_incoming_ping>
            <dt_trackback></dt_trackback>
            <url_trackback></url_trackback>
            <od_tag></od_tag>
          </newsmessagetag>
          <newsmessagetag>
            <gu_tag>c0a801221319537bd7710003191da17c</gu_tag>
            <gu_newsgrp>c0a801221319324133610002 1ba0a3b1</gu_newsgrp>
            <dt_created>2011-08-04T04:34:43</dt_created>
            <tl_tag><![CDATA[hasghtagtext2]]></tl_tag>
            <de_tag></de_tag>
            <nu_msgs></nu_msgs>
            <bo_incoming_ping></bo_incoming_ping>
            <dt_trackback></dt_trackback>
            <url_trackback></url_trackback>
            <od_tag></od_tag>
          </newsmessagetag>
        </newsmessagetags>
      </NewsMessageTag>
    </NewsMessage>
  </NewsMessages>
</Journal>
```


For creating a static HTML copy of the Newsgroup an instance of NewsGroupJournal must bre created firt by calling getJournal() method of NewsGroup class and then excute rebuild() method from the NewsGroupJournal instance with a piece of code like:

```
DBBind oDBB = new DBBind();
JDCConnection oCon = oDBB.getConnection("rebuild_forum");
NewsGroup  oForumsGrp = new NewsGroup(oCon, "guid_of_newsgroup");
NewsGroupJournal oJour = oForumsGrp.getJournal();
oJour.rebuild(oCon, true);
oCon.close("rebuild_forum");
oDBB.close();
```

Customers Relationships Management Module





Deferred file deletion

When a PageSet is deleted, it is not always possible to immediately delete the associated files. In some cases the web server may leave blocked some files until it is restarted. As a workaround for this problem, pageset_edit_store.jsp creates a list of files pending of being deleted at file shell/cleanup.txt. One option is writing a shell script that purges these files each time the web server is restarted.

7 JavaScripts

Third party libraries

DynAPI	http://dynapi.sourceforge.net/dynapi/
FCKEditor	http://www.fckeditor.net/
HTMLArea	http://www.interactivetools.com/

Conventions

Skins and style sheets (CSS)

The application look'n feel is determined by a skin set at `styles.css` files from directories `/web/skins/...`

The default skin is that from subdirectory `/xp`.

The selected look'n feel is kept at a cookie called `skin` and is set for the first time at page `/common/login_chk.jsp`.

The skin must be set at each page by including these JavaScripts :

```
<SCRIPT LANGUAGE="JavaScript" SRC="/javascript/cookies.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript" SRC="/javascript/setskin.js"></SCRIPT>
```

Dates

By convention short dates are always writen in format YYYY-MM-DD no matter what language is used.

Calendar

There is a reusable common calendar at page `/common/calendar.jsp`.

The JavaScript function for opening the calendar in a popup is :

```
function showCalendar (ctrl) {
  var dtnw = new Date();

  // m -> Mes [0..11]
  // a -> Año [0..] (0≡1900, 100≡2000, 101≡2000)

  window.open ("../common/calendar.jsp?a=" + (dtnw.getYear()) +
  "&m=" + dtnw.getMonth() + "&c=" + ctrl, "",
  "toolbar=no,directories=no,menubar=no,resizable=no,width=171,
  height=195"); }
```

where

`ctrl`: Object of type HTML `<INPUT>` where the selectd date will be returned..

JavaScript Libraries

Read and Write Cookies

Location: File /javascripts/cookies.js

`function` **`getCookie`** `(name)`

Return Value: Cookie contents without escape characters or **`null`** if no cookie is found with such name.

`function` **`setCookie`** `(name, value, expire)`

Set value of a cookie.

`name`	Cookie name.
`value`	Internally the JavaScript escape() function will be applied to input value.

`expire` Optional. JavaScript Date object. Cookie absolute expiration date.

`function` **`deleteCookie`** `(name)`

Delete a cookie by making it expire.

Combobox management

Location: File /javascripts/combobox.js

`function` **`setCombo`** `(objCombo, idValue)`

Move selected index to `idValue`.

`objCombo` HTML <SELECT> object.

`idValue` Value to be searched. Each combobox value is compared to `idValue` until one is found or end of select options is reached.

`function` **`comboIndexOf`** `(objCombo, idValue)`

Get the index of a value in a combo.

`objCombo` HTML <SELECT> object.

`idValue` Value to be searched. Each combobox value is compared to `idValue` until one is found or end of select options is reached.

Return Value: Index [0..objCombo.options.length-1] or –1 if `idValue` was not found.

`function` **`comboPush`** `(objCombo,txValue,idValue,defSel, curSel)`

Add an option to a ComboBox.

`objCombo` HTML <SELECT> object.

`txValue` Displayed text of Option.

`idValue` Option internal Value.

`defSel` true if option shall be selected by default.

`curSel` true if option must become current selection.

`function` **`getCombo`** `(objCombo)`

Get the first value selected at a ComboBox.

`objCombo` HTML <SELECT> object.

Return Value: VALUE attribute of currently selected option or **`null`** if there is no selected option.

`function` **`getComboText`** `(objCombo)`

Get the first text selected at a ComboBox.

`objCombo` HTML <SELECT> object.

Return Value: Text of currently selected option or **`null`** if there is no selected option.

`function` **`clearCombo`** `(objCombo)`

Remove all options from a ComboBox.

`objCombo` HTML <SELECT> object.

`function **sortCombo** (objCombo)`

Sort ComboBox in ascending order

Date Validation

Location: File /javascripts/datefuncs.js

`function **getLastDay** (month, year)`

Last day of a month.

`month`	Month [0..11].
`year`	Year (4 digits).

`function **isDate** (dtexpr, dtformat)`

Verifies if a String is formated as a Date.

`dtexpr`	String to be checked.
`dtformat`	Format Code. Currently only "d" is accepted for short dates formated "YYYY-MM-DD" (month [1..12]). Both syntax and last day of month is verified.

`function **parseDate** (dtexpr, dtformat)`

Create a JavaScript Date object from a String.

`dtexpr`	String.
`dtformat`	Format Code. Currently only "d" is accepted for short dates formated "YYYY-MM-DD" (month [1..12]).

Return Value: JavaScriot Date object or `null` if dtexpr does not represent a valid date.

---

e-mail addreses validation

Location: File /javascripts/email.js

`function` **`check_email`** `(email)`

Verify is an e-amil address is syntactically valid

Find substrings inside an HTML page

Location: File /javascripts/findit.js

`function` **`findit`** `(sValue)`

Find a substring inside current page.

Get URL parameters

Location: File /javascripts/getparam.js

`function` **`getURLParam`** `(name,target)`

Get a parameter from page URL.

`name`	Parameter name.
`target`	Optional. Object of type window where parameter is to be searched.

Return Value: String with value for parameter or `null` if no parameter was found with given name.

String manipulation

Location: File /javascripts/trim.js

`function ltrim (str)`

Remove blank spaces on the left.

Return Value: Trimmed string.

`function rtrim (str)`

Remove blank spaces on the right.

Return Value: Trimmed string.

Bank Accounts Validation

Location: File /javascripts/simplevalidations.js

`function isBankAccount (entity,office,dc,cc)`

Verifies control digits for a bank account.

Credit card validations

Location: Archivo /javascripts/creditcards.js

Eric Krock
(c) 1997 Netscape Communications Corp.

8 Creating the database

The hipergate database can be created in two ways: 1st) loading a native database dump –usually provided for each supported DBMS with the standard distribution- 2nd) using class `com.knowgate.hipergate.datamodel.ModelManager` that contains routines for launching the whole SQL-DDL creation script using JDBC.

`ModelManager` is specially designed for being invoked from the command line although it is also callable programmatically from Java code.

Steps in the initial creation of the database

For creating a database 4 things must be done :

1º) Create table views and stored procedures.

2º) Load domains SYSTEM and MODEL, required for staring up the application.

3º) Load additional domains (typically TEST, DEMO and REAL).

Portable SQL scripts

hipergate can create its database from scratch using just SQL scripts.

These scripts can be found uncompressed at directory `com/knowgate/hipergate/datamodel` or in the same package inside hipergate.jar.

Scripts can have extension SQL or DDL. The only difference is the command delimiter used either semicolon ";" for SQL or "GO;" for DDL.

Scripts are divided according to their type :

- tables
- indexes

- constraints
- views
- data
- procedures
- triggers
- drop

Scripts for stored procedures, triggers, views and drops are dependant of the RDBMS.

For achieving database independency only a limited subset of SQL types is used. A special database independent name is given to each type and `ModelManager` translated this name on the fly to the native type when launching the script.

hipergate typename	Oracle	SQL Server	PostgreSQL
CURRENT_TIMESTAMP	SYSDATE	GETDATE()	CURRENT_TIMESTAMP
DATETIME	DATE	DATETIME	TIMESTAMP
LONGVARCHAR	LONG	TEXT	TEXT
LONGVARBINARY	LONG RAW	IMAGE	BYTEA
FLOAT	NUMBER	FLOAT	FLOAT
INTEGER	NUMBER(11)	INTEGER	INTEGER
SMALLINT	NUMBER(6)	SMALLINT	SMALLINT
SERIAL	NUMBER(11)	INTEGER IDENTITY	SERIAL

Command for creating and dropping a database

Create default database

From the command line write

```
java com.knowgate.hipergate.datamodel.ModelManager
/etc/hipergate.cnf create database verbose
```

This will created tables for all modules as well as domains SYSTEM, MODEL, TEST, DEMO y REAL.

Create minimal database

From the command line write

```
java com.knowgate.hipergate.datamodel.ModelManager
/etc/hipergate.cnf create all verbose
```

This will created tables for all modules as well as domains SYSTEM and MODEL.

Drop database

From the command line write

```
java com.knowgate.hipergate.datamodel.ModelManager
/etc/hipergate.cnf drop all verbose
```

This will drop all hipergate objects from the database. Object not belonging to hipergate will be preserved.

How to execute a custom script against database

From the command line write

```
java com.knowgate.hipergate.datamodel.ModelManager
/etc/hipergate.cnf execute /tmp/script.sql verbose
```

How to generate a SQL INSERT script for a table

hipergate has a method for creating a SQL INSERT script for all the data in a given table. From the command line type:

```
java com.knowgate.hipergate.datamodel.ModelManager
/etc/hipergate.cnf script table_name /tmp/output_file.sql
```

It may also be done with a JSP page like:

```
<%@ page language="java"
import="com.knowgate.hipergate.datamodel.ModelManager"
session="false" contentType="text/html;charset=ISO-8859-1"
%><%

ModelManager.main (new String[]{"/etc/test.cnf", "script",
"table_name", "/tmp/output_file.sql"});

%><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;
charset=iso-8859-1" />
    <title>Dump to SQL script</title>
```

```
  </head>
  <body>
  Volcado finalizado con &eacute;xito.
  </body>
</html>
```

How to load a text file into a table from the command line

You can use class `com.knowgate.hipergate.datamodel.TableLoader` for loading a delimited text file into a table.

TableLoader is fast and simple but has some restrictions:

1. The text file columns must be delimited by tabulators.
2. The text file rows must be delimited by an end of line.
3. The text file columns must exactly match the number of columns of the target table.
4. Dates must be in format yyyy-MM-dd HH:mm:ss
5. Floating point numbers must use a dot as decimal delimiter.

TableLoader can be invoked from `com.knowgate.hipergate.datamodel.ModelManager.main()` method which takes the following command line parameters:

- **properties file path**: Usually `/etc/hipergate.cnf` or C:\Windows\hipergate.cnf
- **command**: for loading a text file into a table must be `bulkload`
- **target table**: fully qualified name of target table where the text file is to be loaded.
- **text file character encoding**: must be any of the Java Supported Character Encodings.
- **verbose**: Optional. If verbose parameter is specified the additional progress information is shown at system standard output.

Usage example (for Linux):

```
java -cp /opt/tomcat/webapps/hipergate/WEB-
INF/classes:/opt/tomcat/webapps/hipergate/WEB-INF/lib/bsh-
2.0b4.jar:/opt/tomcat/webapps/hipergate/WEB-INF/lib/jakarta-
oro-2.0.8.jar com.knowgate.hipergate.datamodel.ModelManager
/etc/hipergate.cnf bulkload k_target_table
/tmp/Source_File.txt UTF-16LE verbose
```

☞ You must also add the .jar reference for the JDBC driver of your RDBMS at the –cp parameter of the above example.

☞ See also loading data from a delimited text file section for more complex forms of external data loading.

9 Integration witn Jakarta Lucene

Lucene is the open source indexer from the Jakarta project.
Lucene is a fast generic purpose multi-platform indexer.
http://jakarta.apache.org/lucene/docs/index.html

Example of integration for incidents and forums

An example of how creating Lucene full-text indexes for `k_bugs` and `k_newsgroups` tables can be found class `com.knowgate.lucene.Indexer`.

The key point to be understood is how LONGVARCHAR columns are indexed and how GUIDs are retrieved for registers matching searched text.

See the JavaDoc API for more information.

10 Integration with Jakarta POI

Jakarta POI is the interface for accessing OLE2 documents from Apache Software Foundation. POI read and write compound OLE2 documents from 100% pure Java.

A simple interface for reading and writing OLE2 properties can be found at class `com.knowgate.ole.OLEDocument`.
This class is used from page `docedit_store.jsp` for filling automatically columns of table `k_prod_attr` with properties read from Ole2 documents.

11 Integration with LDAP

hipergate can read user password from an LDAP. It is also possible to synchronize contacts between hipergate database and an LDAP directory and access these contacts from a heavy mail client such as Outlook Express.

How to create an LDAP directory for hipergate

The LDAP structure exported by hipergate tries to be as simple as possible. Its main objective is allowing access from e-mail clients such as Outlook Express, Mozilla Mail or Ximian Evolution. This structure is the foundation for Corporate Directories like Active Directory, OpenLDAP, Novell NDS, etc.

The LDAP interface has been developed using Novell Directory Server SDK (http://developer.novell.com/ndk/), which is a freely redistributable. This API supports LDAP v3.

hipergate 2.1 has been tested with OpenLDAP. The default OpenLDAP configuration is enough for a standard environment with few restrictive security policy.

Quick example of OpenLDAP configuration

1) Install OpenLDAP 2.1. You may build from source or use RPM, DEB, PKG, etc.

2) Add or modify the following parameters at file slapd.conf (located at directory /etc/ or /etc/openldap)

```
suffix          "dc=hipergate,dc=org"
rootdn          "cn=Manager,dc=hipergate,dc=org"

# Cleartext passwords, especially for the rootdn, should
# be avoided.  See slappasswd(8) and slapd.conf(5) for details.
rootpw          happyHacking

access to attr=userPassword
       by anonymous auth
       by dn.base="cn=Manager,dc=hipergate,dc=org" write
       by * none

access to dn.subtree="dc=hipergate,dc=org"
       by dn.base="cn=Manager,dc=hipergate,dc=org" write
       by users read
```

3) Start-up OpenLDAP service. Check that it is listening at port 389 useing the `netstat` command, for example:

```
$ netstat -na | fgrep ":389"
tcp    0    0    0.0.0.0:389        0.0.0.0:*        LISTEN
```

4) Create the following text file and save it as `init.ldif`:

```
dn: dc=hipergate,dc=org
objectclass: dcObject
objectclass: organization
o: The hipergate working group
dc: hipergate

dn: cn=Manager,dc=hipergate,dc=org
objectclass: organizationalRole
cn: Manager
```

☞ Trim all trailing spaces and blank lines from `init.ldif`.

5) Load to LDAP the file just created :

```
ldapadd -x -D "cn=Manager,dc=hipergate,dc=org" -W -f init.ldif
```

Use the password from parameter "rootpw" of file `slapd.conf`

What information is stored at LDAP

The relational model of hipergate contains information about users, contacts and company employees (fellows). This entities are the ones exported to LDAP.

The information is stored at LDAP with the following structure:

```
dc=org
`-- dc=hipergate
    `--dc=k_domain.nm_domain
       `-- dc=k_workareas.nm_workarea
           |-- dc=users
           |   `-- cn=k_users.tx_main_email
           |       `-- dc=privateContacts
           |           `-- cn=k_member_address.tx_email
           |-- dc=publicContacts
           |   `-- cn=k_member_address.tx_email
           `-- dc=employees
               `-- cn=k_fellos.tx_email
```

Each **dc** element is a container. Some containers have a fixed name, (such as org, hipergate, users, privateContacts, publicContacts and employees) imposed by hipergate LDAP API. The fields in bold are information taken from hipergate relational data model.

Branch `hipergate` contains one entry for each domain from `k_domains` table. Each domain contains its Workareas like the relational model does.

Each Workarea has three containers:

- **`users`**: Users from table `k_users`. They have a password but do not keep postal address nor telephone information.
- **`publicContacts`**: Entries from `k_member_address` table which `bo_private` field is zero (public contacts). This entities have postal address and telephone information.
- **`employees`**: Entries from `k_fellows` table. Their postal address is made by their division, department and location.

Each entry from `users` branch can hold a subcontainer called `privateContacts` that contains entries from `k_member_address` which `bo_private` field is not zero (private contacts). For knowing what contact of hipergate corresponds to each LDAP entry the main e-mail `tx_main_email` from its address is used. If a contact has several addresses he will be present several times at LDAP as different entries.

Objects Person and Address created by hipergate are of the simplest LDAP format, compatible with Outlook Express, WAB (Windows Address Book), Mozilla and Ximian Evolution.

These objects are based on LDAP object classes (*objectClass*) `inetOrgPerson` and `organizationalPerson`. Fields loaded by default are :

cn **Common Name**

Main e-mail address

```
Users:     k_users.tx_main_email
Employees: k_fellows.tx_email
Contacts:  k_member_address.tx_email
```

uid **Unique ID**

hipergate Global Unique Identifier for entity

```
Users:     k_users.gu_user
Employees: k_fellows.gu_fellow
Contacts:  k_member_address.gu_address
```

givenName

```
Users:     k_users.nm_user
Employees: k_fellows.tx_name
Contacts:  k_member_address.tx_name
```

sn **Surname**

```
Users:     (k_users.tx_surname1 + ' ' +
```

```
                    k_users.tx_surname2), o bién
                   k_users.tx_nickname
Employees:         k_fellows.tx_name
Contacts:          k_member_address.tx_name
```

userPassword

User password (user entries only)

```
Users:     k_users.pwd
```

displayName

Used at the "Display" field of Outlook Express/WAB.

```
Users:     (k_users.nm_user     + ' ' +
            k_users.tx_surname1 + ' ' +
            k_users.tx_surname2), ó bien
           tx_nickname
Employees: k_fellows.tx_name
Contacts:  k_member_address.tx_name
```

mail

Idéntico al campo **cn**, se utiliza para Outlook Express/WAB.

```
Users:     k_users.tx_main_email
Employees: k_fellows.tx_email
Contacts:  k_member_address.tx_email
```

o **Organization**

Name of Company for Contact

```
Users:     k_users.nm_company
Employees: k_fellows.tx_company
Contacts:  k_member_address.nm_legal
```

telephonenumber

Main phone. Not available for users.

```
Employees: k_fellows.work_phone
Contacts:  k_member_address.work_phone
```

homePhone

Home phone. Not available for users.

```
Employees: k_fellows.home_phone
Contacts:  k_member_address.home_phone
```

mobile

Mobile phone. Not available for users.

```
Employees: k_fellows.mov_phone
Contacts:  k_member_address.mov_phone
```

facsimileTelephoneNumber

Fax phone. Only available for contacts.

```
Contacts:  k_member_address.fax_phone
```

postalAddress

Postal Address. Carriage returns and Line Feeds are coded as *pipes* (ASCII 166), following the convention from Outlook Express/WAB.

```
Employees: (k_fellows.tx_dept     + '|' +
            k_fellows.tx_division + '|' +
            k_fellows.tx_location)
Contacts:  (k_member_address.tp_street + ' ' +
            k_member_address.nm_street + ' ' +
            k_member_address.nu_street + '|' +
            k_member_address.tx_addr1  + '|' +
            k_member_address.tx_addr2)
```

l **Locality**

Postal address city.

```
Contacts:  k_member_address.nm_city
```

st **State**

Postal address state.

```
Contacts:  k_member_address.nm_state, ó bien
           k_member_address.id_state
```

postalCode

```
Contacts:  k_member_address.zipcode
```

How to connect hipergate with an LDAP directory

Once the directory has been created it is neccesary to set the following connection parameters at hipergate.cnf.

`ldapconnect` : Directory URL.
Must be of the form:
`ldap://192.168.1.1:389/dc=hipergate,dc=org`

`ldapuser` : `cn=Manager,dc=hipergate,dc=org`

`ldappassword` : `manager`

`ldapclass` : Class that implements interface `com.knowgate.ldap.LDAPModel`. By default is `com.knowgate.ldap.LDAPNovell` but an alternative one can be written.

Synchronizing hipergate and LDAP

When LDAP connection parameters are set at hipergate.cnf, hipergate automatically synchronizes users, contacts and fellows with LDAP.

hipergate automatically synchronizes users, contacts and fellows from the relational database to the LDAP directory. When a user, contact or fellow is added, modified or deleted in hipergate the changes are reflected in the LDAP directory.

Synchronization is done by JSP pages: addr_edit_store.jsp, addr_edit_delete.jsp, contact_new_store.jsp, usernew_store.jsp, useredit_modify.jsp, fellow_edit_store.jsp, felow_edit_delete.jsp

If hipergate database and LDAP lose their synchronization it is possible to write single entries using methods : `addOrReplaceAddress()`, `addOrReplaceUser()`, `deleteAddress()` and `deleteUser()`.

How to load a full Domain or Workarea into LDAP

LDAPModel interface has a couple of methods for loading domains and Workareas into LDAP:

`LDAPModel.`**`loadDomain`**` (Connection oJdbc, int iDomainId)`
Load all users, contacts and fellows from a Domain into an LDAP directory.
The first parameter is a JDBC Connection object.
The second parameter is the numeric identifier of the domain to be loaded.

`LDAPModel.`**`loadWorkArea`**` (Connection oJdbc, String sDomainNm, String sWorkAreaNm)`
Load all users, contacts and fellows from a Workarea into an LDAP directory.
The second parameter is the name of the domain to be loaded (`k_domains.nm_domain`).
The third parameter is the name of the Workarea to be loaded (`k_workareas.nm_workarea`)

How to delete a Workarea

Use method `LDAPModel.deleteWorkArea()` for deleteing from LDAP all entries belonging to a given hipergate Workarea.

How to delete the full directory

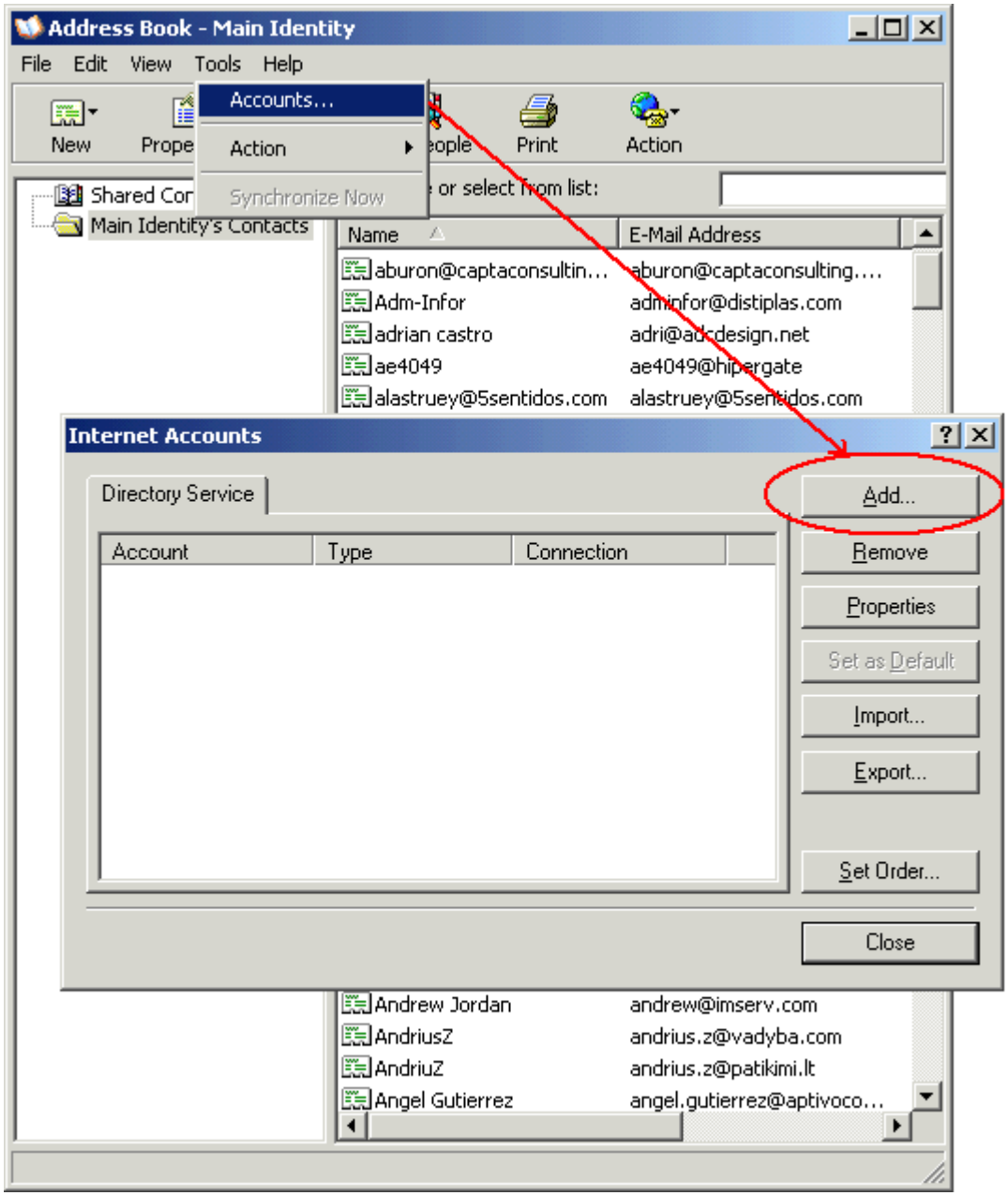
Use method `LDAPModel.dropAll()` for deleting all hipergate entries from an LDAP directory.

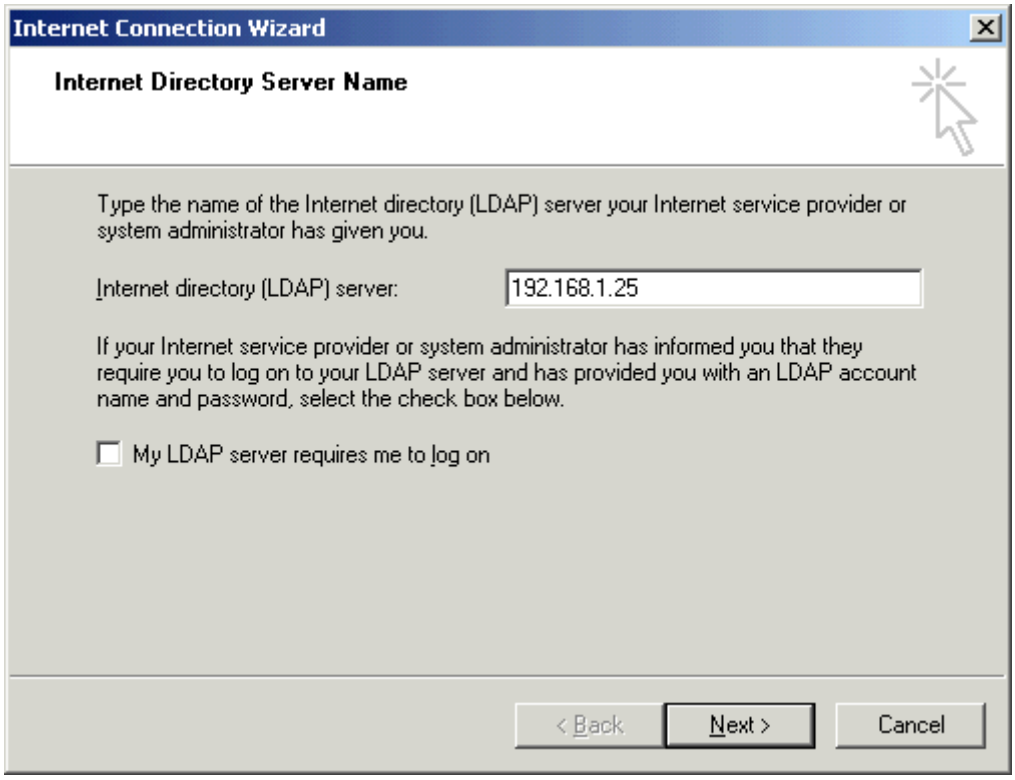
How to access LDAP from Outlook Express

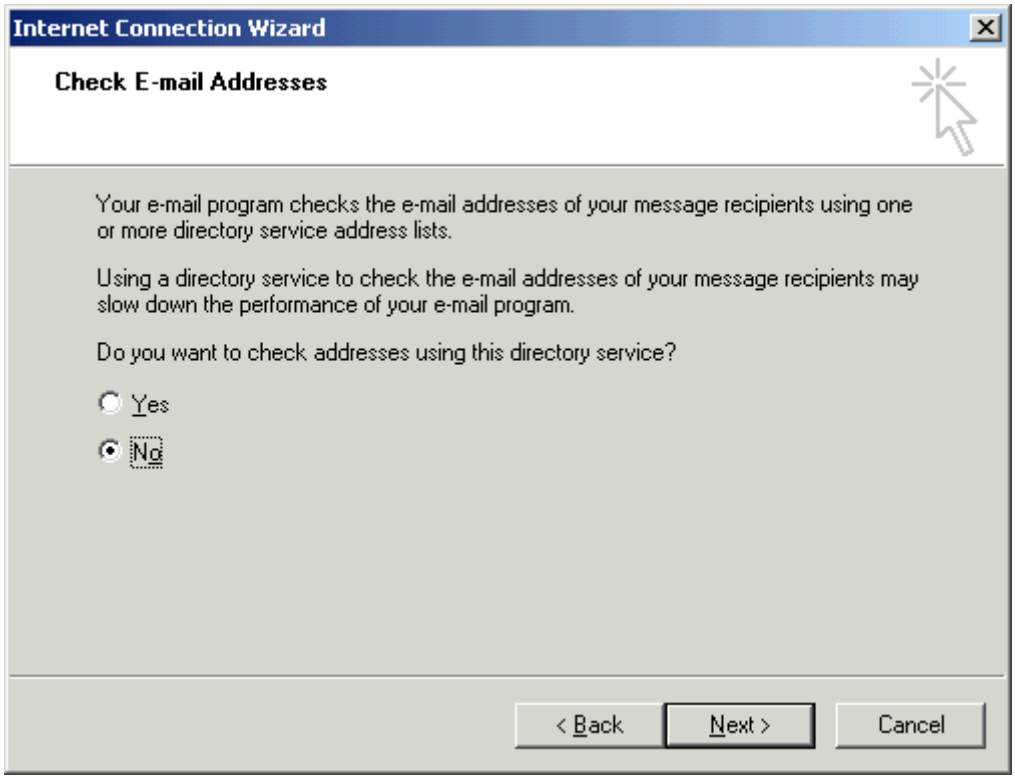
Configuring an LDAP client (Outlook Express, Evolution, Mozilla, etc) depends more on how the LDAP server is configured than on the hipergate data itself.

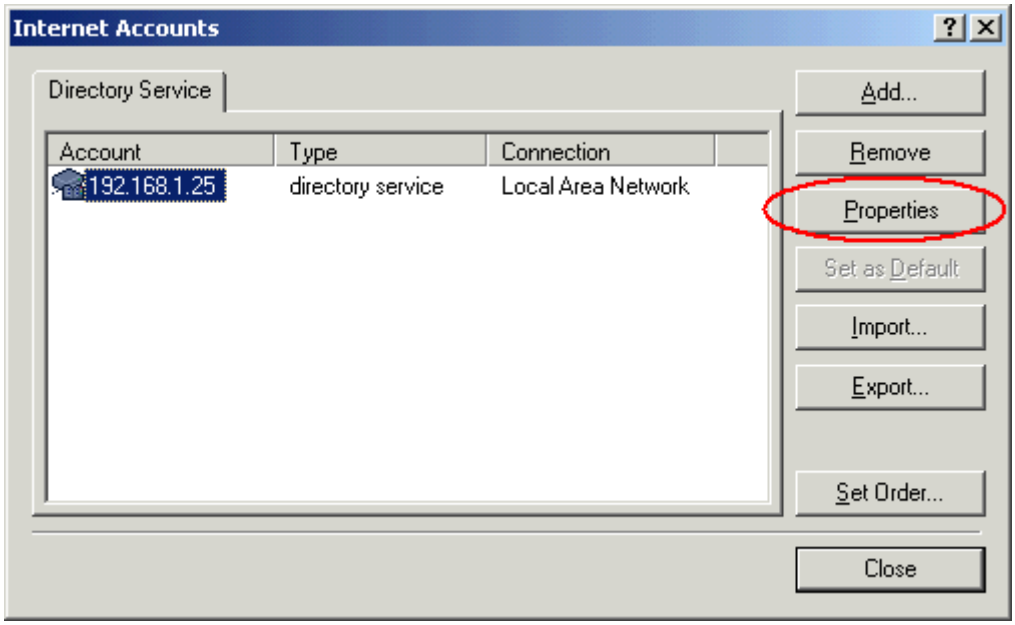
First it is neccesary to stablish the users and passwords that can authenticate against a domain for performing queries. The default OpenLDAP setup only allows not authenticated bindings.
Once the user has impersonated, the security rules are usually very few, all users are allowed to query the entire directory no matter who they are.

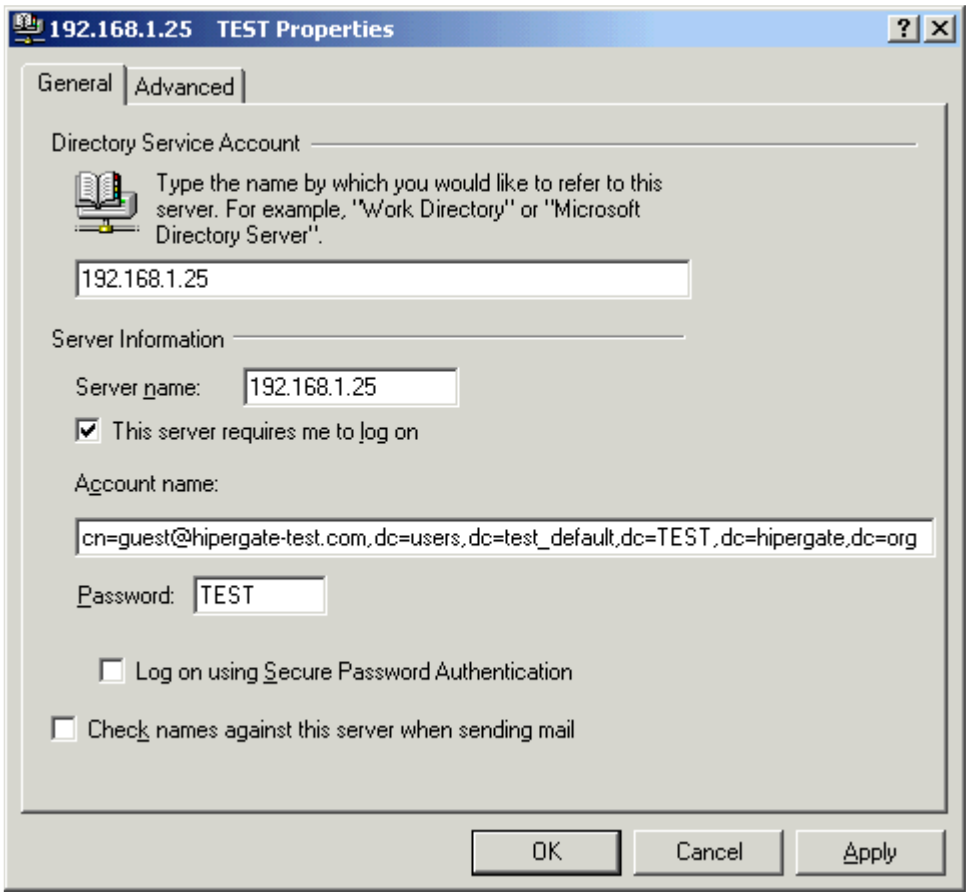
Outlook Express screenshots









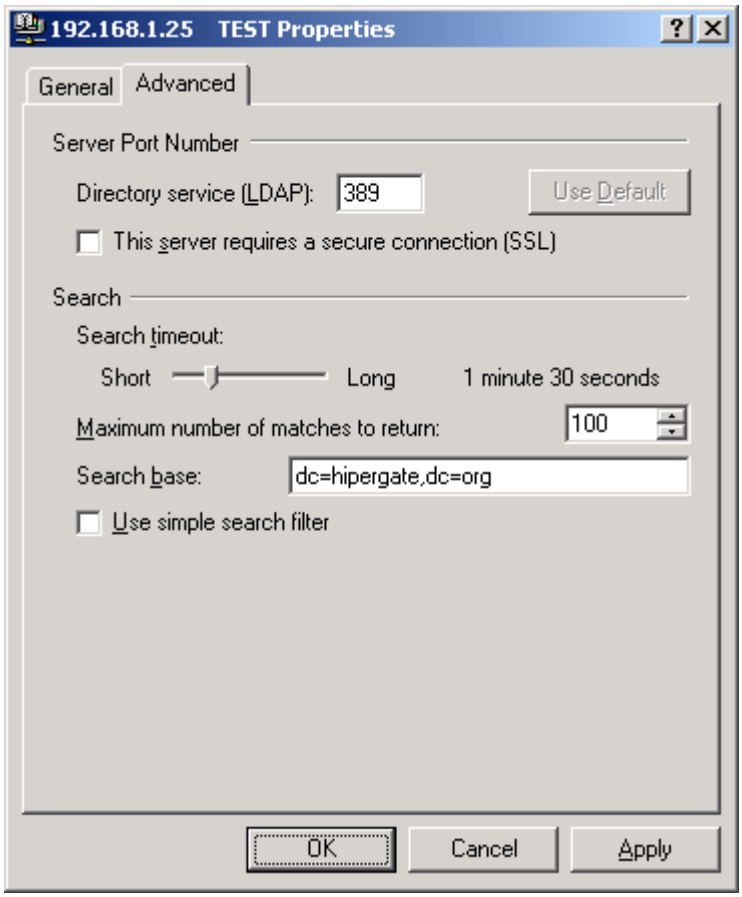


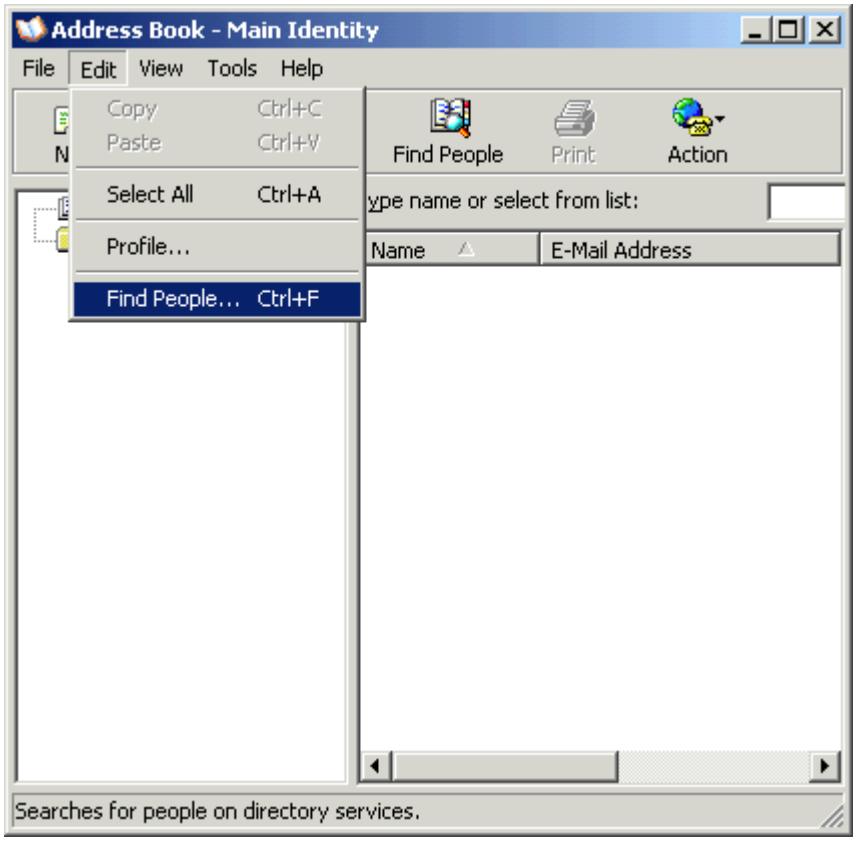
At the configuration form of the Directory Service of the mail client it is neccesary to specify the host address and port (389 by default). Moreover a valid user/password for the service must be used. When loading from

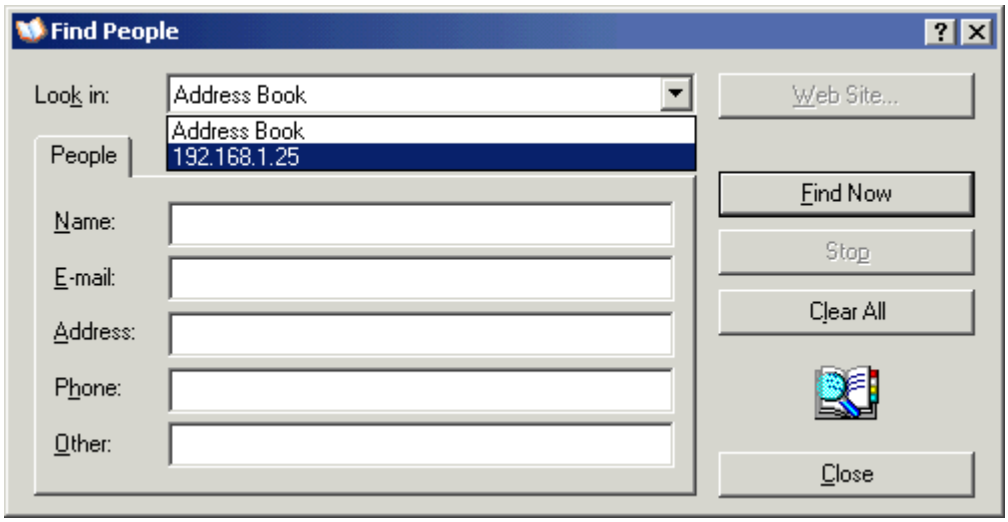
hipergate, all entries from `k_users` table have the field `user Password`, so any user from this table can bind to the LDAP directory.

For setting a user/password the full LDAP path must be added to user Distinguished Name. For example:

cn=user@hipergate.org,dc=users,dc=workarea1,dc=domain1,dc=hipergate,dc=org







By using rewriting rules it is possible that Open LDAP searches an e-mail address across all domains. For more information read this thread from Buchan Milne:
http://www.openldap.org/lists/openldap-software/200404/msg00910.html

User authentication based on LDAP

It is possible to configure hipergate so that user passwords for accessing the application are read from an LDAP directory and not from `k_users` table.

LDAP can only check password, it is not a replacement for hipergate native security model. Even if LDAP is used for verifying password the user accounts must still be created and maintained within hipergate native database.

How to authenticate users with LDAP

The following steps are required:

1. The LDAP schema must be created. Password is checked against `user Password` field of `cn=`*`user@domain.com`*`,dc=users,dc=`*`workarea_name`*`,dc=`*`domain_name`*`,dc=hipergate,dc=org` LDAP entry. For verifying the passwords hipergate tries to bind to LDAP using `LDAPConnection.bind()` supplying the password given by user as a parameter. At the LDAP directory there must be an entry under dc=users... which common name (cn) is the user's e-mail; this entry can be created automatically by hipergate if automatic LDAP synchronization is activated.
2. Set properties `ldapconnect`, `ldapuser`, `ldappassword` y `ldapclass` at `hipergate.cnf` as described previously in this chapter.
3. Set property `authmethod=ldap` at `hipergate.cnf`.

An LDAP connection is only performed once during the logon process. User/Password pair is extracted from LDAP and stored in session cookies which are later checked by the standard security routines of each hipergate page.

12 Single sign on with NTLM

When using Microsoft Internet Explorer, hipergate can authenticate users without requesting for a password using their Windows session credentials.

Integration with NTLM is done using a filter at the servlet container. This filter converts NTLM credentials into HTTP session cookies that are later used by hipergate.

For using NTLM single sign on fields `tx_nickname` and `tx_pwd` from table `k_users` must match the user/password pair used for initiating the Windows session and the user must belong to a Domain which name is the same on Windows than at hipergate database.

How to install the NTLM integration filter

The authentication filter is class `com.knowgate.jcifs.http.NtlmHipergateFilter` located inside `hipergate.jar`.

The filter is installed by adding the following XML fragment to `<web-app>` section of file `/WEB-INF/web.xml`.

```
<filter>
  <filter-name>NtlmHipergateFilter</filter-name>
  <filter-class>
    com.knowgate.jcifs.http.NtlmHipergateFilter
  </filter-class>
  <init-param>
    <param-name>jcifs.http.domainController</param-name>
    <param-value>192.168.1.1</param-value>
  </init-param>
  <init-param>
  <param-name>jcifs.smb.client.logonShare</param-name>
    <param-value>shared_dir_name</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>NtlmHipergateFilter</filter-name>
  <url-pattern>/loginntlm.html</url-pattern>
</filter-mapping>
```

There are two parameters to be configured :

jcifs.http.domainController : IP address of Windows domain controller.

jcifs.smb.client.logonShare : Name of a shared directory at the domain controller.

Page `loginntlm.html` substitutes to `login.html`. When `loginntlm.html` is requested the filter obtains the NTLM credentials and the page redirects to `login_chk.jsp` for validating the Windows session user/password against hipergate database.

Configuring hipergate.cnf

Once the filter is installed, set property `authmethod=ntlm` at `hipergate.cnf` for activating NTLM single sign on.

13 JDBC HTTP Servlet Bridge

From version 3.0, the servlet HttpDataObjsServlet of package com.knowgate.http provides read and write access to the database throught HTTP.

Features

Class HttpDataObjsServlet provides de possibility of accessing the database through HTTP. This feature is useful, for example, for creating Excel sheets capable of updating hipergate data by editing it locally on an offline computer.

Setup

HttpDataObjsServlet is not installed by default.

At section `<web-app>` of file /WEB-INF/web.xml it is necessary to add :

```
<servlet>
  <servlet-name>HttpDataObjsServlet</servlet-name>
  <servlet-class>com.knowgate.http.HttpDataObjsServlet</servlet-class>
  <url-pattern>/servlet/HttpDataObjsServlet</url-pattern>
  <init-param>
    <param-name>profile</param-name>
    <param-value>hipergate</param-value>
  </init-param>
</servlet>
```

and

```
<servlet-mapping>
  <servlet-name>HttpDataObjsServlet</servlet-name>
  <url-pattern>/servlet/HttpDataObjsServlet</url-pattern>
</servlet-mapping>
```

After rebooting the servlet runner (Tomcat), a test can be done by typing the following URL on the web browser :

`http://my_host/servlet/HttpDataObjsServlet?command=ping`

The servlet must return :

```
HttpDataObjsServlet ping OK
```

Input parameters

The servlet must be requested by HTTP POST for writing data and by POST or GET for reading.

Input parameters are the following :

`profile` : Name of configuration file from which database connection properties will be retrieved. This parameter is optional. The default value is "hipergate".

`user` : GUID or e-mail of user from table `k_users` that will be used for reading or writing data. This is NOT the value of property `dbuser` of the configuration file `.cnf` but the unique key for a hipergate user. This parameter is requieredrequired.

`password` : Password for previous user. This parameter is required.

`command` : Must be `query`, `update` o `ping`. This parameter is required.

`class` : Name of a subclass of `com.knowgate.dataobjs.DBPersist` or a class implementing the interface `com.knowgate.hipergate.datamodel.ImportLoader`. This parameter is optional. The default value is `com.knowgate.dataobjs.DBPersist`.

`table` : Name of a table or view from the datamodel. This parameter is required for reading data, and used for writing data only when `class` parameter is omitted.

fields : Names of table columns delimited by commas. This parameter is required for reading data and not used for writing it.

where : Filter clause for table. This parameter is required for reading data and not used for writing it.

maxrows : Maximum number of rows to be retrieved from server on each single query. This parameter is optional for reading data and not used for writing it. The default value is 500.

skip : Count of records to skip before reading the first one on a query. This parameter is optional for reading data and not used for writing it. The default value is 0.

coldelim : Output column delimiter. This parameter is optional for reading data and not used for writing it.

rowdelim : Output row delimiter. This parameter is optional for reading data and not used for writing it.

Reading data

This is an example about how to read data from a VBA client. It gets all the companies from a given Workarea which identifier of sector is not null.

```
Public Const MAX_ROWS = 500
Public Const COL_DELIM = "|"
Public Const ROW_DELIM = ";"
Public Const SERVLET_URL = "http://
demo.hipergate.com/servlet/HttpDataObjsServlet"
Public Const WORKAREA = "Guid_of_the_test_workarea_000001"
Public Const CONNECTION_PARAMETERS =
"profile=hipergate&rowdelim=" + ROW_DELIM + "&coldelim=" +
COL_DELIM + "&maxrows=" & MAX_ROWS & "&skip=0&gu_workarea=" +
WORKAREA + "&user=testwa_administrator&password=user_pwd"

Dim HttpReq As New MSXML.XMLHTTPRequest
With HttpReq
  .Open "POST", SERVLET_URL, False
  .setRequestHeader "Content-Type", "application/x-www-form-
                                    urlencoded"
  .send CONNECTION_PARAMETERS + "&" + "
        command=query&table=k_companies" +
        "&where=gu_workarea%3D'"+WORKAREA +
        "'%20AND%20id_sector%20IS%20NOT%20NULL" +
        "&fields=nm_legal,id_sector,id_status"
  MsgBox .responseText
```

```
End With ' HttpReq
```

Data is get as delimited text, like :

```
ACME|GADGETS|ACTIVE;ASTROTECH|SPACE|ACTIVE;IBM|COMPUTERS|ACTIVE
```

Then this data can be easily processed by using VBA Split() sentences:

```
Dim vRows As Variant
Dim vCols As Variant
Dim r As Long
vRows = Split(HttpReq.responseText, ROW_DELIM, MAX_ROWS)
For r = LBound(vRows) To UBound(vRows)
  vCols = Split(vRows(r), COL_DELIM)
  ' Do whatever here...
Next r
```

Writing data

For writing data, the column names must de passed as parameters of the POST request. If the column is of type numeric or date, then after the name the SQL type must be specified and, for dates, its format mask.

This examples writes at `k_companies` using class `Company` from package `com.knowgate.crm`.

```
Dim HttpReq As New MSXML.XMLHTTPRequest
With HttpReq
  .Open "POST", SERVLET_URL, False
  .setRequestHeader "Content-Type", "application/x-www-form-
urlencoded"
  .send CONNECTION_PARAMETERS + "&" +
"class=com.knowgate.crm.Company&gu_company=012345678901234567
890123456789AB&dt_founded DATETIME yyyy-MM-dd HH:mm:ss=1999-
02-12 00:00:00&nm_legal=ACME%20CORP &nu_employees
INTEGER=352"
  MsgBox .responseText
End With ' HttpReq
```

The datatypes that may follow column names are : CHAR, VARCHAR, DATE, DATETIME, TIMESTAMP, SMALLINT, INTEGER, FLOAT, DOUBLE, DECIMAL, NUMERIC.

How data is saved

Data is saved either by instantiating the specified subclass of DBPersist and calling `store()` method of that subclass or else by calling an

implementation of interface
`com.knowgate.hipergate.datamodel.ImportLoader`

Security

The specified user must have enough privileges over the Workarea for reading and writing data.

Object HttpDataObjsServlet calls internally first to method `autenticate()` from class `ACL` of package `com.knowgate.acl`. Afterwards, if the table from which to read or write contains a column named `gu_workarea`, then it calls methods `isAdmin()` `isPowerUser()` and `isUser()` from class `WorkArea` of `com.knowgate.workareas` for determining whether or not the user has enough permissions for the requested operation.

☞ It is strongly recommended to limit usage of HttpDataObjsServlet by imposing additional security measures such as basic web server authentication and client IP restrictions.

Aditional examples

Global variables and generic purpose function for sending HTTP POST requests from VBScript

```
WORKAREA = "5262a821135070db7b3100126c066ced" ' TEST Work
Area
USERID = "5262a821135070db7d310012ac122ac7" ' GUID of user of
TEST Work Area
PASSWD = "TEST" ' user password
COL_DELIM = "|"
ROW_DELIM = ";"
MAX_ROWS = 100

CONNECTION_PARAMETERS = "profile=hipergate&rowdelim=" &
ROW_DELIM & "&coldelim=" & COL_DELIM & "&maxrows=" & MAX_ROWS
& "&skip=0&gu_workarea=" & WORKAREA + "&user=" & USERID &
"&password=" & PASSWD

Function AjaxPost(querystr)
  Set HttpReq = CreateObject("Msxml2.XMLHTTP")
  With HttpReq
    .open "POST",
"http://localhost:8080/hipergate/servlet/HttpDataObjsServlet"
, False
    .setRequestHeader "Content-Type", "application/x-www-
form-urlencoded"
```

```
    .send querystr
    RespTxt = HttpReq.responseText
  End With
  Xcpt = InStr(RespTxt, "Exception")
  If Xcpt>0 Then RespTxt = Mid(RespTxt, Xcpt)
  ' MsgBox RespTxt
  AjaxPost = RespTxt
End Function
```

Search contact by e-mail

```
' Return: Contact_GUID|Name|Surname|Company_Name
EMAIL = "user@knowgate.com"
AjaxPost CONNECTION_PARAMETERS &
"&command=query&table=k_member_address" &
"&where=gu_workarea%3D'" & WORKAREA &
"'%20AND%20tx_email%3D'" & EMAIL & "'" &
"&fields=gu_contact,tx_name,tx_surname,nm_legal"
```

Search contact using an identifier previously saved for him from a third party application

```
' Return: Contact_GUID |Name|Surame
ID = "101" ' Id. of contact from a third party application
VARCHAR(50)
AjaxPost CONNECTION_PARAMETERS &
"&command=query&table=k_contacts" & "&where=gu_workarea%3D'"
& WORKAREA & "'%20AND%20id_ref%3D'" & ID & "'" &
"&fields=gu_contact,tx_name,tx_surname"
```

Insert or update a contact

```
' Use email as alternative primary key for avoiding
duplicates
' Return: Contact Global Unique Identifier CHAR(32)
ID = "104" ' Identifier of the contact at the tirad party
application VARCHAR(50)
NAME = "John"
SURNAME = "Brown"
EMAIL = "john@thebrowns.com"
COMPANY = "ACME"
NATIONALITY = "es" ' 2 letters country ISO code
COUNTRY = "us" ' 2 letters country ISO code
GUID = AjaxPost (CONNECTION_PARAMETERS &
"&command=update&class=com.knowgate.crm.ContactLoader" &
"&id_contact_ref=" & ID & "&tx_name=" & NAME & "&tx_surname="
& SURNAME & "&tx_email=" & EMAIL & "&nm_legal=" & COMPANY &
"&id_nationality=" & NATIONALITY & "&id_country=" & COUNTRY)
```

Save new opportunity

```
' Return: GUID of new opportunity CHAR(32)
ID = "104" ' Identifier of contact at tirad party application
VARCHAR(50)
' For updating instead of inserting the opportunity and
parameter "&gu_oportunity=" & GUID_DE_LA_OPORTUNIDAD
OBJETIVE = "Product or Service 1" ' Value of field
k_oportunities.id_objetive
AMOUNT = "5"
INTEREST = "1" ' Degree of interest: 0=None, 1=A few, 2=Some,
3=Much
STATUS = "NUEVA" ' Opportunity status: NUEVA | ABIERTA |
GANADA | PERDIDA | APLAZADA | ABANDONADA
NOTES = "" ' Notes and comments
OPORTUNIDAD = AjaxPost (CONNECTION_PARAMETERS &
"&command=update&class=com.knowgate.crm.OportunityLoader" &
"&id_ref=" & ID & "&bo_private SMALLINT=0&id_ref=" & ID &
"&id_objetive=" & OBJETIVE & "&im_revenue FLOAT=" & AMOUNT &
"&tx_company=" & COMPANY & "&tx_contact=" & NAME & "%20" &
SURNAME & "&tl_oportunity=" & OBJETIVE & "%20/%20" & NAME &
"%20" & SURNAME & "&lv_interest SMALLINT=" & INTEREST &
"&id_status=" & STATUS & "&tx_note=" & NOTES)
```

Change opportunity status from "NEW" to "WON"

```
ID = "104" ' Id. Of contact at tirad party application
VARCHAR(50)
OBJETIVE = "Product or Service 1" ' Value of field
k_oportunities.id_objetive
STATUS = "GANADA"
' First get contact GUID at hipergate from his Id. at another
application
CONTACT_GUID = Left(AjaxPost(CONNECTION_PARAMETERS &
"&command=query&table=k_contacts" & "&where=gu_workarea%3D'"
& WORKAREA & "'%20AND%20id_ref%3D'" & ID & "'" &
"&fields=gu_contact"), 32)
' Every field must be re-stored in each transaction
FIELDS =
"gu_oportunity,gu_writer,bo_private,dt_next_action,dt_last_ca
ll,lv_interest,nu_oportunities,gu_campaign,gu_company,gu_cont
act,tx_company,tx_contact,tl_oportunity,tp_oportunity,tp_orig
in,im_revenue,im_cost,id_objetive,id_message,tx_note"
' Con el GUID del individuo en Hipergate buscar la
oportunidad por objetivo (si hay varias considerar sólo la
primera)
OPPORTUNITIES = AjaxPost(CONNECTION_PARAMETERS &
"&command=query&table=k_oportunities" &
"&where=gu_workarea%3D'" & WORKAREA &
"'%20AND%20gu_contact%3D'" & CONTACT_GUID &
"'%20AND%20id_objetive%3D'" & OBJETIVE & "'&fields=" &
FIELDS)
' Store the opportunity at an array by splitting the returned
```

```
string
OPORTUNITY = Split(Split(OPPORTUNITIES, ROW_DELIM,
MAX_ROWS)(0), COL_DELIM)
PARAMETERS = "" ' This string holds the input parameters
except id_status and tx_cause
' Concat input parameters from previously readed data
FIELDS = Split(FIELDS,",")
For o = 0 To UBound(OPORTUNITY)
  If OPORTUNITY (o)="null" Or IsNull(OPORTUNITY(o)) Then
OPORTUNITY (o) = ""
  ' Do not send empty string
  If Len(OPORTUNITY (o))>0 Then
    TIPO = ""
    ' For fields that are not VARCHAR add their type after
the name
    If FIELDS(o)="bo_private" Or FIELDS(o)="lv_interest" Then
TIPO = " SMALLINT"
    If FIELDS(o)="nu_oportunities" Then TIPO = " INTEGER"
    If FIELDS (o)="im_cost" Or FIELDS (o)="im_revenue" Then
TIPO = " FLOAT"
    If FIELDS (o)="dt_next_action" Or FIELDS
(o)="dt_last_call" Then TIPO = " DATE"
   PARAMETERS = PARAMETERS & "&" & FIELDS(o) & TIPO & "=" &
OPORTUNITY(o)
  End If
Next
' Use the GUID of the opportunity for updating it
AjaxPost CONNECTION_PARAMETERS &
"&command=update&table=k_oportunities" &
"&where=gu_workarea%3D'" & WORKAREA &
"'%20AND%20gu_oportunity%3D'" & OPORTUNITY(0) & "'" &
PARAMETROS & "&id_status=" & STATUS & "&tx_cause=VENTA"
```

14 Integration with Google Data

Calendar

hipergate calendar can be synchronized with Google calendar by using class `com.knowgate.gdata.GCalendarSynchronizer`.

The synchronization is bidirectional: hipergate meetings can be written to Google Calendar and Google Events can be written to hipergate calendar.

The matching between hipergate meetings and Google events is done by using the iCalendar unique identifier of the event as its primary key.

When automatic synchronization is enabled an hipergate user can only have one Google Calendar associated. If a Google account has several calendars, only one of them can be synchronized with hipergate at a time.

For enabling automatic synchronization between hipergate and Google Calendar do the following:

1. Put property gdatasync=1 at hipergate.cnf and re-start the web server.
2. Go to Work Area Configuration and ensure that both Collaborative Tools and Passwords Manager modules are enabled for a permissions group to which the main hipergate user of the calendar belongs.
3. Logged as the user of hipergate who will manage the calendar, go to the Passwords Manager and create a new GMail password entry. Put in it the e-mail of the Google account, his password and the name of the Google Calendar.
4. Once the GMail record is created at the Passwords Manager each read or write operation performed from the web interface at hipergate calendar is automatically synchronized with Google Calendar. Direct calls to com.knowgate.addrbook methods do not trigger any Google Calendar synchronization.

Google Maps

hipergate addresses can be sent to Google API from the web interface for displaying their positions at a map. For enabling Google Maps integration do the following:

1. Put property googlemapskey at hipergate.cnf if you do not have a Google Maps key sign up for one here.
2. For an address to be sent to Google Maps, it must have at least a street name and a city name.

The JSP generates Google maps is `common/google_map.jsp`

This page takes as parameter the GUID of an address and tries to show it at Google Maps by using the street type, street name, building number, city, state and country.

The page `google_map.jsp` is oponed as a pop-up either from the company or contact liftings of from the address edition form..

Computing distances

There is another page for computing the distance between two addresses at `common/distance_gmap.jsp`

The distance in kilometers between two point is first computed using Google Maps and the cached at table `k_distances_cache`. The origin and destination points are identified by any string which is recognized by Google Maps.

`distance_gmap.jsp` is a page designed to be placed inside a FRAME, it writes the computed or cached distance at an INPUT control of the first child of its parent window.

16 Calendar API

Hipergate calendar may be accessed from third party applications by using three different client APIs.

Calendar model

The basic work unito f the calendar is the meeting which is represented by class Hipergate.CalendarMeeting at Microsoft .NET client API and by class com.knowgate.addrbook.client.CalendarMeeting at the Pure Java client.

Meetings are identified by a string which follows iCalendar standard recomendation for unique keys.

Each meeting has an organizer, a title, a start date, an end date anda n optional extended description.

Meetings may also have attendants and resources.

The attedants are uniquely identified by their emails.

Resources must have a unique name for eaach calendar.

Security model

Access to a calendar is granted by checking an email and password against k_users table. In hipergate there is a one to one relationship between calendars and application users.

The access to the calendar is done by verifying the user's email and his password. For accesing a calendar, his owner must have been previously created at hipergate. There is a one to one relationship between users and calendars so that the email and password of a user uniquely identified a single calendar.

Authentification process

The calendar service does not maintain server side states. The authentification process requests a security token upon its first call by

passing the user email and password. This token is then used at each next call to the calendar service on the same session.

Installing then calendar service at Server side

The calendar service is implemented at servlet
`com.knowgate.http.HttpCalendarServlet`

This servlet may be activated by adding the following lines to /WEB-INF/web.xml file

```
<servlet>
  <servlet-name>HttpCalendarServlet</servlet-name>
  <servlet-class>
    com.knowgate.http.HttpCalendarServlet
  </servlet-class>
  <init-param>
    <param-name>profile</param-name>
    <param-value>hipergate</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>HttpCalendarServlet</servlet-name>
  <url-pattern>/servlet/HttpCalendarServlet</url-pattern>
</servlet-mapping>
```

Data types

The data types recognized by calendar service function calls are:

id: Maximum 50 characters.
gu: Maximum 32 characters.
type: Maximum 16 characters.
name: Maximum 100 characters.
surname: Maximum 100 characters.
title: Maximum 100 characters.
active: Boolean 0 or 1.
privacy: Boolean 0 or 1.
email: Maximum 100 characters of lowercase letters only. It must match the regular expression: [\w\x2E_-]+@[\w\x2E_-]+\x2E\D{2,4}
description: Maximum 254 characters.
comments: Maximum 254 characters.
startdate: Date and time. For the input format for HTTP GET or POST parameters is yyyyMMddHHmmss The XML output format is yyyy-MM-ddTHH:mm:ss
enddate: Same format as startdate
timezone: 6 characters with format [+|-]hh:mm

REST API

The access through HTTP GET supports the following commads:

connect

Get a security token for a given email and password.
Because each calendar is associated with a single user, the logged user will be the owner of all meetings created Turing the session stablished by calling the connect command.

Request

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command
=connect&user=johns@yoourmail.com&password=xxxxxxx
```

Response XML (success) with the security token at tag <value>

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="connect" code="0">
<error></error>
<value>pur74y7abpckbpq4h8twxkv94rf8fhjebuyvb8vj</value>
</calendarresponse>
```

Response XML (error)

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="connect" code="-1">
<error>User not found</error>
<value></value>
</calendarresponse>
```

disconnect

Close session and discard the active security token.

Request

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command
=disconnect&token=xxxxxxx
```

Response XML

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="disconnect" code="0">
<error></error>
<value>true</value>
```

```
</calendarresponse>
```

getMeetings

Get a listo f all meetings at calendar between two dates.

Request

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command
=getMeetings&token=xxxxxxxx&startdate=19900101000000&enddate=
20200101000000&type=meeting
```

The type is optional.

The format for start and end dats must be yyyyMMddHHmmss

Response XML

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="getMeetings" code="0">
  <error></error>
  <value>2</value>
  <meetings count="2">
    <meeting type="meeting">
      <id>c0a810a212c400a563e100004dead712@hipergate.org</id>
      <gu>c0a810a212c400a563e100004dead712</gu>
      <startdate>2010-11-08T09:00:00</startdate>
      <enddate>2010-11-08T13:00:00</enddate>
      <privacy>false</privacy>
      <title>Something to talk about</title>
      <description></description>
      <rooms count="0"></rooms>
      <attendants count="1">
        <attendant>
          <id>1232</id>
          <gu>x0a8w0a212c400a563e100004orgv335</gu>
          <name>John</name >
          <surname>Smith</surname>
          <email>Johns@yourmail.com</email>
          <timezone>+01:00</timezone>
       </attendant>
      </attendants>
    </meeting>
    <meeting type="meeting">
      <id>c0a810a212c4005fa47100003ea5d84b@hipergate.org</id>
      <gu>c0a810a212c4005fa47100003ea5d84b</gu>
      <startdate>2010-11-11T09:00:00</startdate>
      <enddate>2010-11-11:15:00:00</enddate>
      <privacy>0</privacy>
      <title>Wednesday at three o’clock</title>
      <description></description>
      <organizer>
```

```
        <id>1232</id>
        <gu>x0a8w0a212c400a563e100004orgv335</gu>
        <name>John</name >
        <surname>Smith</surname>
        <email>Johns@yourmail.com</email>
        <timezone>+01:00</timezone>
      </organizer>
      <rooms count="0"></rooms>
      <attendants count="2">
        <attendant>
          <id>1232</id>
          <gu>x0a8w0a212c400a563e100004orgv335</gu>
          <name>John</name >
          <surname>Smith</surname>
          <email>Johns@yourmail.com</email>
          <timezone>+01:00</timezone>
       </attendant>
        <attendant>
          <id>241</id>
          <gu>e0a8w0a212c400a563e100004orgv887</gu>
          <name>Paul</name >
          <surname>Brown</surname>
          <email>Paulb@yourmail.com</email>
          <timezone>+00:00</timezone>
       </attendant>
      </attendants>
    </meeting>
  </meetings>
</calendarresponse>
```


getMeetingsForRoom

Get listo of meetings that use a given resource between two dates.

Request

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command
=getMeetings&token=xxxxxxxx&startdate=19900101000000&enddate=
20200101000000&room=EINSTEIN
```

getMeeting

Geta all the details about an meeting identified by its iCalendar Id..

Request

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command=
getMeetings&token=xxxxxxxx&meeting=icalendar_id_of_meeting@hip
ergate.org
```

Respuesta XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="getMeeting" code="0">
  <error></error>
  <value>true</value>
  <meetings count="1">
    <meeting type="meeting">
      <id>c0a810a212c5a74a926100000a9716f3@hipergate.org</id>
      <gu>c0a810a212c5a74a926100000a9716f3</gu>
      <startdate>2010-11-17T09:00:00</startdate>
      <enddate>2010-11-17T15:00:00</enddate>
      <privacy>0</privacy>
      <title>Encounter X</title>
      <description>Description of encounter X</description>
      <organizer>
        <id>1232</id>
        <gu>x0a8w0a212c400a563e100004orgv335</gu>
        <name>John</name>
        <surname>Smith</surname>
        <email>Johns@yourmail.com</email>
        <timezone>+01:00</timezone>
      </organizer>
      <rooms count="1">
        <room type="CLASSROOM" active="1">
          <name>EINSTEIN</name>
          <comments></comments>
        </room>
      </rooms>
      <attendants count="2">
        <attendant>
          <id>1232</id>
          <gu>x0a8w0a212c400a563e100004orgv335</gu>
          <name>John</name >
          <surname>Smith</surname>
          <email>Johns@yourmail.com</email>
          <timezone>+01:00</timezone>
        </attendant>
        <attendant>
          <id>241</id>
          <gu>e0a8w0a212c400a563e100004orgv887</gu>
          <name>Paul</name >
          <surname>Brown</surname>
          <email>Paulb@yourmail.com</email>
          <timezone>+00:00</timezone>
        </attendant>
      </attendants>
    </meeting>
  <meetings>
</calendarresponse>
```


getRooms

Get a listo f all resources.

Request

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command
=getRooms&token=xxxxxxxx&type=CLASSROOM
```

The type parameter is optional.

Respponse XML

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="getMeeting" code="0">
  <error></error>
  <value>2</value>
  <rooms>
    <room type="CLASSROOM" active="1">
      <name>EINSTEIN</name>
      <comments></comments>
    </room>
    <room type="CLASSROOM" active="0">
      <name>NEWTON</name>
      <comments>This classroom is closed</comments>
    </room>
  </rooms>
</calendarresponse>
```

getAvailableRooms

Get a list of resources that are available between two dates.

Request

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command
=getAvailableRooms&token=xxxxxxxx&startdate=19900101000000&en
ddate=20200101000000
```

El formato para la fecha de inicio y fin debe ser yyyyMMddHHmmss

Response XML

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="getAvailableRooms" code="0">
  <error></error>
  <value>2</value>
  <rooms>
    <room type="AULA" active="1">
      <name>EINSTEIN</name>
      <comments></comments>
    </room>
    <room type="AULA">
    <name>NEWTON</name>
    <comments>Podría estar reservada</comments></room>
```

```
  </rooms>
</calendarresponse>
```

isAvailableRoom

Get whether a resource is available between two dates.

Request

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command
=isAvailableRoom&token=xxxxxxxx&startdate=199001010000000&endd
ate=20200101000000&room=EINSTEIN
```

The format for start and end date must be yyyyMMddHHmmss

Response XML (true at element <value> if the resource is available, false otherwise)

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="isAvailableRoom" code="0">
  <error></error>
  <value>true</value>
</calendarresponse>
```

storeMeeting

Insert a new meeting or update a previously existing one.

Request

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command
=storeMeeting&token=xxxxxx&meeting=idicalendar@hipergate.org&
title=Activity%20Title&startdate=20101117184000&enddate=20101
118195000&rooms=ROOM1,ROOM2&attendants=guest@mail.com
```

The parameters meeting, title, startdate and enddate are all required.

The meeting parameter must be the unique iCalendar identifier of the meeting.

The parameters rooms and attendants are optional. The must contain a list of comma separated values with the names of resources ande mails of the attendants. These emails must belong to user accounts that already exist at the same domain of the calendar.

The logged use ris always added as attendant and organizer of the meeting, even if his email is not explicitly added to the attendants list at the request.

Respponse


```
<?xml version="1.0" encoding="UTF-8"?>

<calendarresponse command="storeMeeting" code="0">
  <error></error>
  <value>true</value>
  <meetings count="1">
    <meeting type="meeting">
      <id>idprueba@hipergate.org</id>
      <gu>c0a8012112c5bb9435b100000c940265</gu>
      <startdate>2010-11-17T18:40:00</startdate>
      <enddate>2010-11-18T19:50:00</enddate>
      <privacy>0</privacy>
      <title>Activity Title</title>
      <description>Activity Description</description>
      <organizer>
        <id>458</id>
        <gu>uua8w0a212c400a563e100004orga765</gu>
        <name>Organizer</name>
        <surname>Attendant</surname>
        <email>user@mail.com</email>
        <timezone>+01:00</timezone>
      </organizer>
      <rooms count type="2">
        <room type="" active="1">
          <name>ROOM1</name>
          <comments></comments>
        </room>
        <room type="" active="1">
          <name>ROOM2</name>
          <comments></comments>
        </room>
      </rooms>
      <attendants count="2">
       <attendant>
          <id>458</id>
          <gu>uua8w0a212c400a563e100004orga765</gu>
          <name>Organizer</name >
          <surname>Attendant</surname>
          <email>user@mail.com</email>
          <timezone>+01:00</timezone>
       </attendant>
        <attendant>
          <id>387</id>
          <gu>ppa8w0a212c400a563e100004orgd937</gu>
          <name>Guest</name >
          <surname>Attendant</surname>
          <email>guest@mail.com</email>
          <timezone>+01:00</timezone>
       </attendant>
      </attendants>
    </meeting>
```

```
    </meetings>
</calendarresponse>
```

.NET client library

The library HipergateCalendarClient.dll contains the classes CalendarMeeting, CalendarRoom, CalendarAttendant y CalendarClient.

The main class for accessing the calendar service from a .NET client program is Hipergate.CalendarClient.

Class Hipergate.CalendarClient

This class keeps the session at client side and provides access to all relevant methods for calendar interaction.

Connect (sServiceUrl As String, sUserEmail As String, sPassword As String) As Boolean

Before calling any method of CalendarClient it is necessary to stablish a session by calling Connect().

sServiceUrl: Base URL for service. Usually http://hostname.com/hipergate/servlet/HttpCalendarServlet

sUserEmail: E-mail of user owner the calendar. This email must be present at tables k_users and k_fellows of hipergate.

sPassword: User password. The same that the user has at k_users table of hipergate.

Disconnect () As Boolean

Close session.

IsAvailable (sRoom As String, dtStart As Date, dtEnd As Date) As Boolean

Check whether a resource is available between two given dates.

sRoom: Resource name.

dtStart: Start date.

dtEnd: End date.

GetRooms () As CalendarRoom()

Get an array with all resources, no matter if they are available or not. If there are no resources then it returns `Nothing`.

GetRooms(sType As String) As CalendarRoom()

Get an array with all resources of a given type no matter if they are available or not. If there are no resources of such type then it returns `Nothing`.

sType: Resource type.

GetAvailableRooms(dtStart As Date, dtEnd As Date) As CalendarRoom()

Get an array with all resources that are available between two given dates. If there are no available resources then it returns `Nothing`.

dtStart: Start date.

dtEnd: End date.

GetAvailableRooms(sType As String, dtStart As Date, dtEnd As Date) As CalendarRoom()

Get an array with all available resources of a given type between two dates. If there are no available resources of such type then it returns `Nothing`.

sType: Resource type.

dtStart: Start date.

dtEnd: End date.

GetMeeting (sMeetingId As String) As CalendarMeeting

Retrieve a meeting object from its iCalendar Id. or its GUID.

sMeetingId: iCalendar Id. or GUID of meeting (any of both).

GetMeetings(dtStart As Date, dtEnd As Date) As CalendarMeeting()

Get an array with all meetings between two dates.

dtStart: Start date.

dtEnd: End date.

GetMeetingsOfType(dtStart As Date, dtEnd As Date, sType As String) As CalendarMeeting()

Get an array with all mettings of a given type between two dates.

dtStart: Start date.

dtEnd: End date.

sType: Meeting type. The standard types are:

```
meeting
call
followup
breakfast
lunch
course
demo
workshop
congress
tradeshow
```

GetMeetingsForRoom (dtStart As Date, dtEnd As Date, sRoom As String) As CalendarMeeting()

Get an array with all mettings that make use of a given resource between two dates.

dtStart: Start date.

dtEnd: End date.

sRoom: Resource name.

StoreMeeting (oMeet As CalendarMeeting) As CalendarMeeting

Store a new meeting or update an already existing one. Meetings are uniquely identified by their iCalendar Id.

The iCalendar Id. may be explicitly set by the caller program or, if it is Leith blank, a default value is assigned to it before storing the meeting at Server side.

This function returns the saved object as it is stored at Server side. The returned object may have slight changes from the one sent. The automatically assigned iCalendar Id. and GUID are two of those changes.

oMeet: Meeting to be stored.

DeleteMeeting (sMeetingId As String) As CalendarMeeting

Delete a meeting.

oMeet: iCalendar Id. or GUID.

Example of how to use the .NET client from VisualBASIC

```
' Create a client calendar object
Dim c As New Hipergate.CalendarClient

' Connect to service
c.Connect("http://localhost/hipergate/servlet/HttpCalendarServlet",
"user@test.com", "TEST")

' Check if the resource of name NEWTON is available right now
Dim a As Boolean = c.IsAvailable("NEWTON", DateValue(Now), DateValue(Now))

' Get an array with all resources
```

```
Dim r() As Hipergate.CalendarRoom = c.GetRooms()

' Get an array with all available resources
Dim d() As Hipergate.CalendarRoom = c.GetAvailableRooms(DateValue(Now),
DateValue(Now))

' List activities between 1/1/2000 and 12/31/2020
Dim m() As Hipergate.CalendarMeeting = c.GetMeetings(New Date(2000, 1, 1, 0,
0, 0), New Date(2020, 12, 31, 23, 59, 59))

' Create a new activity at room NEWTON
Dim e As New Hipergate.CalendarMeeting
Dim f As Hipergate.CalendarMeeting
e.title = "Meeting title up to 100 characters"
e.description = "Meeting description up to 1000 characters"
e.startdate = Now ' Start date
e.enddate = DateAdd(DateInterval.Hour, 2, e.startdate)
e.AddRoom("NEWTON") ' Call AddRoom once for each resource
e.AddAttendant("administrator@hipergate-test.com")
f = c.StoreMeeting(e)

' Get stored meeting
Dim g As Hipergate.CalendarMeeting = c.GetMeeting(f.id)
' Move the start date ten minutes
g.startdate = DateAdd(DateInterval.Minute, 10, g.startdate)
c.StoreMeeting(g)

' Close session
c.Disconnect()
```

Java client library

The client side Java API for remotely accesing the calendar may be found at package `com.knowgate.addrbook.client`

The main class for accesing the calendar is `CalendarServices`.

Example of usage of Java client library

```
CalendarServices oCal = new CalendarServices();

// Connect to service
oCal.connect("http://localhost/hipergate/servlet/HttpCalendarServlet","admin
istrator@hipergate-test.com","TEST");

// Ceheck if resource NEWTON is available right now
boolean a = oCal.isAvailableRoom("NEWTON", new Date(),new Date ());

// Obtener un array con todos los recursos
ArrayList<CalendarRoom> r = oCal.getRooms();

// Get an array with all available resources
ArrayList<CalendarRoom> d = oCal.getAvailableRooms(new Date (),new Date ());

// Get an array with activities between 1/1/2000 and 12/31/2020
ArrayList<CalendarMeeting> m = oCal.getMeetings(new Date (100,0,1), new Date
(120,11,31));

// Store a new activity at room NEWTON
CalendarMeeting e = new CalendarMeeting();
e.setTitle("Test Activity Title");
e.setDescription("Test Activity Extended Description");
```

```
e.setStartDate(new Date());
e.setEndDate(new Date(e.getStartDate().getTime()+3600000l));
e.addRoom("NEWTON");
e.addAttendant("administrator@hipergate-test.com");
oCal.storeMeeting(e);
```

15 Common customization tasks

Changing the skin

The look'n feel skin for hipergate is determined by file styles.css which is located at a subdirectory under /skins directory.

The default skin is xp which may be changed by editing hipergate.cnf properties file and the initial JavaScript code of login.html page.

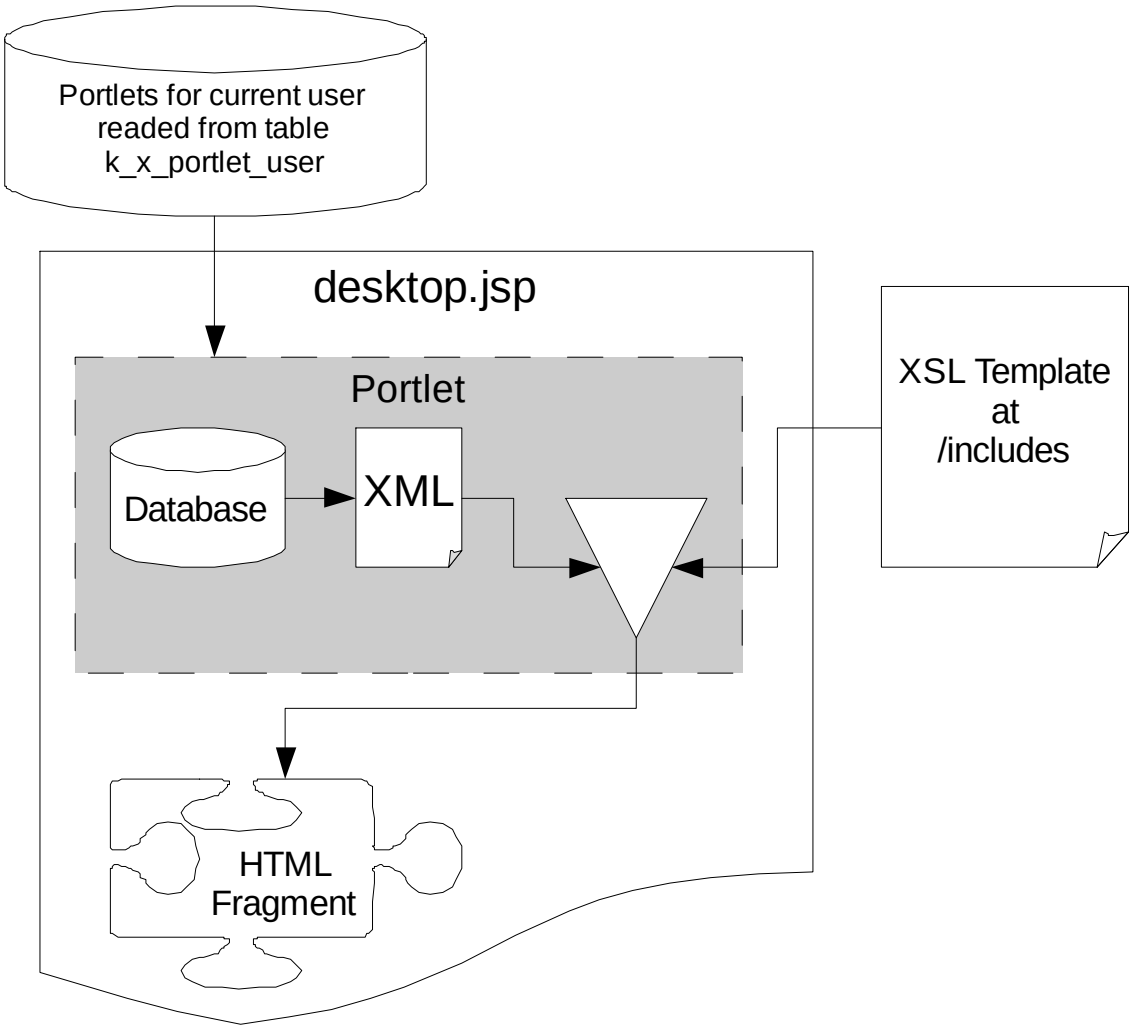
Adding a portlet to the home page

hipergate home page (`/common/desktop.jsp`) allows displaying dynamic contents in a two columns layout based on classes implementing `javax.portlet.GenericPortlet` interface.

The contents shown at the home page of hipergate may be customized for each user. The database table `k_x_portlet_user` holds a list of the contents to be displayed for each user.

Each active portlet generates an XHTML code fragment that is painted at the home page.

Diagram of how the portlets are painted at the home page



The standard portlets of hipergate public build are located at `com.knowgate.http.portlets` package.

Here is an example of a Hello World portlet.

```
import java.io.File;
import java.io.IOException;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;

import java.util.Date;
import java.util.Properties;
import java.util.Enumeration;

import java.sql.SQLException;

import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerConfigurationException;

import javax.portlet.GenericPortlet;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.portlet.PortletException;
import javax.portlet.WindowState;
```

```
import com.knowgate.jdc.JDCConnection;
import com.knowgate.dataobjs.DB;
import com.knowgate.dataobjs.DBBind;
import com.knowgate.dataobjs.DBCommand;
import com.knowgate.dataxslt.StylesheetCache;
import com.knowgate.dfs.FileSystem;

public class HelloWorld extends GenericPortlet {

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

  public HelloWorld() { }

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

  public HelloWorld(HipergatePortletConfig oConfig)
    throws javax.portlet.PortletException {

    init(oConfig);
  } // HelloWorld

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

  public String render(RenderRequest req, final String sEncoding)
    throws PortletException, IOException, IllegalStateException {

    String sOutput;
    ByteArrayInputStream oInStream;
    ByteArrayOutputStream oOutStream;

    FileSystem oFS = new FileSystem(FileSystem.OS_PUREJAVA);

   // ***************************************************
   // These are properties passed from desktop.jsp page

    String sDomainId   = req.getProperty("domain");
    String sWorkAreaId = req.getProperty("workarea");
    String sUserId     = req.getProperty("user");
    String sZone       = req.getProperty("zone");
    String sLang       = req.getProperty("language");
    String sStorage    = req.getProperty("storage");
    String sTemplatePath  = req.getProperty("template");
   String sCacheFilesDir = sStorage+"domains"+File.separator+sDomainId+
   File.separator+"workareas"+File.separator+sWorkAreaId+File.separator+
   "cache"+File.separator+sUserId;
    String sCachedFile = getClass().getName() + "." +
   req.getWindowState().toString() + ".xhtm";

    // No more properties
   // *******************

   // ***************************************************
   // Portlets are cached for reducing database accesses

    Date oDtModified = (Date) req.getAttribute("modified");

    if (null!=oDtModified) {
      try {

        File oCached = new File(sCacheFilesDir+File.separator+sCachedFile);

        if (!oCached.exists())
          oFS.mkdirs(sCacheFilesDir);
        else if (oCached.lastModified()>oDtModified.getTime())
          return oFS.readfilestr("file://"+
                 sCacheFilesDir+File.separator+sCachedFile,
                 sEncoding==null ? "ISO8859_1" : sEncoding);
      } catch (Exception xcpt) {
        System.err.println(xcpt.getClass().getName() + " " +
                           xcpt.getMessage());
      }
    } // fi (oDtModified)
```

```
// ***************************************************

String sXML = "<?xml version=\"1.0\" encoding=\"UTF-8\"?><?xml-
               stylesheet type=\"text/xsl\"?>";

if (req.getWindowState().equals(WindowState.MINIMIZED)) {

  // If portlet state is minimized then there is no need to do any
     database access

  sXML += "<FullName/>";
}
else {

  // Get database connection from desktop.jsp page

  DBBind oDBB = (DBBind)
                getPortletContext().getAttribute("GlobalDBBind");

  JDCConnection oCon = null;

  try  {
    oCon = oDBB.getConnection(getClass().getName());

    // This is the data retrived from the database that must be
   //  shown by the portlet

    String sFullName = DBCommand.queryStr(oCon, "SELECT "+DB.nm_user+",'
                       ","+DB.tx_surname1+" FROM "+DB.k_users+" WHERE
                       '"+DB.gu_user+"='"+sUserId+"'");

    oCon.close(getClass().getName());
    oCon = null;

     // The XSL stylesheet will read this XML node <FullName>

    sXML += "<FullName>"+sFullName+"</FullName>";
  }
  catch (SQLException e) {
    sXML += "<FullName/>";

    try {
      if (null!=oCon) if (!oCon.isClosed()) oCon.close("HelloWorld");
    } catch (SQLException ignore) { }
  }
} // fi (WindowState)

try {

   // ******************************************
   // Set input parameters for XSL StyleSheet

   Properties oProps = new Properties();
   Enumeration oKeys = req.getPropertyNames();
   while (oKeys.hasMoreElements()) {
     String sKey = (String) oKeys.nextElement();
     oProps.setProperty(sKey, req.getProperty(sKey));
   } // wend

   oProps.setProperty("windowstate",
    req.getWindowState().equals(WindowState.MINIMIZED) ?
   "MINIMIZED" : "NORMAL");

   // ******************************************

   // ******************************************
   // Perform XSLT Transformation for generating
   // portlet XHTML code fragment.

   if (sEncoding==null)
     oInStream = new ByteArrayInputStream(sXML.getBytes());
```

```
        else
          oInStream = new ByteArrayInputStream(sXML.getBytes(sEncoding));

        oOutStream = new ByteArrayOutputStream(4000);

        StylesheetCache.transform (sTemplatePath, oInStream, oOutStream,
                                   oProps);

        if (sEncoding==null)
          sOutput = oOutStream.toString();
        else
          sOutput = oOutStream.toString("UTF-8");

        oOutStream.close();

        oInStream.close();
        oInStream = null;

       // ************************************
       // Cache generated XHTML code into a file

        oFS.writefilestr ("file://"+sCacheFilesDir+
                          File.separator+sCachedFile, sOutput,
                          sEncoding==null ? "ISO8859_1" : sEncoding);
      }
      catch (Exception xcpt) {
        throw new PortletException(xcpt.getClass().getName() + " " +
                                   xcpt.getMessage(), xcpt);
      }

      return sOutput;
    } // render

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

    public void render(RenderRequest req, RenderResponse res)
      throws PortletException, IOException, IllegalStateException {
      res.getWriter().write(render(req,res.getCharacterEncoding()));
    } // render

  } // HelloWorld
```

This portlet uses an XSL file which must be placed at /includes directory of hipergate webapp.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" version="4.0" media-type="text/html" omit-
xml-declaration="yes"/>

<xsl:param name="param_domain" />
<xsl:param name="param_workarea" />
<xsl:param name="param_skin" />

<xsl:template match="/">

  <TABLE CELLSPACING="0" CELLPADDING="0" BORDER="0" />
    <TR>
      <TD WIDTH="2px" CLASS="subtitle"
       BACKGROUND="../images/images/graylineleftcorner.gif">
        <IMG SRC="../images/images/spacer.gif" WIDTH="2"
         HEIGHT="1" BORDER="0" /></TD>
      <TD BACKGROUND="../images/images/graylinebottom.gif">
        <TABLE CELLSPACING="0" CELLPADDING="0" BORDER="0" />
          <TR>
            <TD COLSPAN="2" CLASS="subtitle"
```

```
            BACKGROUND="../images/images/graylinetop.gif">
             <IMG SRC="../images/images/spacer.gif" HEIGHT="2"
              BORDER="0" />
           </TD>
           <TD ROWSPAN="2" CLASS="subtitle" ALIGN="right">
             <IMG SRC="../skins/{$param_skin}/tab/angle45_24x24.gif"
              WIDTH="24" HEIGHT="24" BORDER="0" /></TD>
          </TR>
         <TR>
           <TD CLASS="subtitle">
             <xsl:if test="$param_windowstate='NORMAL'">
             <A
                HREF="windowstate.jsp?gu_user={$param_user}&amp;nm_
                page=desktop.jsp&amp;nm_portlet=com.knowgate.http.p
                ortlets.HelloWorld&amp;gu_workarea={$param_workarea
                }&amp;nm_zone={$param_zone}&amp;id_state=MINIMIZED"
                ><IMG
                SRC="../skins/{$param_skin}/tab/minimize12.gif"
                WIDTH="16" HEIGHT="16" BORDER="0" HSPACE="4"
                VSPACE="2"/></A>
             </xsl:if>

             <xsl:if test="$param_windowstate='MINIMIZED'">
             <A
                HREF="windowstate.jsp?gu_user={$param_user}&amp;nm_
                page=desktop.jsp&amp;nm_portlet=com.knowgate.http.p
                ortlets.HelloWorld&amp;gu_workarea={$param_workarea
                }&amp;nm_zone={$param_zone}&amp;id_state=NORMAL"><I
                MG SRC="../skins/{$param_skin}/tab/maximize12.gif"
                WIDTH="16" HEIGHT="16" BORDER="0" HSPACE="4"
                VSPACE="2"/></A>
             </xsl:if>
           </TD>
            <TD BACKGROUND="../skins/{$param_skin}/tab/tabbackflat.gif"
             CLASS="subtitle" ALIGN="left" VALIGN="middle">
              <IMG SRC="../images/images/3x3puntos.gif" BORDER="0" />
              Hello World!
             </TD>
         </TR>
       </TABLE>
     </TD>
     <TD VALIGN="bottom" ALIGN="right" WIDTH="3px"
      CLASS="htmlbody">
       <IMG SRC="../images/images/graylinerightcornertop.gif"
        WIDTH="3" BORDER="0" /></TD>
   </TR>
   <TR>
     <TD WIDTH="2px" CLASS="subtitle"
      BACKGROUND="../images/images/graylineleft.gif">
       <IMG SRC="../images/images/spacer.gif" WIDTH="2" HEIGHT="1"
        BORDER="0" />
     </TD>
     <TD CLASS="subtitle"><IMG SRC="../images/images/spacer.gif"
      HEIGHT="1" BORDER="0" /></TD>
     <TD WIDTH="3px" ALIGN="right">
       <IMG SRC="../images/images/graylineright.gif" WIDTH="3"
        HEIGHT="1" BORDER="0" /></TD>
   </TR>
   <TR>
     <TD WIDTH="2px" CLASS="subtitle"
      BACKGROUND="../images/images/graylineleft.gif">
       <IMG SRC="../images/images/spacer.gif" WIDTH="2" HEIGHT="1"
        BORDER="0" /></TD>
     <TD CLASS="menu1">
```

```
          <TABLE SUMMARY="" CELLSPACING="8" BORDER="0" />
            <TR>
              <TD ALIGN="middle">
                <IMG SRC="../images/images/chequeredflag.gif"
                 BORDER="0" ALT="Chequered Flag">
              </TD>
              <TD ALIGN="left" VALIGN="middle">
                <TABLE SUMMARY="New Item">
                  <TR>
                    <TD>
                      <IMG SRC="../images/images/new16x16.gif"
                       BORDER="0" />
                    </TD>
                    <TD VALIGN="middle">
                      <A HREF="#" onclick="window.open('#',
null,'directories=no,toolbar=no,menubar=no,width=500,height=400')"
CLASS="linkplain">New Item</A>
                    </TD>
                  </TR>
                </TABLE>
              </TD>
            </TR>
            <TR>
              <TD COLSPAN="2">

                   <!-- *** Content HERE *** -->

                   HELLO WORLD!

              </TD>
            </TR>
            <TR>
              <TD COLSPAN="2">

                  <!-- *** Content HERE *** -->

                  <xsl:value-of select="FullName"/>

              </TD>
            </TR>
          </TABLE>
        </TD>
        <TD WIDTH="3px" ALIGN="right"
         BACKGROUND="../images/images/graylineright.gif">
          <IMG SRC="../images/images/spacer.gif" WIDTH="3" BORDER="0" />
        </TD>
      </TR>
      <TR>
        <TD WIDTH="2px" CLASS="subtitle"
         BACKGROUND="../images/images/graylineleft.gif"><IMG
src="../images/images/spacer.gif" WIDTH="2" HEIGHT="1" BORDER="0" />
        </TD>
        <TD CLASS="subtitle"><IMG SRC="../images/images/spacer.gif"
         HEIGHT="1" BORDER="0" /></TD>
        <TD WIDTH="3px" ALIGN="right">
          <IMG SRC="../images/images/graylineright.gif" WIDTH="3"
           HEIGHT="1" BORDER="0" />
        </TD>
      </TR>
      <TR>
        <TD WIDTH="2px" CLASS="subtitle">
          <IMG SRC="../images/images/graylineleftcornerbottom.gif"
           WIDTH="2" HEIGHT="3" BORDER="0" /></TD>
        <TD CLASS="htmlbody"
```

```
        BACKGROUND="../images/images/graylinefloor.gif">
       </TD>
       <TD WIDTH="3px" ALIGN="right">
         <IMG SRC="../images/images/graylinerightcornerbottom.gif"
          WIDTH="3" HEIGHT="3" BORDER="0" /></TD>
    </TR>
  </TABLE>
</xsl:template>
</xsl:stylesheet>
```

Portlets does not necessarily have to be based on XSL templates. The output HTML code could be generated directly by the Java class itself or by any other method.

Step by step guide for adding a new portlet

1. Write your own subclass of GenericPortlet by following the HelloWorld portlet example previously given.
2. Write an XSL template for your HTML code fragment.
3. Make the render() method of your GenericPortlet subclass return the final HTML code by doing an XSLT transformation of the XSL template.
4. Put your XSL template at /includes subdirectory of your webapp.
5. At your XSL template you can paint either a normal or minimized HTML fragment depending on the value of param_windowstate which may be either NORMAL or MINIMIZED. Just add a couple XSLT if statements like these ones:

```
<xsl:if test="$param_windowstate='NORMAL'">
  <A
  HREF="windowstate.jsp?gu_user={$param_user}&amp;nm_pag
  e=desktop.jsp&amp;nm_portlet=com.knowgate.http.portlet
  s.HelloWorld&amp;gu_workarea={$param_workarea}&amp;nm_
  zone={$param_zone}&amp;id_state=MINIMIZED"><IMG
  SRC="../skins/{$param_skin}/tab/minimize12.gif"
  WIDTH="16" HEIGHT="16" BORDER="0" HSPACE="4"
  VSPACE="2"/></A>
</xsl:if>

<xsl:if test="$param_windowstate='MINIMIZED'">
  <A
  HREF="windowstate.jsp?gu_user={$param_user}&amp;nm_pag
  e=desktop.jsp&amp;nm_portlet=com.knowgate.http.portlet
  s.HelloWorld&amp;gu_workarea={$param_workarea}&amp;nm_
  zone={$param_zone}&amp;id_state=NORMAL"><IMG
  SRC="../skins/{$param_skin}/tab/maximize12.gif"
  WIDTH="16" HEIGHT="16" BORDER="0" HSPACE="4"
  VSPACE="2"/></A>
</xsl:if>
```

6. Edit page /common/desktop_custom.jsp and at the JavaScript arrays named `portlets`, `labels` and `enabled` add your portlet class name, a short human readable label for it, and which module of hipergate should be active in order for the portlet to be visible.
7. Log into the application and at the home page click on the link at the bottom Customize This Page. It will give you the chance of showing the new portlet either at the Leith or right column.