



Guía del Programador

Versión 7.0
Diciembre 2013

¿Crees que falta algo o hay algún error en este documento?
Ayúdanos a mejorar.
Envíanos tus sugerencias a translations@hipergate.org

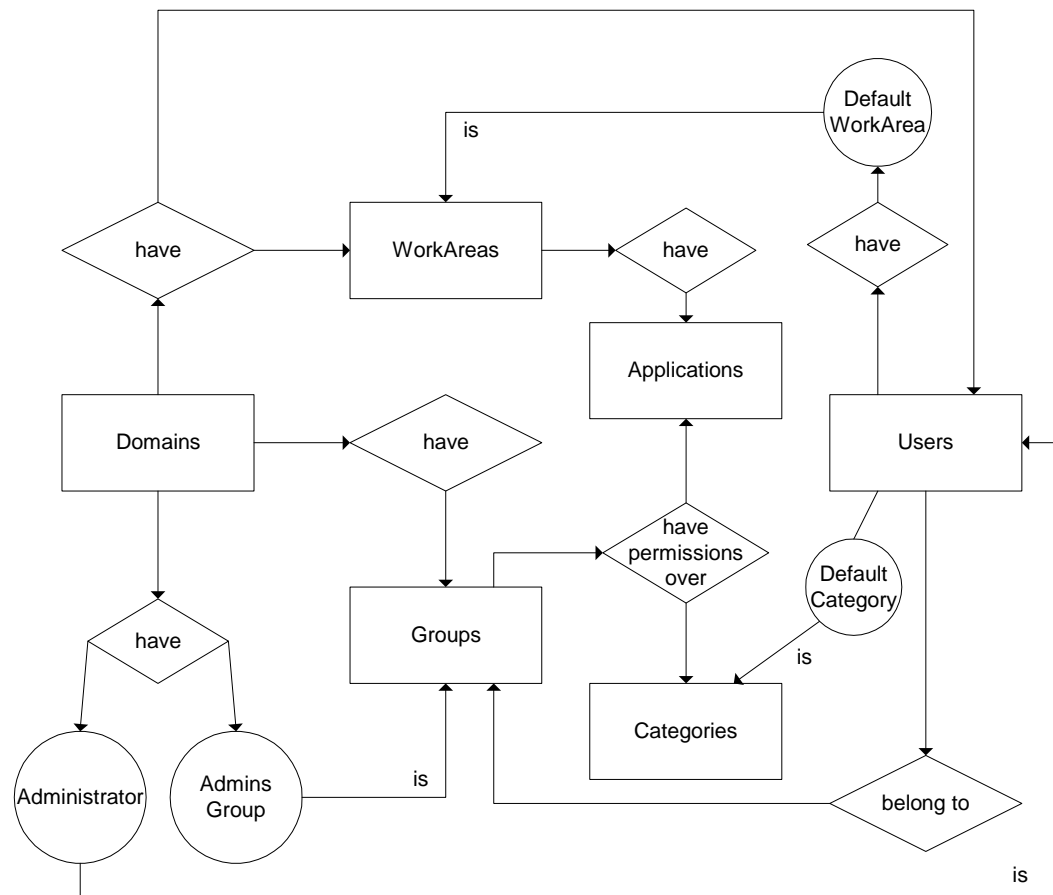
© KnowGate 2003-2013
<http://www.hipergate.org>

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

GUÍA DEL PROGRAMADOR	1
INTRODUCCIÓN	9
CAPAS DE LA APLICACIÓN	9
PASOS PARA EXTENDER HIPERGATE	10
CÓMO AGREGAR OBJETOS AL MODELO DE DATOS	10
Módulos funcionales	10
Tipos independientes del SGBDR	11
Constraints	11
Borrado de objetos	11
Tablas de remonte (lookups)	11
CÓMO CREAR CLASES JAVA DE ACCESO A DATOS	12
DBPersist	12
DBSubset	13
MANTENIMIENTO DE SESIONES	14
PLANTILLAS DE FORMULARIOS DE EJEMPLO	14
void.jsp	14
form_edit.jsp	15
simpleform.jsp	17
form_store.jsp	18
listing.jsp	18
Carga de fechas desde textos de formularios en la base de datos	21
Prevención de ataques por cross-site scripting e inyección SQL	24
CAMBIO DEL SKIN	25
AÑADIR UN PORTLET A LA PÁGINA DE INICIO	25
Guía paso a paso para añadir un nuevo portlet	32
MODELO DE DATOS	33
DÓNDE ENCONTRAR LOS FUENTES DEL MODELO	33
CONVENCIONES	34
Nomenclatura	34
Tablas y Vistas	34
Tipos de datos	36
Identificadores únicos de registro (GUIDs)	36
Fechas de creación y modificación de registro	38
Indicadores de activación de registros	39
Recorrido de datos jerárquicos	39
VERSIONADO DEL MODELO DE DATOS	40
SEGMENTACIÓN DE DATOS POR ÁREAS DE TRABAJO	40
TABLAS DE VALORES ESTÁTICOS GLOBALES	41
TABLAS DE REMONTE POR ÁREA DE TRABAJO	42
Atributos definibles por el usuario	43
Cómo añadir nuevas columnas al modelo de datos	45
INTERNACIONALIZACIÓN DE DATOS	46
Uso del robot asistente de traducción	46
SUBMODELO DE CATEGORIZACIÓN	47
Tablas y Vistas	47
Cómo crear categorías directamente sobre el modelo	50

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

SUBMODELO DE SEGURIDAD Y AUTENTIFICACIÓN DE USUARIOS	53
Dominios	53
Área de Trabajo	54
Roles	54
Usuarios y Grupos	54
Aplicaciones	57
Máscaras estándar de permisos	58
Carga Inicial de Datos para el Submodelo de Seguridad	59



Auditoría de logins	61
Submodelo de Portlets	62
SUBMODELO DE TESAuros, DIRECCIONES E IMÁGENES	63
Tesauros	63
Direcciones Postales	66
Imágenes	68
Cuentas Bancarias	69
SUBMODELO DEL PLANIFICADOR DE TAREAS	70
Comandos	70
Consultas por Formulario (QBF)	72
SUBMODELO DE TRABAJO EN GRUPO	76
Calendarios laborables	78
SUBMODELO DE PRODUCTOS Y DOCUMENTOS	81
Definiciones y Conceptos	81
Elementos del modelo	82
Control de Versiones	86
SUBMODELO DE TIENDA VIRTUAL	88
SUBMODELO DE GESTIÓN DE VENTAS Y RELACIONES CON CLIENTES	91

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

Definiciones y Conceptos	91
Elementos del modelo	92
Restricciones de acceso para compañías e individuos	99
SUBMODELO DE PRODUCCIÓN DE CONTENIDOS	100
SUBMODELO DE FOROS	103
SUBMODELO DE GESTIÓN DE PROYECTOS E INCIDENCIAS	105
SUBMODELO DE LISTAS DE DISTRIBUCIÓN	113
Definiciones y Conceptos	113
Tablas y Vistas	115
SUBMODELO DEL PLANIFICADOR DE TAREAS	117
Tablas y Vistas	117
SUBMODELO HIPERMAIL	122
Almacenamiento de correo	122
Enlace entre el correo y el gestor de contactos de hipergate	123
Cache de mensajes	123
Tablas y Vistas	123
 PROCEDIMIENTOS ALMACENADOS	 126
 DÓNDE ENCONTRAR LOS FUENTES DE LOS PROCEDIMIENTOS ALMACENADOS	 126
Porqué procedimientos almacenados en el gestor	126
CÓMO LLAMAR A LOS PROCEDIMIENTOS DESDE CÓDIGO JAVA	127
PROCEDIMIENTOS DE SEGURIDAD Y AUTENTIFICACIÓN DE USUARIOS	129
PROCEDIMIENTOS DE GESTIÓN DE CATEGORÍAS	133
PROCEDIMIENTOS DE TRABAJO EN GRUPO	137
PROCEDIMIENTOS DE GESTIÓN DE PRODUCTOS	137
PROCEDIMIENTOS DE GESTIÓN DE VENTAS Y RELACIONES CON CLIENTES	138
PROCEDIMIENTOS DE GESTIÓN DE LISTAS	139
Disparadores de Miembros de Listas	140
Cómo reconstruir la vista materializada k_member_address	141
PROCEDIMIENTOS DE FOROS	141
PROCEDIMIENTOS DE GESTIÓN DE PROYECTOS E INCIDENCIAS	142
PROCEDIMIENTOS DE GESTIÓN DE CORREO	144
 CLASES JAVA	 145
 PROPÓSITO DE LAS LIBRERÍAS JAVA	 145
PAQUETES GENÉRICOS Y PAQUETES DEPENDIENTES DEL MODELO	146
PAQUETES ADICIONALES COMPILADOS DENTRO DE HIPERGATE.JAR	147
PAQUETES ADICIONALES COMPILADOS FUERA DE HIPERGATE.JAR	147
MODALIDAD DE DEPURACIÓN Y MODALIDAD DE EXPLOTACIÓN	147
Como averiguar el modo de traza desde la línea de comandos	147
PROPIEDADES DE INICIALIZACION	148
LANZADOR DE TABLAS Y DATOS INICIALES	148
POOL DE CONEXIONES A BASE DE DATOS	150
MAPEO DE OBJETOS JAVA A REGISTROS DE BASE DE DATOS	151
Nombres de Tablas y Campos	153
Cache en RAM de metadatos del SGBDR	153
Transacciones	153
Auditoría	154
Manejo de campos largos	154
Carga de Datos XML	155
Conjuntos de filas leídos a ráfagas	156

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

Uso de la clase ImportExport para cargar texto delimitado	158
Cómo funciona el cargador de usuarios y empleados	163
Cómo funciona el cargador de compañías	166
Cómo funciona el cargador de contactos	167
Cómo funciona el cargador de productos	168
Carga de datos desde archivos de texto delimitado (CSV)	169
Ejemplo de carga de texto compleja.	171
Volcado de base de datos a archivos de texto	175
MÉTODOS ABREVIADOS DE ACCESO A TABLAS DE REMONTE	176
SEGURIDAD Y ROLES DE USUARIOS	176
Cómo crear un nuevo Dominio	176
Cómo eliminar un Dominio	177
Cómo crear una nueva Área de Trabajo Vacía	178
Cómo duplicar un Área de Trabajo	180
Cómo eliminar un Área de Trabajo	180
Cómo crear un Usuario	180
CONSULTAS POR FORMULARIO (QBF)	184
ACCESO A ARCHIVOS	186
Leer archivos de texto en un solo paso	187
Convertir archivos de texto ASCII a Unicode	187
TRANSFORMACIONES XSLT	187
Metadatos	187
Datos	189
EJECUTOR DE TAREAS SENCILLO	190
Ejecución mono-thread por línea de comandos	190
PLANIFICADOR DE TAREAS	190
Ejecución por línea de comandos	192
Callbacks	193
Configuración del número de threads ejecutores	194
Archivo de Log	194
SUBCLASES DE JOB	194
Propiedades de entorno para las subclases de Job	195
Parámetros para cada subclase de Job	195
ENVÍO DE E-MAILS DESDE LA LÍNEA DE COMANDOS	196
ENVÍO DE E-MAILS A DIRECCIONES EXTERNAS MEDIANTE SENDMAIL	197
Jobs en la clase SendMail	199
ENVÍO DE E-MAILS MEDIANTE MIMESENDER	199
Ejemplo de archivo de propiedades para MimeSender	200
Plantilla de contenidos para los e-mails	201
Personalización de los e-mails	201
Seguimiento de mails leídos con Web Beacons	202
Línea de comandos MimeSender	202
Cómo funciona el proceso de mailing	203
Logs de los lotes	204
CÓMO ESCRIBIR RUTINAS PROPIAS DE ENVÍO DE MAILS	204
Obtención de listas de destinatarios	204
Personalización de los mensajes	205
Imágenes adjuntas a los mensajes	207
Ejemplo de una clase basada en SendMail para enviar correos	207
CÓMO USAR EL CARRO DE COMPRA	208
Cómo cargar el carro desde JavaScript	208
Funciones de búsqueda y suma	209
SOPORTE MULTIMONEDA	209
 SCRIPTS JAVA BEANSHELL	 210

CONVENIOS EN EL PASO DE PARÁMETROS ENTRE JAVA Y BEANSHELL 210

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

PÁGINAS JSP **212**

ESTRUCTURA GENERAL DE LAS PÁGINAS	212
JUEGO DE CARACTERES DEL CONTENEDOR DE SERVLETS	213
CONVENIOS DE PROGRAMACIÓN	213
Cabeceras de página	213
Manejo de Conexiones	213
Tipos de Páginas	214
BEANS DE APLICACIÓN	214
Acceso a Datos GlobalDBBind	214
Conexiones a múltiples bases de datos	215
Cache Distribuido GlobalCacheClient	215
SESIONES DE USUARIO	216
Cookies	216
Información cacheada	217
PROCESO DE CREACIÓN DE CUENTAS	218
Tipos de cuentas	218
El proceso de registro	218
PROCESO DE AUTENTIFICACIÓN DE USUARIOS	222
Autenticación Inicial	222
Re-autenticación por página	222
Cómo conectar un usuario a diferentes Áreas de Trabajo	223
Cómo reemplazar el modelo de seguridad nativo	224
Cómo simular accesos de usuarios anónimos	225
LIBRERÍAS DE MÉTODOS ESTÁTICOS PARA PÁGINAS JSP	226
authusrs.jsp	227
cookies.jsp	228
nullif.jsp	228
reqload.jsp	228
EL MENÚ SUPERIOR DE PESTAÑAS	229
Cómo mostrar la opción seleccionada	230
Máscara de Aplicaciones	230
Cache de opciones de menú	230
Acceso a Configuración para el administrador del dominio	230
PERSONALIZACIÓN DE LA PÁGINA DE INICIO	231
SELECCIÓN DE VALORES DE REMONTE	232
Cómo llamar a la página lookup_f.jsp	233
Carga de remotes desde scripts SQL	233
ATRIBUTOS DEFINIDOS POR EL USUARIO	234
ENVÍO DE CORREO EN RESPUESTA A ACCIONES	235
MÓDULO DE TRABAJO EN GRUPO	236
MÓDULO DE FOROS	236
Modo de almacenamiento de los mensajes	237
Generación de documentos RDF Site Summary (RSS)	237
MÓDULO DE GESTIÓN DE RELACIONES CON CLIENTES	238
MÓDULO DE WEBBUILDING	239
BORRADO DIFERIDO DE ARCHIVOS	240

JAVASCRIPTS **241**

LIBRERÍAS DE TERCEROS INCLUIDAS EN EL PRODUCTO	241
CONVENCIONES	241
Decorados y Hoja de Estilo (CSS)	241
Fechas	241
Calendario	242

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

MENÚS	242
LIBRERÍAS JAVASCRIPT	242
Leer y Escribir Cookies	242
Manipulación de ComboBoxes	243
Validación de fechas	245
Validación de direcciones de e-mail	247
Búsqueda de subcadenas dentro de una página	247
Obtención de parámetros de la URL	247
Manipulado de cadenas	248
Validación de Documentos de Identidad	248
Validaciones básicas	249
Validación de Cuentas Bancarias	249
Validación de Tarjetas de Crédito	250
 CREACIÓN DE LA BASE DE DATOS	 250
 PASOS EN LA CREACIÓN DE LA BASE DE DATOS	 250
Scripts SQL transportables	250
División modular de los scripts SQL/DDI	252
COMANDOS PARA CREAR Y BORRAR LA BASE DE DATOS	252
Crear una base de datos por defecto	252
Crear una base de datos mínima	252
Borrar una base de datos	252
Cómo ejecutar un script SQL contra la base de datos	253
Cómo generar un script SQL a partir de los datos de una tabla	253
Cómo cargar un archivo en una tabla desde línea de comandos	253
 INTEGRACIÓN CON JAKARTA LUCENE	 254
 INTERFAZ DE INDEXACIÓN LUCENE - HIPERGATE	 255
ESTRUCTURA DE LOS DOCUMENTOS INDEXADOS	255
 INTEGRACIÓN CON JAKARTA POI	 256
 INTEGRACIÓN CON LDAP	 256
 CÓMO CREAR UN DIRECTORIO COMPATIBLE CON HIPERGATE	 256
EJEMPLO RÁPIDO DE CONFIGURACIÓN CON OPENLDAP	257
QUÉ INFORMACIÓN SE ALMACENA EN LDAP	258
CÓMO CONECTAR HIPERGATE CON EL DIRECTORIO LDAP	261
Sincronización entre LDAP e hipergate	262
Cómo cargar un dominio o un área de trabajo en LDAP	262
Cómo borrar un área de trabajo	263
Cómo borrar un directorio completo	263
CÓMO ACCEDER AL DIRECTORIO DESDE OUTLOOK EXPRESS	263
Capturas de pantalla de Outlook Express	264
AUTENTIFICACIÓN DE USUARIOS BASADA EN LDAP	269
Cómo autenticar a los usuarios usando LDAP	269
 INTEGRACIÓN DE SEGURIDAD CON NTLM	 270

INSTALACIÓN DE FILTRO DE INTEGRACIÓN NTLM	270
Configuración de hipergate.cnf	271
<u>ACCESO REMOTO A LA BB.DD. POR HTTP</u>	<u>271</u>
FUNCIONALIDADES	271
INSTALACIÓN	272
PARÁMETROS DE ENTRADA	272
LEER DATOS	273
ESCRIBIR DATOS	274
Cómo se graban los datos	275
SEGURIDAD	275
EJEMPLOS ADICIONALES	276
Variables globales y función genérica AjaxPost para enviar peticiones HTTP POST desde VBScript	276
Buscar un contacto dado su e-mail	276
Grabar o actualizar un contacto	277
Grabar una nueva oportunidad	277
Cambiar estado de oportunidad "NUEVA" a "GANADA"	278
<u>INTEGRACIÓN CON GOOGLE DATA</u>	<u>279</u>
CALENDARIO	279
GOOGLE MAPS	280
Cálculo de distancias	281
<u>API DE ACCESO EXTERNO AL CALENDARIO</u>	<u>282</u>
MODELO LÓGICO DEL CALENDARIO.	282
MODELO DE SEGURIDAD	283
Proceso de autenticación	283
INSTALACIÓN EN EL SERVIDOR	283
TIPOS DE DATOS	283
API REST	284
connect	284
disconnect	285
getMeetings	285
getMeetingsForRoom	287
getMeeting	287
getRooms	288
getAvailableRooms	289
isAvailableRoom	289
storeMeeting	290
LIBRERÍA CLIENTE .NET	291
Clase Hipergate.CalendarClient	291
Ejemplo de uso del cliente .NET desde VisualBASIC	295
LIBRERÍA CLIENTE JAVA	296
Ejemplo de uso del cliente desde Java	296

1

Introducción

El arte de programar está tan relacionado con los ordenadores, como la astronomía con la fabricación de telescopios.

E. W. Dijkstra

Capas de la aplicación

hipergate está dividido en 5 capas:

1. Procedimientos Almacenados y Modelo de Datos
2. Clases Java compiladas
3. Scripts Java BeanShell
4. Servlets / JSP
5. JavaScript cliente

Esta guía está estructurada de abajo a arriba, comenzando por el modelo hasta llegar a la capa final de presentación en el cliente, con el propósito de dar una visión completa de cómo escribir ampliaciones funcionales desde cero.

2

Pasos para extender hipergate

Antes de describir el funcionamiento de todas las partes de la aplicación, esta sección proporciona una guía paso a paso sobre cómo añadir extender hipergate.

La adición de una extensión requiere básicamente tres pasos:

1. Añadir los scripts SQL-DDL de creación de tablas, vistas, índices y procedimientos almacenados en la base de datos.
2. Crear las clases Java que forman la capa de acceso a datos de las tablas anteriores.
3. Crear los formularios JSP de listado y edición.

Cómo agregar objetos al modelo de datos

Los scripts de creación de objetos en el modelo de datos se encuentran dentro de la estructura de directorios de las clases Java en

```
com/knowgate/hipergate/datamodel
```

Bajo este directorio se encuentran los subdirectorios:

```
/constraints  
/data  
/drop  
/indexes  
/procedures  
/tables  
/views
```

Algunos de ellos con subdirectorios para cada SGBDR.

Módulos funcionales

Dentro de cada subdirectorio del modelo de datos existen archivos con el nombre de los módulos funcionales del producto. Así en /tables podemos encontrar:

```
/addrbook.ddl  
/billing.ddl  
/categories.ddl
```

/crm.ddl
Etc.

Cada archivo contiene los objetos para el módulo funcional que lleva su nombre.

Los archivos pueden llevar extensión .ddl o .sql

Dentro de los archivos, el separador de sentencias puede ser punto y coma o "GO;" dependiendo del archivo en cuestión. Cuando se añaden nuevos objetos hay que respetar el separador que ya se esté usando.

Tipos independientes del SGBDR

Para la creación de tablas, hipergate usa una lista de tipos independiente del SGBDR.

Hipergate DDL	MySQL	PostgreSQL	SQL Server	Oracle
DATETIME	TIMESTAMP	TIMESTAMP	DATETIME	DATE
VARCHAR	VARCHAR	VARCHAR	NVARCHAR	VARCHAR2
LONGVARCHAR	MEDIUMTEXT	TEXT	NTEXT	CLOB
LONGVARBINARY	MEDIUMBLOB	BYTEA	IMAGE	BLOB
SERIAL	INTEGER	AUTO_INCREMENT	SERIAL	NUMBER(11)

Las definiciones de tablas en los archivos DDL deben utilizar estos tipos en lugar de aquellos nativos del SGBDR. El proceso de creación del modelo de datos de hipergate reemplaza los tipos abstractos por su nombre concreto en el SGBDR que esté siendo utilizado.

Constraints

Para reducir la dependencia de orden en la creación de objetos, todas las constranitns que no sean la clave primaria de la tabla deben declararse independientemente con una sentencia ALTER TABLE en un archivo del subdirectorio /constraints

Borrado de objetos

Por cada nuevo objeto que se agregue a los scripts de creación, hay que añadir su correspondiente sentencia de borrado en el script pertinente del subdirectorio /drop.

Tablas de remonte (lookups)

Hay un convenio general para las tablas de remonte o tablas de valores (lookups). Por cada tabla principal existe una sola tabla de remonte que tiene el mismo nombre de la tabla principal con el sufijo `_lookup`.

Los valores de remonte son diferentes para cada Área de Trabajo.

Los valores de remonte pueden presentarse en pantalla en un idioma u otro dependiendo de las preferencias idiomática que el usuario haya establecido en su navegador web.

Cada tabla de remonte contiene siempre las mismas columnas, que son:

gu_owner: GUID del Área de Trabajo a la cual pertenece el valor de remonte.
id_section: Nombre de la columna en la cual se almacenará el valor en la tabla principal.
pg_lookup: Ordinal identificador del valor.
vl_lookup: Valor interno del lookup almacenado en la tabla principal (no mostrado en pantalla).
tr_xx: Columnas con los textos mostrados en pantalla para los diferentes idiomas.

Cómo crear clases Java de acceso a datos

Existen esencialmente dos clases Java para acceder a los datos, una que representa un único registro `com.knowgate.dataobjs.DBPersist` y otra que representa un *record set* `com.knowgate.dataobjs.DBSubset`.

Se debe crear una subclase de `DBPersist` para cada tabla que no sea un lookup.

`DBSubset` se utiliza directamente y no es preciso crear ninguna subclase.

DBPersist

<http://www.hipergate.org/docs/api/5.0.0/com/knowgate/dataobjs/DBPersist.html>

El uso típico de `com.knowgate.dataobjs.DBPersist` es crear una subclase que implementa la lógica de negocio de acceso a un registro de una determinada tabla.

Prácticamente todos los formularios de mantenimiento de datos de hipergate usan una subclase de `DBPersist`.

DBPersist implementa el interfaz java.util.Map. El comportamiento de DBPersist es esencialmente mapear los valores de este mapa en las columnas del mismo nombre en la tabla designada en la base de datos.

Los valores contenidos en el mapa deben ser objetos de tipos básicos Java: String, Integer, Float, Double, BigDecimal y Date.

DBPersist tiene 6 métodos básicos:

Put: Añade un nuevo valor al mapa.

Replace: Reemplaza un valor en el mapa.

Remove: Elimina un valor del mapa.

Load: Carga un registro de la base datos en el mapa.

Store: Graba los valores del mapa en una fila de una tabla de la BB.DD.

Delete: Borra un registro de la base de datos.

La implementación más simple de una subclase suele incluir alguna acción en los métodos Load, Store o Delete.

```
import com.knowgate.dataobjs.DBPersist;

import java.sql.SQLException;
import com.knowgate.jdbc.JDCCConnection;
import com.knowgate.misc.Gadgets;

public class MyTable extends DBPersist {

    public ContactShortCourse() {
        super("k_my_table", "MyTable");
    }

    public boolean store(JDCCConnection oConn)
        throws SQLException {
        if (!AllVals.containsKey("gu_mytable")) {
            put("gu_mytable", Gadgets.generateUUID());
        }
        return super.store(oConn);
    }
}
```

DBSubset

<http://www.hipergate.org/docs/api/6.0.0/com/knowgate/dataobjs/DBSubset.html>

Para acceder a conjuntos de filas hipergate usa la clase com.knowgate.dataobjs.DBSubset que carga dichas filas en una matriz bidimensional en memoria mediante una única ráfaga de lectura.

La clase DBSubset se utiliza principalmente en los formularios de listado.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

Mantenimiento de sesiones

Las sesiones en hipergate se mantienen mediante cookies.
La aplicación no mantiene estados en el lado del servidor.
En la página /common/login_chk.jsp se establecen las siguientes cookies de sesión:

profilenm: Nombre sin extensión del fichero .cnf del cual se leen las propiedades de conexión a la base de datos.
domainid: Identificador numérico del dominio al cual se está conectado.
workarea: GUID del Área de Trabajo a la cual se halla conectado el usuario.
userid: GUID del usuario conectado.
appmask: Máscara de bits que indica los permisos del usuario sobre las aplicaciones.
skin: Hoja de estilo CSS que se usará para la presentación.

Plantillas de formularios de ejemplo

En el subdirectorio /examples de la webapp estándar de hipergate pueden encontrarse varias plantillas de ejemplo para construir páginas de edición y listado de datos.

A continuación revisaremos paso a paso el contenido de dichas plantillas:

void.jsp

Juego de caracteres UTF-8

void.jsp es la página JSP más sencilla de todas. Observar en ella que el juego de caracteres es UTF-8
`contentType="text/plain;charset=UTF-8"` este es el estándar de hipergate para todas sus páginas JSP.

Bean de aplicación para acceder al pool de conexiones a la BB.DD.

La página tiene un include que carga el bean de aplicación GlobalDBBind
`<%@ include file="../methods/dbbind.jsp" %>`
GlobalDBBind es el punto de acceso al pool de conexiones a la base de datos a través de su método `getConnection("...")`. Requiere un nombre para cada conexión solicitada. Este nombre es totalmente

arbitrario y se usa sólo para identificar la conexión en el pool y con propósitos de depuración.

Captura de excepciones

Se deben capturar siempre todas las excepciones.

☞ Para garantizar que no se quedan conexiones abiertas con la base de datos llamar siempre al método `disposeConnection()` en el manejador de las excepciones.

form_edit.jsp

Es la plantilla de página JSP para editar un registro en una tabla.

Verificación de credenciales de seguridad del usuario

Entre los archivos externos puede encontrarse:

```
<%@ include file="../methods/authusrs.jspf" %>
```

que es el archivo que contiene el método `authenticateSession` utilizado en todas las páginas JSP para validar la sesión de usuario mediante la sentencia:

```
if (authenticateSession(GlobalDBBind, request, response)<0) return;
```

☞ Es preciso añadir esta sentencia al inicio de todas las páginas JSP ya que es la que comprueba las cookies y verifica que la petición es legítima.

Idioma por defecto

Existe un método especial para detectar el idioma del navegador cliente:

```
final String sLanguage = getNavigatorLanguage(request);
```

Este método devuelve un código de idioma de dos letras minúsculas.

Se utiliza principalmente para determinar en qué idioma deben mostrarse los valores de las tablas de remonte (lookups).

Parámetros

Los parámetros utilizados por la página pueden llegar por GET o POST en el objeto `request` o ser leídos de las cookies de sesión.

En el ejemplo de `form_edit.jsp` el usuario actual se lee de la cookie de sesión pero el área de trabajo debe llegar como parámetro en el `request`.

Instancia de una subclase de DBPersist

Los formularios de edición de datos trabajan con una subclase de `DBPersist` para leer los campos de la base de datos. En `form_edit.jsp` se utiliza la clase `com.knowgate.example.Example`

La subinstancia de DBPersist se carga mediante una llamada a su método load() si y sólo si el parámetro que identifica el GUID del objeto a cargar no es null.

```
if (null!=gu_example) oObj.load(oConn, new Object[]{gu_example});
```

En el caso de que el parámetro gu_example sea null ello significa que el formulario está siendo abierto para crear un nuevo registro y, por consiguiente, no procede intentar cargarlo de la base de datos.

Lectura de los valores de remonte

En la clase com.knowgate.hipergate.DBLanguages existen métodos especiales para leer valores de las tablas de remonte y presentarlos en tags SELECT de HTML. En el caso de form_edit.jsp se usa

```
sTypeLookUp = DBLanguages.getHTMLSelectLookUp  
(GlobalCacheClient, oConn, "k_examples_lookup", gu_workarea,  
"tp_example", sLanguage);
```

Esto lee los valores de la sección tp_example en la tabla

k_examples_lookup y los almacena en un String de opciones tipo

```
<OPTION VALUE="...">...</OPTION>
```

Para abrir el pop-up de adición de nuevos valores de remonte se usa la función JavaScript lookup(). Esta función toma como parámetro de entrada un entero arbitrario que identifica un objeto <SELECT> en la página principal y un <INPUT TYPE="hidden"> para el valor mostrado y el valor interno del remonte respectivamente.

Skin CSS

La hoja de estilo CSS que usará el formulario se establece por JavaScript:

```
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript"  
SRC="../../javascript/setskin.js"></SCRIPT>
```

Calendario

Todas las fechas se introducen en formato YYYY-MM-DD

Existe un pop-up especial para mostrar un calendario y elegir un día que asignar a un campo.

La función JavaScript showCalendar() muestra un ejemplo de cómo rellenar una fecha para un objeto HTML de tipo INPUT que entra como parámetro en la función.

Validación de datos

Todas las páginas validan los datos en el evento onSubmit del formulario llamando a la función JavaScript validate(). Esta función es diferente para cada JSP y debe ser escrita por el desarrollador.

Las librerías principales de validación de datos están en los siguientes archivos del subdirectorio /javascript

datefuncs.js: Validación de fechas.

email.js: Validación de sintaxis de direcciones de e-mail.

simplevalidations.js: Validación de cantidades numéricas, NIFs, etc.

Posicionamiento en la opción correcta de cada combo

Todas las páginas llaman a la función JavaScript `setCombos()` en el evento `onLoad` para posicionar los objetos de tipo `<SELECT>` en el valor correcto mediante la función `setCombo()` del archivo `combobox.js`

Rellenar los valores para cada `<INPUT>`

Los objetos `<INPUT>` de HTML deben llamarse igual que las columnas de la tabla en la base de datos en las cuales vayan a ser insertados.

```
<INPUT TYPE="text" NAME="nm_example" MAXLENGTH="50"
VALUE="<%=oObj.getStringNull("nm_example","")%>">
```

☞ Hay que poner objetos `<INPUT>` tanto para los campos visible y editables por el usuario como para las claves internas

```
<INPUT TYPE="hidden" NAME="gu_example"
VALUE="<%=oObj.getStringNull("gu_example","")%>">
```

simpleform.jsp

Esta página es un ejemplo de un formulario de mantenimiento más sencillo que `form_edit.jsp` para un registro de la base de datos. Los formularios de mantenimiento de hipergate sirven igualmente para crear nuevos registros como para actualizar datos en registros ya existentes. El patrón de funcionamiento es que cada formulario de mantenimiento hace POST contra una página de su mismo nombre pero terminada en `_store.jsp`.

Paso a paso, las acciones de la página `simpleform.jsp` son:

1. Verificar la validez de las credenciales del usuario.
2. Añadir cabeceras para impedir que la página se mantenga en caché.
3. Obtener los parámetros de entorno y la clave primaria del objeto a editar (si es un objeto ya existente).
4. Crear una instancia de la subclase de `DBPersist` que se utilice para cargar los valores de los campos.
5. Obtener una conexión del pool. El nombre para la conexión debe cambiarse para que el recolector de estadísticas pueda identificar unívocamente la procedencia de la petición.
6. Obtener los valores de lookup para la tabla editada usando el método estático `DBLanguages.getHTMLSelectLookUp()`.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

7. Función JavaScript para mostrar el pop up de calendario. Esta función recibe como parámetro el nombre del campo donde colocar la fecha seleccionada.
8. Función JavaScript para abrir las páginas de selección y/o creación de valores de remonte.
9. Función de validación previa a la remisión del formulario.
10. Menú de acciones comunes.
11. Ejemplo de campos obligatorios.
12. Ejemplo de campos opcionales.
13. Ejemplo de campos de selección de valores de remonte.
14. Ejemplo de campos fecha con selección en calendario.

form_store.jsp

Es la plantilla de página JSP para grabar un registro en la base de datos. form_store.jsp recibe los datos por HTTP POST de form-edit.jsp

Carga de parámetros del request en un DBPersist

En el archivo

```
<%@ include file="../../methods/reqload.jspf" %>
```

Se incluyen métodos para cargar directamente los parámetros de un request HTTP en una instancia de DBPersist.

En particular

```
loadRequest(oConn, request, oObj);
```

carga los valores del request en la instancia de DBPersist representada por oObj.

Grabación del DBPersist en la base de datos

Lo lleva a cabo el método `store()` de la subinstancia de DBPersist.

Auditoría de operaciones

En la clase `com.knowgate.dataobjs` existe un método especial para grabar auditoría de operaciones llamado [DBAudit](#).

listing.jsp

Es la plantilla de página JSP para presentar listados y resultados de búsquedas en la base de datos.

Los listados se basan en la clase `com.knowgate.dataobjs.DBSubset`

Los listados no tienen estado. Se pagina mediante el parámetro skip de la query string de listing.jsp que indica cuántas filas hay que saltar a partir de la primera al leer los registros. Es decir, la misma sentencia SQL se lanza contra la base de datos cada vez que se pulsa Anteriores o Siguientes en la página de listado.

Todas las cláusulas WHERE de los DBSubset de los listados deben filtrar siempre los resultados para limitarlos sólo a aquellos pertenecientes al Área de Trabajo Actual.


Consultas almacenadas

Desde los listados es posible lanzar consultas SQL previamente almacenadas. Esto se realiza mediante el objeto `com.knowgate.hipergate.QueryByForm`

Los archivos XML de definición de las consultas SQL se almacenan en el subdirectorío /qbf del directorio storage de hipergate.cnf

Es posible especificar un filtro para recuperar sólo las filas que cumplan un determinado criterio y también paginar hacia delante y hacia atrás a través de los resultados mostrando sólo un número limitado de filas por página.

Paso a paso, las acciones de la página `listing.jsp` son:

1. Obtener el idioma del navegador cliente.
2. Obtener el decorado (skin) actual.
3. Leer las cookies para el Dominio y Área de Trabajo actual.
4. Leer el parámetro de resolución de pantalla en el cliente, o asumir 800x600 si el parámetro no existe.
 -  No es posible obtener la resolución de pantalla cliente desde JSP de ninguna forma sin pasarla explícitamente como parámetro.
5. Leer las cláusulas de filtrado de registros que pueden ser de 2 tipos.
 - 5.1 Cláusula WHERE libre u obtenida de un [QBF](#) que se concatena al filtro base de registros para el Área de Trabajo actual en la sentencia SQL de recuperación de filas.
 - 5.2 Par {Campo, Valor} para buscar las filas con uno de sus campos asignados a un patrón específico. El valor buscado se concatena como '%Valor%' y se busca con un operador LIKE en el campo especificado.
6. Leer el máximo número de filas por página y la fila de inicio.

☞ Dado que el modelo relacional no garantiza el orden de recuperación de filas, el desplazamiento de registros recargando la página y repitiendo la sentencia SQL con un desplazamiento que realiza `listing.jsp` no es realmente consistente a menos que se especifique un orden para los resultados. De una sentencia a otra las filas recuperadas pueden (potencialmente) cambiar haciendo que aparezcan registros de nuevo o que otros no se listen en ninguna página.

7. Leer la columna por la que se deben ordenar los resultados.
8. Obtener una conexión del pool. El nombre `"instancelisting"` para la conexión debe cambiarse para que el recolector de estadísticas pueda identificar unívocamente la procedencia de la petición.
9. Si la página recibió el parámetro `where` entonces aplicar dicho filtro a la sentencia SQL de recuperación de registros.
10. Si la página recibió los parámetros `field` y `find` entonces recuperar sólo las filas cuyo campo cumpla el patrón especificado.
11. Si no hay filtro recuperar todos los registros hasta el máximo de filas permitido por página.
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. Definir la llamada a la página para eliminar instancias. Hay que cambiar el archivo `"instance_edit_delete.jsp"` por el nombre del formulario de creación pertinente. Durante el proceso de carga, las claves primarias de las filas se almacenan en un Array JavaScript. Cuando se llama al método `deleteInstances()` se comprueba qué checkboxes están marcadas y se escribe la clave primaria de esas filas en el campo oculto `checkeditems`; este campo de claves primarias separadas por comas es el que se envía por POST a la página de eliminación de instancias.
14. Definir la llamada a la página para modificar una nueva instancia. Hay que cambiar el archivo `"instance_edit.jsp"` por el nombre del formulario de creación pertinente.
15. Definir la función para recargar la página ordenando por un campo diferente.
16. Definir la función para seleccionar/ deseleccionar todas las instancias.
17. Definir la función para recargar la página buscando un patrón en un campo.
18. Definir la llamada a la página de clonado de instancia. Para clonar una instancia es necesario haber escrito previamente un archivo XML de definición de estructura de datos como los que se encuentran en el

subdirectorio /storage/datacopy. Los parámetros para la página /common/clone.jsp son:

id_domain	Identificador numérico del dominio actual
nm_domain	Nombre del numérico del dominio actual.
datastruct	Nombre (sin extensión) del archivo XML que contiene la definición de la estructura de datos a clonar.
gu_instance	GUID de la instancia Origen a clonar.
opcode	Código de Operación para auditoría.
classid	Identifiacor numérico de la clase de objteo clonado.

Una vez que se ha abierto la página clone.jsp, se crea un temporizador con una función de espera activa que busca la ventana del proceso de clonado cada 100 milisegundos. En el momento en que el proceso de clonado termina y su ventana se cierra automáticamente, la página de listado se recarga para posicionarse con un filtro por campo en la nueva instancia clonada.

19. Pintar el menú superior con las caas de búsqueda.
20. Pintar los enlaces de Anterior y Siguiente para la paginación.
21. Pintar las filas leídas de la base de datos. Se procesa el evento onclick para abrir el formulario de modificación de instancia y el evento oncontextmenu para ver el menú contextual.
22. Establecer el menú de botón derecho contextual para las filas.

Carga de fechas desde textos de formularios en la base de datos

Por convenio, todas las fechas de los formularios de hipergate tienen el formato YYYY-MM-DD HH24:MI:SS independientemente del idioma en el que funcione la aplicación.

Se sugieren 4 formas estándar de cargar fechas desde texto de formularios en la base de datos que llamaremos *Cast*, *Split*, *Parse* y *Escape* respectivamente :

1^a) Cast. Producir una conversión de tipos para concatenar en una sentencia de SQL dinámico dependiente del gestor.

```

import java.sql.Statement;

import com.knowgate.jdbc.JDBCConnection;
import com.knowgate.dataobjs.DBBind;

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

// Get Connection from Pool
JDBCConnection oCon = oDBB.getConnection("cast_date");

String sDateTime = "2003-09-18 02:38:00";
String sDateDB;

switch (oCon.getDataBaseProduct()) {

    case JDBCConnection.DBMS_ORACLE:
        sDateDB = "TO_DATE(' " + sDateTime + "', 'YYYY-MM-DD
HH24:MI:SS')";
        break;

    case JDBCConnection.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. Convertir una fecha corta de entrada en un Timestamp de JDBC.

```

import java.sql.PreparedStatement;
import java.sql.Timestamp;

import java.util.Date;

import com.knowgate.jdbc.JDBCConnection;
import com.knowgate.misc.Gadgets;

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

// Get Connection from Pool
JDBCConnection 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());

PreparedStatement oStm = oCon.prepareStatement("UPDATE k_users SET
dt_modified=?");

oStm.setTimestamp (1, oTs);

oStm.executeUpdate ();

oStm.close();

oCon.close("split_date");

```

3ª) Parse. Convertir una fecha y hora en un Timestamp de JDBC.

```

import java.text. SimpleDateFormat;

import java.sql.PreparedStatement;
import java.sql.Timestamp;

import com.knowgate.jdbc.JDCCConnection;
import com.knowgate.misc.Gadgets;

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

// Get Connection from Pool
JDCCConnection 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());

PreparedStatement oStm = oCon.prepareStatement("UPDATE k_users SET
dt_modified=?");

oStm.setTimestamp (1, oTs);

oStm.executeUpdate ();

oStm.close();

oCon.close("parse_date");

```

4ª) Escape. Usar el método escape de la clase DBBind con el parámetro “ts” para un Timestamp o “d” para una fecha corta.

```

import java.sql.Statement;

import java.util.Date;

import com.knowgate.jdbc.JDBCConnection;
import com.knowgate.dataobjs.DBBind;

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

// Get Connection from Pool
JDBCConnection oCon = oDBB.getConnection("escape_date");

// Get different escape or cast date strings depending on witch
// 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");

```

Prevención de ataques por cross-site scripting e inyección SQL

Los ataques por cross-site scripting (XSS) e inyección SQL se producen típicamente insertando código JavaScript o SQL en los campos de entrada de los formularios HTML.

Un ataque XSS puede tener por ejemplo la forma

```
"><script>document.write("Hola Mundo")</script>
```

Cuando el código JSP intenta insertar esta cadena recibéndola como parámetro en el request puede hacer algo como:

```
<INPUT TYPE="text" VALUE="<%=request.getParameter("hola")%>">
```

Dando como resultado una salida de la forma

```
<INPUT TYPE="text" VALUE=""><script>document.write("Hola
Mundo")</script>>
```

La cual ejecutará el código JavaScript que escribe Hola Mundo en el documento.

El ataque por inyección SQL es similar, excepto que lo que se manda en el request es código SQL en vez de JavaScript.

Hay tres medidas que se pueden tomar para prevenir los ataques por inyección de JavaScript y SQL:

1ª) Impedir mediante JavaScript que el usuario pueda teclear código ejecutable en los campos de entrada de los formularios. Esto puede hacerse de varias maneras, una muy sencilla es validar las entradas con la función JavaScript [hasForbiddenChars\(\)](#) que impide, entre otras cosas, que se introduzcan comillas simples o dobles en el texto.

2ª) Verificar con expresiones regulares que los parámetros de entrada en el request de una página no son código JavaScript o SQL. Esto puede hacerse con código Java como el de la función [hasXssSignature\(\)](#).

3ª) Escribir todas las salidas al documento generado como entidades HTML codificadas. Esto puede hacerse empleando el método `getStringHtml()` de las clases `com.knowgate.dataobjs.DBPersist` y `com.knowgate.dataobjs.DBSubset`

Cambio del skin

El look'n feel skin de hipergate lo determina el archivo `styles.css` ubicado en un subdirectorio del directorio `/skins` de la webapp.

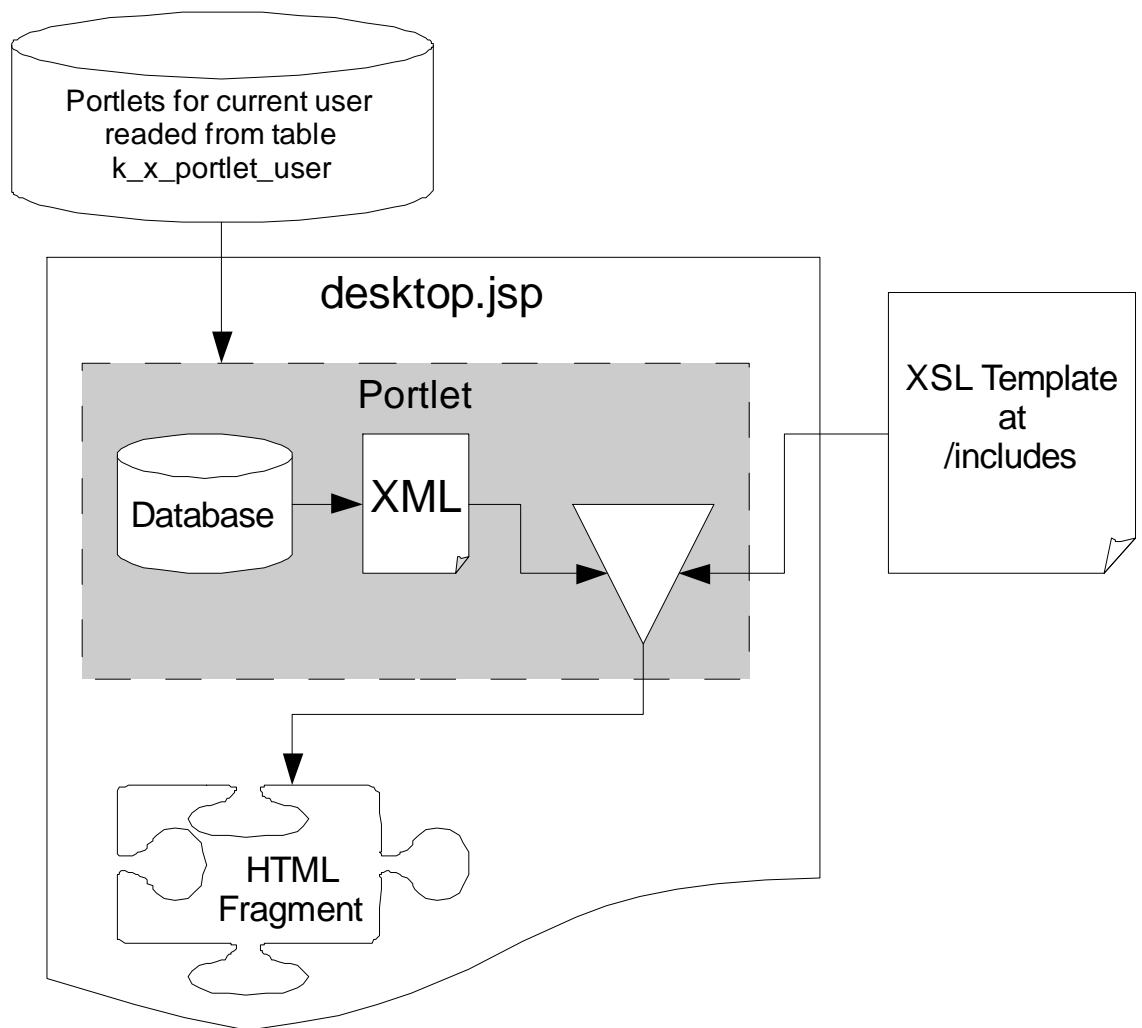
El skin predeterminado se puede cambiar editando el archivo `hipergate.cnf` y el JavaScript de la página de inicio `login.html`

Añadir un portlet a la página de inicio

La página de inicio de hipergate (`/common/desktop.jsp`) permite mostrar contenidos dinámicos en dos columnas basándose en clases que implementen el interfaz `javax.portlet.GenericPortlet` interface.

Los contenidos mostrados en la página de inicio de hipergate pueden ser diferentes para cada usuario. La tabla `k_x_portlet_user` de la base de datos contiene la lista de contenidos (portlet) que se deben mostrar para cada usuario de forma individualizada. Cada portlet genera un fragmento de XHTML que se muestra en la página principal.

Diagrama de cómo se incluyen los portlets en la página de inicio



Los portlets estándar de hipergate se encuentran en el paquete `com.knowgate.http.portlets` de las librerías base Java.

He aquí un ejemplo de un portlet Hola Mundo!

```
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+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())

```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```

        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");

    JDCCConnection oCon = null;

    try {
        oCon = oDBB.getConnection(getClass().getName());

        // This is the data retrieved 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

```

Este portlet usa la siguiente hoja de estilo XSL que debe estar en el subdirectorio /includes de la webapp de hipergate.

```

<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" />

```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```

<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/{ $param_skin }/tab/angle45_24x24.gif"
          WIDTH="24" HEIGHT="24" BORDER="0" /></TD>
    </TR>
  </TR>
  <TR>
    <TD CLASS="subtitle">
      <xsl:if test="$param_windowstate='NORMAL'">
        <A
          HREF="windowstate.jsp?gu_user={ $param_user }&nm
            _page=desktop.jsp&nm_portlet=com.knowgate.http
            .portlets.HelloWorld&gu_workarea={ $param_worka
            rea }&nm_zone={ $param_zone }&id_state=MINIMI
            ZED"><IMG
              SRC=" ../skins/{ $param_skin }/tab/minimizel2.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 }&nm
              _page=desktop.jsp&nm_portlet=com.knowgate.http
              .portlets.HelloWorld&gu_workarea={ $param_worka
              rea }&nm_zone={ $param_zone }&id_state=NORMAL
              "><IMG
                SRC=" ../skins/{ $param_skin }/tab/maximizel2.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')">
                <A href="#">New Item</A>
            </TD>
          </TR>
        </TABLE>
      </TD>
    </TR>
  </TABLE>
  </TD>
</TR>
<TR>
  <TD COLSPAN="2">

    <!-- *** Content HERE *** -->

    HOLA MUNDO!

  </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>

```

Los portlets no tienen porqué estar basados necesariamente en hojas de estilo XSL. La salida HTML se puede generar directamente desde la clase Java o por cualquier otro método.

Guía paso a paso para añadir un nuevo portlet

1. Escribir una subclase de GenericPortlet siguiendo el ejemplo del HelloWorld portlet descrito anteriormente en este documento.
2. Escribir una plantilla XSL.
3. Hacer que el método render() de la subclase de GenericPortlet devuelva el HTML final mediante una transformación XSLT.
4. Poner la hoja de estilo XSL en el subdirectorio /includes de la webapp.
5. La hoja de estilo XSL template debe de pintar el portlet en estado normal o minimizado dependiendo del valor del parámetro param_windowstate : NORMAL o MINIMIZED. Para ello añadir un par de tags <xsl:if> a la hoja XSL:

```

<xsl:if test="$param_windowstate='NORMAL'">
    <A
HREF="windowstate.jsp?gu_user={$param_user}&nm_p
age=desktop.jsp&nm_portlet=com.knowgate.http.por
tlets.HelloWorld&gu_workarea={$param_workarea}&
mp;nm_zone={$param_zone}&id_state=MINIMIZED"><IM
G 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}&nm_p
    age=desktop.jsp&nm_portlet=com.knowgate.http.por
    tlets.HelloWorld&gu_workarea={$param_workarea}&a
    mp;nm_zone={$param_zone}&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. Editar la página /common/desktop_custom.jsp y en los arrays JavaScript llamados portlets, labels y enabled añadir el nombre de la subclase de GenericPortlet, una etiqueta legible por el usuario y qué modulo de hipergate debe estar activado para que el portlet sea visible.
7. Conectarse a la aplicación y en el enlace que hay al pie de la página de inicio seleccionar los portlets que se desea visualizar en cada columna.

3

Modelo de Datos

hipergate se basa en un modelo de datos relacional escrito para ser transportable entre diferentes sistemas gestores de base de datos.

La transportabilidad se combina con un esfuerzo especial por aprovechar las mejores características nativas de cada gestor.

Dónde encontrar los fuentes del modelo

Los fuentes originales del modelo de datos están almacenados como recursos dentro del archivo hipergate.jar que contiene las clases de la aplicación.

Hay que buscarlos en el paquete **com.knowgate.hipergate.datamodel**. Normalmente no es necesario acceder directamente a estos fuentes, excepto en el caso de que se esté escribiendo una versión de hipergate para un nuevo sistema gestor de base de datos.

El modelo completo está dividido en submodelos lógicos que representan partes de la aplicación. Algunos submodelos básicos son requeridos

siempre para que cualquier aplicación de la suite pueda funcionar, como el submodelo de seguridad o el submodelo de categorías. Otros submodelos pueden instalarse o no en función de si van a instalarse las aplicaciones de la suite que los utilizan.

Convenciones

Nomenclatura

La nomenclatura del modelo de datos está escrita en inglés.

Todos los nombres de objetos se escriben en minúsculas.



Los nombres de objetos se convierten automáticamente a mayúsculas al crearlos en una base de datos Oracle. Las librerías de hipergate no son sensibles a mayúsculas/minúsculas, pero otras librerías o consultas SQL hechas a medida pueden si serlo.

Tablas y Vistas

Tablas Las tablas comienzan por el prefijo “k_”.

Campos Se utilizan los siguientes prefijos según el concepto que el campo represente:

“gu_”	: El campo es un identificador único de registro a nivel global.
“id_”	: Identificador de objeto.
“tx_”	: Texto.
“de_”	: Descripción de un objeto.
“tl_”	: Título de un objeto.
“dt_”	: Fecha.
“nm_”	: Nombre de objeto.
“tp_”	: Tipo de objeto.
“od_”	: Posición ordinal dentro de una lista.
“tr_”	: Texto traducido para algún idioma en particular.
“pg_”	: Campo numérico entero progresivo (como una secuencia).
“bo_”	: Flag booleano (1=verdadero, 0=falso).
“is_”	: Indicador (si/no).
“nu_”	: Número, cuenta de objetos.
“ny_”	: Número, cuenta de años transcurridos.
“nd_”	: Número, cuenta de días transcurridos.
“pr_”	: Precio, unidades monetarias.
“im_”	: Importe, unidades monetarias.

“pct_” : Porcentaje.

Ejemplo (SQL Server):

```
CREATE TABLE k_companies (  
gu_company      CHAR(32)      NOT NULL, /* Identificador único de compañía */  
dt_created      DATETIME      DEFAULT GETDATE(), /* Fecha de creación */  
nm_legal        VARCHAR(50) NOT NULL, /* Razón Social */  
gu_workarea     CHAR(32)      NOT NULL, /* GUID de la workarea */  
nm_commercial   VARCHAR(50)   NULL, /* Nombre comercial */  
dt_modified     DATETIME      NULL, /* Fecha de modificación */  
id_legal        VARCHAR(16)   NULL, /* NIF/CIF */  
id_sector       VARCHAR(16)   NULL, /* Código sectorial */  
id_status       VARCHAR(30)   NULL, /* Estado:activa,quiebra...*/  
id_ref          VARCHAR(50)   NULL, /* Identificador externo */  
tp_company      VARCHAR(30)   NULL, /* Tipo de compañía: */  
de_company      VARCHAR(254)  NULL, /* Descripción Actividad */  
  
CONSTRAINT pk_companies PRIMARY KEY (gu_company),  
CONSTRAINT ul_companies UNIQUE (gu_workarea,nm_legal))
```

Vistas Las vistas comienzan por el prefijo “v_”.

Procedimientos Los procedimientos comienzan por el prefijo “k_sp_”.

Disparadores Los disparadores comienzan por el prefijo “k_tr_”.

Tipos de datos

Por cuestiones de transportabilidad entre gestores de base de datos, sólo se usan los siguientes tipos de datos:

Tipo base	SQL Server	Oracle	PostgreSQL
Entero con signo de 16 bits	SMALLINT	NUMBER(6)	SMALLINT
Entero con signo de 32 bits	INTEGER	NUMBER(11)	INTEGER
Caracteres longitud fija (ASCII)	CHAR	CHAR	CHAR
Caracteres long. Variable (ASCII)	CHARACTER VARYING	CHARACTER VARYING	CHARACTER VARYING
Caracteres long. Variable (Unicode)	NVARCHAR	VARCHAR2 *	VARCHAR
Caracteres long. Indefinida (Unicode)	NTEXT	CLOB	TEXT
Fecha y Hora	DATETIME	DATE	TIMESTAMP
Decimal	DECIMAL(n,m)	NUMBER(n,m)	DECIMAL(n,m)
Coma flotante doble precisión	FLOAT	NUMBER	FLOAT
Binario long. indefinida	IMAGE	BLOB	BYTEA



En Oracle no se soporta Unicode de forma parcial con campos NVARCHAR2. Toda la base de datos debe estar creada en Unicode, de modo que los campos VARCHAR2 se almacenen internamente como Unicode. Para una guía sobre cómo crear bases de datos Oracle con soporte multi-idioma véanse los documentos [Creating an Oracle Database](#) y [Enabling multilingual support with Unicode databases](#).



El paquete de acceso a datos es genérico y, en principio, no imponen ninguna restricción sobre los tipos de datos. No obstante, en el proceso de verificación de la suite sólo se han testeado los tipos de la tabla anterior, por lo que no existe ninguna garantía de que funcione con otros tipos de datos.

Identificadores únicos de registro (GUIDs)

Por convenio, casi todas las tablas de la aplicación utilizan un formato común de identificador único de registro de tipo CHAR(32).

El código fuente hace a menudo referencia a estos identificadores como GUIDs (Global Unique Identifiers).

El valor del GUID está garantizado para identificar unívocamente el registro en toda la base de datos y en otras bases de datos.

El prefijo de campo para los GUIDs es "gu_".

Hay varias formas de generar GUIDs:

1. **Desde línea de comandos :**

```
$>java com.knowgate.misc.Gadgets uuidgen
```

2. **Desde código Java :**

```
com.knowgate.misc.Gadgets.generateUUID();
```

3. **Con Microsoft Visual Studio :**

Puede usarse el programa UUIDGEN.EXE incluido dentro de los archivos comunes de Visual Studio.

4. **Mediante el API de Win32 :**

Usar la función UuidCreate() contenida en la librería de enlace dinámico RPCRT4.DLL. Las declaraciones necesarias desde Visual BASIC son

```
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. **Desde JavaScript**

Usar una función para generar una cadena aleatoria de 32 caracteres:

```
function createUuid() {
    var chrset = "abcdefghijklmnopqrstuvwxyz23456789";
    var guid = "";
    for (var i=0; i<32; i++) {
        guid += chrset.charAt(Math.round
            (Math.random()*30));
    } // next
    return guid;
}
```

6. Desde PL/pgSQL

Usar una función para generar una cadena aleatoria de 32 caracteres:

```
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'
```

Fechas de creación y modificación de registro

Muchas tablas del modelo utilizan los campos `dt_created` y `dt_modified` para indicar las fechas de creación y modificación de cada registro.

Por convenio, las fechas de creación no hay que escribirlas nunca desde las aplicaciones cliente y se rellenan automáticamente mediante valores por defecto desde la base de datos. Por consiguiente, las fechas de creación contienen siempre la fecha de creación del registro en la base de datos. El campo `dt_created` es obligatorio en aquellas tablas en las que exista y debe ser declarado como NOT NULL.

Las fechas de modificación se escriben desde las aplicaciones cliente. El campo `dt_modified` es opcional y si está puesto a NULL deberá entenderse que el registro no ha sido modificado nunca desde su creación. Debido a que la fecha de creación se rellena desde la base de datos, pero la fecha remodificación se rellena desde las aplicaciones, pueden producirse inconsistencias ellas si la fecha del servidor de base de datos es diferente de la fecha de la máquina que ejecuta la aplicación que graba el registro.

Indicadores de activación de registros

Para marcar registros activos o no activos se utiliza el campo `bo_active`, de tipo SMALLINT (NUMBER(6) en Oracle).

El valor 1 indica que el registro está activo, el valor 0 que está inactivo.

Desactivar registros es una forma útil de borrado lógico.

Recorrido de datos jerárquicos

Un problema común en el diseño de aplicaciones sobre bases de datos relacionales es la representación de datos con estructura jerárquica. En hipergate esta situación se presenta en el árbol de categorías, en el árbol de proyectos y en los tesauros.

No existe una solución única y óptima para este problema.

Si el número máximo de niveles es conocido y limitado a priori, lo más eficiente suele ser listar todos los antecesores de un objeto en el mismo registro del objeto (así trabajan los tesauros de hipergate) o emplear un número determinado de JOINS en las consultas de búsqueda de ascendientes/descendientes.

Si no hay límite en el número de niveles permitidos la mejor solución depende del SGBDR utilizado.

Las tres tácticas más comunes consisten en:

1ª) Emplear tablas temporales para los resultados intermedios del proceso de recorrido del árbol. Esta es la táctica utilizada, por ejemplo, en la versión de SQL Server del procedimiento almacenado `k_sp_cat_expand`.

2ª) Emplear una pila en memoria para el proceso de exploración.

3ª) Hacer llamadas recursivas al mismo procedimiento. Ver por ejemplo, en la versión SQL Server de `k_sp_cat_del_grp`.

4ª) Tratar el árbol como un conjunto de subconjuntos anidados. Esta técnica proporciona la mejor forma de hallar todos los hijos de un nodo dado sin un proceso de expansión previo como el que usar los procedimientos de hipergate. Una buena referencia sobre la técnica puede encontrarse en el libro de Joe Celko *SQL for Smarties*, o en su artículo http://www.intelligententerprise.com/001020/celko1_1.shtml de Octubre de 2000 en Intelligent Enterprise.

5ª) Escribir el proceso de expansión en una subrutina externa C o Java. Ver, por ejemplo, el método `browse()` de la clase `Java.com.knowgate.hipergate.Category`.

Oracle

Oracle resuelve el problema de las jerarquías de forma nativa mediante la cláusula `START WITH ... CONNECT BY ...`

PostgreSQL

Es posible encontrar una función C de expansión de Joe Conway en:

<http://developer.postgresql.org/docs/pgsql/contrib/tablefunc>

Microsoft SQL Server

SQL Server no proporciona ningún mecanismo

Versionado del modelo de datos

La versión del modelo de datos instalada puede consultarse en el campo `vs_stamp` de la tabla `k_versión` a partir de la build 1.1.0.

Segmentación de datos por áreas de trabajo

Por convenio, se utiliza un campo de tipo GUID para identificar el Área de Trabajo a la que pertenecen los datos en todas las tablas que puedan contener registros compartidos de varias entidades clientes.

Este campo de filtrado se llama `gu_workarea` o a veces también `gu_owner`.

El filtrado por el campo `gu_workarea` es el método que utiliza `hipergate` para mostrar sólo los datos de una determinada área de trabajo.

☞ Si se están desarrollando aplicaciones multi-entidad, el manejo del campo `gu_workarea` es absolutamente fundamental para el funcionamiento de la aplicación. Además de añadir el campo en todas las tablas principales de cada submodelo, es esencial componer todas las consultas SQL teniendo en cuenta este campo como criterio obligatorio de filtrado de registros.

La lista de áreas de trabajo disponible se almacena en la tabla **`k_workareas`**.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

Tablas de valores estáticos globales

Su propósito es almacenar valores que no cambian en el tiempo.

Existen dos clases de tablas de valores: 1ª) las tablas de valores globales y 2ª) las tablas de valores por área de trabajo.

Las tablas de valores estáticos globales almacenan datos comunes a todas las áreas de trabajo. Las tablas de valores estáticos por área de trabajo almacenan valores que sólo son visibles para los usuarios de dicha área de trabajo.

Las tablas de valores estáticos globales empiezan por el prefijo “k_lu_”.

Tablas

k_lu_languages Códigos de idioma ISO 639.

k_lu_countries Códigos de país ISO 3166.

k_lu_currencies Códigos de moneda ISO 4217.

k_lu_states Subdivisiones en provincias y estados por país.

k_lu_status Estados genéricos de productos y documentos.
Los estados predefinidos son:
-2 → Retirado.
-1 → Corrupto.
0 → Pendiente de Aprobación.
1 → Activo.
2 → Bloqueado.

k_lu_prod_types Tipos de archivo MIME.

k_lu_cont_types Tipos de Contenedor.
Los valores estándar son.

1	Sistema de Ficheros local (protocolo file://)
2	Hiperenlace (protocolo http://)
3	Hiperenlace (protocolo https://)
4	Ficheros remotos (protocolo ftp://)
5	Base de datos (protocolo jdbc://)

6	Lotus Notes (protocolo notes://).
100	Ubicación física (protocolo ware://)

Tablas de remonte por área de trabajo

Estas tablas contienen valores que sólo son visibles dentro de un área de trabajo concreta.

Cada tabla de remonte siempre va asociada a una tabla base. Siguiendo una forma estándar de creación de remotes es posible fabricar unos pocos formularios de mantenimiento genéricos que sirven para todas las tablas de remonte de la suite.

Las tablas de remonte se llaman como su tabla base, pero con el sufijo “_lookup”.

Ejemplo:

Tabla Base

```
CREATE TABLE k_companies
(
  gu_company      CHAR(32) NOT NULL, /* GUID de la compañía */
  gu_workarea     CHAR(32) NOT NULL, /* GUID de la workarea */
  nm_legal        VARCHAR(50),
  id_legal        VARCHAR(16),
  id_sector       VARCHAR(16), /* Remonte de Código sectorial*/
  id_status       VARCHAR(30), /* Remonte de Estado: activa, liquidada, ... */
  tp_company      VARCHAR(30), /* Remonte de Tipo de compañía: cliente, ... */
  de_company      VARCHAR(254),

  CONSTRAINT pk_companies PRIMARY KEY(gu_company)
)
```

Tabla Remonte

```
CREATE TABLE k_companies_lookup
(
  gu_owner      CHAR(32) NOT NULL, /* GUID de la workarea */
  id_section    VARCHAR(30) NOT NULL, /* Nombre del campo en la tabla base */
  pg_lookup     INTEGER NOT NULL, /* Progresivo del valor */
  vl_lookup     VARCHAR(255) NULL, /* Valor real del lookup */
  tr_es        VARCHAR(50) NULL, /* Valor visualizado en pantalla (esp) */
  tr_en        VARCHAR(50) NULL, /* Valor visualizado en pantalla (ing) */

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

Valores de ejemplo en la tabla de remonte para un área de trabajo

id_section	pg_lookup	vl_lookup	tr_es
id_sector	1	CNAE - 72	INFORMÁTICA
id_sector	2	CNAE - 53	VENTA AL POR MENOR
id_status	1	A	ACTIVA
id_status	2	Q	QUIEBRA
tp_company	1	CLIENTE	CLIENTE
tp_company	2	COMPETENCIA	COMPETENCIA
tp_company	3	PARTNER	PARTNER

☞ Si se quiere utilizar el mecanismo estándar de visualización y actualización de remotes que viene incluido en la librería de páginas JSP, hay que crear cualquier nueva tabla de remonte siguiendo el ejemplo anterior. Para ello las tablas de remonte deben cumplir las siguientes restricciones:

1. debe haber un campo llamado `gu_owner` que identifica el área de trabajo a la que pertenecen los datos del registro.
2. debe haber un campo llamado `id_section` que identifica el campo de la tabla base al cual se refiere el registro de remonte.
3. para cada valor debe un campo llamado `pg_lookup` que identifica un valor incremental progresivo por cada campo de la tabla base.
4. debe haber un campo llamado `vl_lookup` que contiene el valor del dato de remonte en un formato independiente del idioma.
5. debe haber un campos de etiqueta traducida llamados con el prefijo `tr_` y el código del idioma de la tabla `k_lu_languages` para cada idioma en el que se desee mostrar el valor traducido.
6. en la tabla base, cualquier valor susceptible de ser cargado mediante un remonte debe ser de tipo `VARCHAR(1...254)`.

Atributos definibles por el usuario

hipergate proporciona un mecanismo para la creación de atributos (campos) definidos por el usuario para cada tabla.

Los atributos definibles por el usuario sirven para enriquecer la información de una tabla ya existente sin necesidad de alterar físicamente el modelo de datos.

En la creación de nuevos atributos definibles por el usuario distinguiremos 2 etapas:

- 1ª) Definición de meta-datos. La definición misma de qué atributos se crearán y adscritos a qué tabla.
- 2ª) Grabación de datos. En los formularios de mantenimiento de la tabla base se añadirán dinámicamente los atributos definidos por el usuario.

k_lu_meta_attrs Esta tabla mantiene la información (meta-datos) de qué campos definibles por el usuario existen para cada tabla en cada área de trabajo. La tabla k_lu_meta_attrs no contiene los datos en si mismos sino sólo su definición.

Valores de ejemplo de la tabla k_lu_meta_attrs

gu_owner	nm_table	id_section	tp_attr	pg_attr	max_len	tr_es
[GUID]	k_companies_attrs	tx_propietarios	1	1	100	Propietarios
[GUID]	k_contacts_attrs	bo_navidad	1	1	1	Felicitar por Navidad

gu_owner	Identificador del Área de Trabajo a la que pertenece la definición del campo.
nm_table	Nombre de la tabla base.
id_section	Nombre de la sección a añadir a la tabla base (es cómo un nombre de campo virtual).
tp_attr	Tipo de atributo añadido. Actualmente sólo se admite el tipo 1, que se corresponde con texto plano introducido a través de un tag <INPUT TYPE=text> de HTML.
pg_attr	Valor ordinal progresivo del atributo añadido.
max_len	Longitud máxima del atributo en caracteres.
tr_es	Texto traducido para la etiqueta de sección en español.

Cómo añadir nuevas columnas al modelo de datos

Existen básicamente tres opciones para añadir columnas al modelo de datos:

1. Con atributos definibles por el usuario.
2. Añadendo las columnas directamente sobre las tablas del estándar con un comando SQL ALTER TABLE.
3. Creando subregistros de las tablas estándar en nuevas tablas.

Cada opción tiene sus propias ventajas y desventajas.

Lo más fácil es utilizar atributos definidos por el usuario, pero esta opción es también la más difícil de manejar a la hora de elaborar informes o forzar la consistencia de claves foráneas. Los atributos definidos por el usuario mezclan datos y metadatos y complican la tarea de escribir sentencias SQL sencillas para recuperar los valores almacenados.

Si se añade una nueva columna a una tabla del modelo de datos, la aplicación podrá seguir leyendo y escribiendo la tabla sin problemas. hipergate detecta automáticamente la estructura de las tablas en la bb.dd. y se ajusta a ellas. Tras añadir una nueva columna es preciso reiniciar el servidor web ya que la estructura de las tablas se mantiene cacheada en memoria y dicho cache sólo se actualiza una vez cuando arranca la aplicación.

Es necesario agregar la nueva columna en todos los formularios de edición y de grabación de datos. hipergate re-escribe todas las columnas de un registro cada vez que éste se actualiza. Si se añade una columna a una tabla pero no se agrega en los formularios de edición el valor de dicha columna se perderá cuando el usuario actualize el registro desde el interfaz web.

Internacionalización de datos

El modelo de datos utiliza diversas formas de mantener datos traducidos en varios idiomas.

En algunos casos, como las tablas de valores estáticos y las tablas de remonte por área de trabajo, existe un campo en el registro por cada idioma soportado. Ello implica que para añadir un nuevo idioma habrá físicamente que alterar el modelo de datos.

En otros casos, como las etiquetas de categorías, las traducciones están puestas en vertical, con un registro en una tabla especial por cada idioma. En este caso no será necesario ampliar el modelo para añadir un nuevo idioma sino que bastará con añadir un registro a la tabla de traducciones.

Uso del robot asistente de traducción

```
CREATE TABLE k_translations (
  tx_directory VARCHAR(128) NOT NULL,
  tx_filename   VARCHAR(128) NOT NULL,
  tx_tag        VARCHAR(255) NOT NULL,
  dt_created    TIMESTAMP     DEFAULT current_timestamp,
  dt_modified   TIMESTAMP,
  tr_en         VARCHAR(255),
  tr_es         VARCHAR(255),
  tr_de         VARCHAR(255),
  tr_it         VARCHAR(255),
  tr_fr         VARCHAR(255),
  tr_pt         VARCHAR(255),
  tr_ca         VARCHAR(255),
  tr_eu         VARCHAR(255),
  tr_ja         VARCHAR(255),
  tr_cn         VARCHAR(255),
  tr_ru         VARCHAR(255),
  tr_fi         VARCHAR(255),
  tr_pl         VARCHAR(255),
  tr_tw         VARCHAR(255),
  tr_nl         VARCHAR(255),
  tr_th         VARCHAR(255),
  tr_cs         VARCHAR(255),
  tr_uk         VARCHAR(255),
  tr_no         VARCHAR(255),
  tr_sk         VARCHAR(255),
  tr_ko         VARCHAR(255),
  tr_gl         VARCHAR(255)
)
```

Submodelo de Categorización

hipergate incorpora un árbol genérico de categorías que sirve para categorizar y jerarquizar todo tipo de objetos e información.

Este árbol de categorías está formado por una jerarquía de nodos (categorías) cada uno de los cuales puede contener un conjunto de objetos de diverso tipo.

El árbol admite que las etiquetas para los nodos se presenten al usuario final en distintos idiomas, permitiendo de esta forma crear un directorio multi-idioma sin necesidad de duplicar los nodos base.

Tablas y Vistas

Tablas	k_categories	Lista de categorías.
	gu_category	GUID de la Categoría.
	gu_owner	Usuario propietario de la Categoría.
	nm_category	Nombre de la Categoría. Este campo es una clave primaria alternativa para la tabla. Está pensado para poder localizar fácilmente una categoría mediante un nombre significativo a nivel humano. Los nombres de categoría se usan para generar rutas de directorio hasta los archivos asociados a dicha Categoría por lo que se recomienda emplear sólo caracteres ASCII-7 en los caracteres del nombre. La aplicación rellena automáticamente el campo nm_category mediante el método <code>com.knowgate.hipergate.Category.makeName()</code> , véase el API JavaDoc para más detalles.
	bo_active	Indica si la Categoría está activada. Este indicador está pensado para poder mostrar u ocultar Categorías cuando se está componiendo un directorio jerárquico para mostrar en web.
	dt_created	Fecha de creación del registro.

<code>dt_modified</code>	Fecha de modificación del registro.
<code>nm_icon</code>	Nombre del icono 1º para la Categoría. El primer icono se utiliza para representar nodos cerrados (no expandidos en el árbol de categorías). Por convenio sólo se almacena el nombre del archivo del icono y no su ruta completa. En la instalación estándar las imágenes para los iconos están en los directorios <code>/web/images/tree</code> y <code>/web/skins/skin/nav</code> dependiendo de si se usan iconos universales o iconos diferentes para cada skin.
<code>nm_icon2</code>	Nombre del icono 2º para la Categoría. El segundo icono se utiliza para representar nodos abiertos (expandidos en el árbol de categorías). No es obligatorio que el 2º icono y el 1º sean diferentes.
<code>id_doc_status</code>	Estado inicial de los productos/documentos que se asocian a esta categoría. Este campo está pensado para que se puedan moderar, aprobar, rechazar o caducar los contenidos que se incluyen en una Categoría. Los valores posibles se almacenan en la tabla <code>k_lu_status</code> . 1 Activo 0 Pendiente de Aprobación
<code>k_root_cats</code>	Categorías que son nodos raíz de primer nivel. Por defecto hipergate se suministra con una única categoría raíz padre de todas las otras. Pero se pueden crear categorías raíz adicionales.
<code>k_cat_labels</code>	Etiquetas traducidas de las categorías para cada idioma. Cada categoría tiene un conjunto de etiquetas de traducción que sirven para mostrar el nombre o título de la Categoría en diferentes idiomas.
<code>gu_category</code>	GUID de la Categoría.
<code>id_language</code>	Código del idioma para esta etiqueta. Según los valores de la tabla <code>k_lu_languages</code> .
<code>tr_category</code>	Etiqueta traducida de la Categoría.

<code>url_category</code>	URL asociada a la Categoría Traducida (opcional)
<code>k_cat_tree</code>	Árbol de categorías. Mantiene la relación padre-hijo entre las categorías. El modelo permite tener categorías hijas con varios padres, aunque, por simplicidad de manejo, se recomienda crear Categorías mono-parentales siempre que sea posible.
<code>k_cat_expand</code>	Lista pre-expandida de categorías hijas de una categoría dada. Esta tabla es un cache intermedio que la aplicación maneja automáticamente y que sirve para acelerar la expansión de ramas completas de categorías hijas, nietas, biznietas, etc.
<code>k_x_cat_objs</code>	Relación entre una categoría y los objetos que contiene. Basándose en el hecho de que todos los objetos de hipergate se identifican mediante un GUID , esta tabla usa un único campo llamado <code>gu_object</code> que mantiene una referencia a cualquier registro de otra tabla cuya clave primaria sea un GUID.
<code>gu_category</code>	GUID de la Categoría Contenedora.
<code>gu_object</code>	GUID del Objeto Contenido.
<code>id_class</code>	Identificador numérico de la clase a la que pertenece el objeto contenido. Debe ser un valor del campo <code>id_class</code> de la tabla <code>k_classes</code> . Se corresponde con el valor de la variable estática <code>IdClass</code> de cada clase Java estándar de hipergate.
<code>od_position</code>	Es posible establecer una posición relativa para cada objeto dentro de una categoría. Esto permite ordenar los objetos de forma arbitraria a la hora de presentarlos por pantalla o exportarlos a otro formato. Es responsabilidad del programa que mantiene la tabla <code>k_x_cat_objs</code> asignar las posiciones a los objetos. Varios objetos pueden compartir la misma posición relativa, en cuyo caso el orden de aparición no está definido.

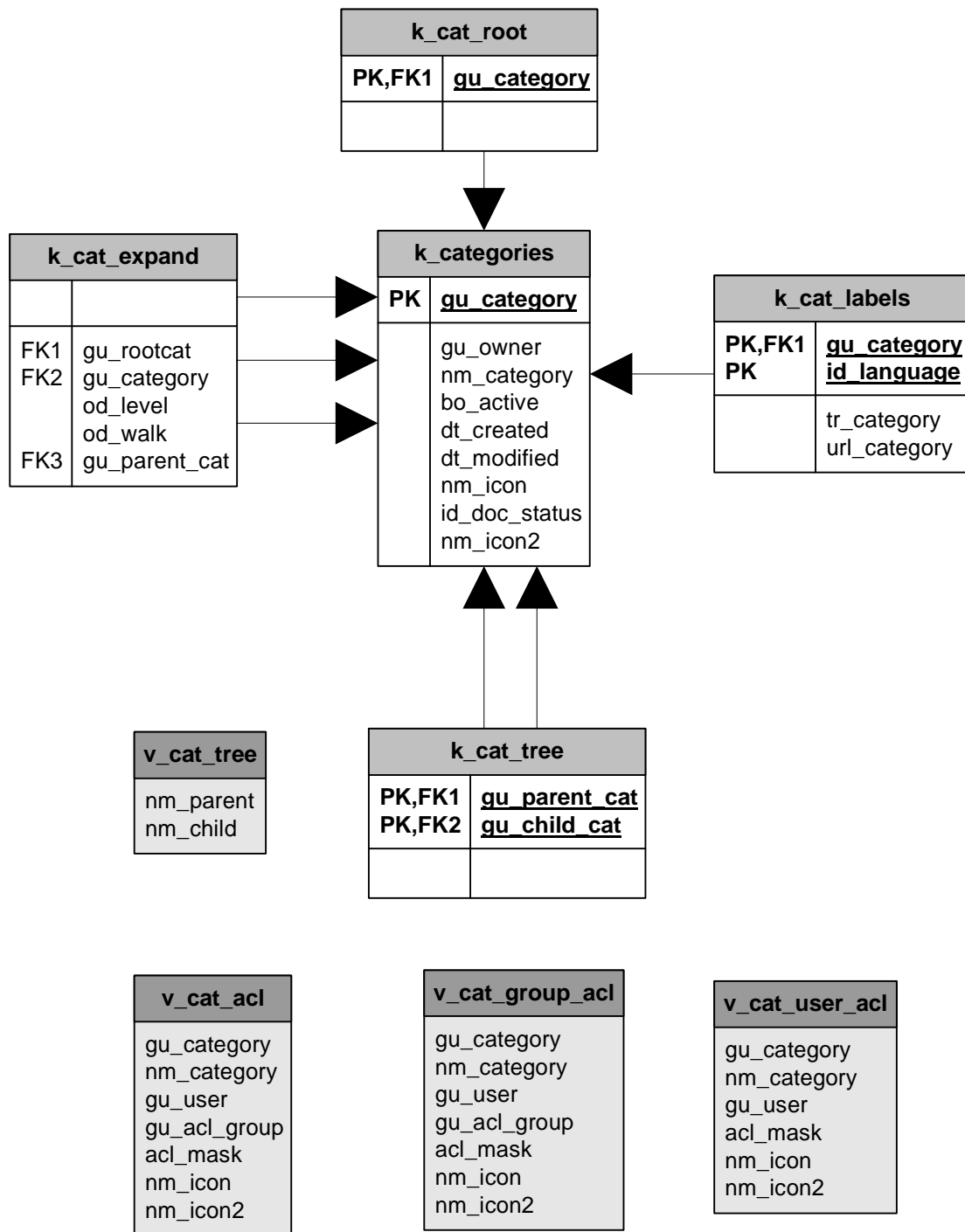
	<code>bi_attribs</code>	Máscara de bits que sirve para asociar indicares adicionales sobre la relación de pertenencia del objeto a la Categoría. La semántica del campo está definida por el programa que lo utilice.
Vistas	<code>v_cat_tree</code>	Muestra el árbol de categorías pero con nombres y no con GUIDs, para facilitar su lectura.
	<code>v_cat_tree_labels</code>	Vista del árbol de categorías cada una con sus etiquetas traducidas. En los sistemas donde están disponibles las vistas indexadas (SQL Server) o las vistas materializadas (Oracle 8i) esta vista se utilizada para acelerar la presentación de etiquetas de categorías descendientes de una determinada.
	<code>v_cat_group_acl</code>	Contiene la máscara de permisos de cada grupo en cada categoría. Para cada grupo hay un registro por usuario del mismo con su máscara de permisos resultante de su pertenencia al grupo.
	<code>v_cat_user_acl</code>	Contiene los permisos otorgados por categoría directamente a usuarios sin pertenencia a ningún grupo en particular.
	<code>v_cat_acl</code>	Es la unión de conjuntos de <code>v_cat_group_acl</code> y <code>v_cat_user_acl</code> . Uniendo los permisos garantizados a través de pertenecer a un grupo y aquellos otorgados directamente, se obtiene el conjunto total de permisos de un usuario sobre una categoría.

Cómo crear categorías directamente sobre el modelo

Para crear una categoría con sentencias SQL directamente sobre el modelo hay que seguir los pasos listados a continuación:

1. Generar un GUID para la Categoría. Esto puede hacerse con el método `generateUUID()` de `com.knowgate.misc.Gadgets`.
2. Insertar el registro principal de la Categoría en la tabla `k_categories`.
3. Si es una Categoría Raíz, insertarla en la tabla `k_cat_root`. Si no es raíz insertarla en `k_cat_tree` colgando del padre deseado.

4. Insertar las etiquetas de traducción en `k_cat_labels`.
5. Asignar los permisos por Grupo de Usuarios insertando registros en `k_x_cat_group_acl`.
6. Asignar los permisos por Usuario insertando registros en `k_x_cat_user_acl`.



Submodelo de Seguridad y Autenticación de Usuarios

hipergate proporciona un completo modelo nativo de seguridad basada en roles. Este modelo puede complementarse o sustituirse por otro, pero, básicamente, el modelo nativo está pensado para proveer todas las funcionalidades necesarias para autenticar usuarios y gestionar permisos sobre todos objetos del sistema.

El nivel de privilegios de seguridad se manifiesta al usuario de dos formas:

1ª) restringiendo las cosas que puede o no puede hacer en cada aplicación según el rol que tenga.

y 2ª) restringiendo el acceso a los objetos sobre los cuales no tenga permisos.

Dominios

La unidad básica de división administrativa en hipergate es el Dominio. Cada dominio contiene áreas de trabajo, grupos de usuarios, y usuarios individuales.

Tabla	k_domains	Lista de los dominios existentes en la base de datos.
	id_domain	Identificador del Dominio. Cada dominio se identifica unívocamente por un número. Los números de dominio del 0 al 2048 están reservados para uso interno del sistema y no deben utilizarse.
	dt_created	Fecha de creación.
	bo_active	Indica si el dominio está activado.
	nm_domain	Nombre del dominio. Este campo es una clave primaria alternativa para la tabla.
	gu_owner	GUID del Usuario Administrador del Dominio. Un Dominio puede tener varios administradores, bien declarados directamente bien a través del Grupo de

Administradores del Dominio; pero uno de los usuarios administradores debe estar directamente referenciado en la tabla de dominio, para impedir que se borren accidentalmente todos los administradores y el dominio resulte imposible de administrar.

<code>gu_admins</code>	GUID del Grupos Administradores del Dominio.
<code>dt_expire</code>	Fecha de Expiración del Dominio.

Área de Trabajo

El Área de Trabajo es una subdivisión de nivel más fino que el Dominio. Su propósito es separar datos de diferentes grupos de trabajo dentro de una misma organización para que cada grupo sólo vea los suyos propios.

Tabla	<code>k_workareas</code>	Lista de las áreas de trabajo. Cada área de trabajo pertenece a un Dominio. Un dominio puede contener un número ilimitado de áreas de trabajo.
--------------	---------------------------------	--

Roles

Las aplicaciones de la suite reconocen 4 roles predefinidos:

- Administrador
- Usuario Avanzado
- Usuario Estándar
- Invitado

Existe un quinto rol definible a medida por el programador, pero no se pueden crear más roles. Todas las aplicaciones de la suite reconocen y trabajan sólo con estos roles predefinidos.

Usuarios y Grupos

Puede existir un número ilimitado de grupos de permisos, aunque, por defecto se crean 4 para cada Área de Trabajo: Administradores, Usuarios Avanzados, Usuarios Estándar e Invitados; correspondientes a los roles del sistema.

Cada usuario puede pertenecer a un número arbitrario de grupos de permisos.

Tablas	k_acl_users	Usuarios por dominio.
	gu_user	GUID del usuario.
	id_domain	Identificador del Dominio al que pertenece el usuario.
	tx_nickname	Nombre abreviado. Se usa como identificador para conectarse a un Dominio y Área de Trabajo específica. El par [id_domain , tx_nickname] constituye una clave primaria alternativa para la tabla k_users.
	tx_pwd	Clave del acceso del usuario.
	bo_change_pwd	Indica si el usuario puede cambiar su clave.
	bo_active	Indica si el usuario está activado.
	len_quota	Cuota de disco actualmente utilizada por el usuario.
	max_quota	Máxima cuota de disco permitida para el usuario.
	tp_account	Tipo de Cuenta de Acceso Contratada.
	id_account	Identificador de la Cuenta Contratada.
	dt_last_update	Fecha actualización más reciente de los datos del usuario.
	dt_last_visit	Fecha del último login del usuario.
	dt_cancel	Fecha de cancelación de la cuenta de usuario.
	tx_main_email	E-mail principal del usuario. Este campo es una clave primaria alternativa para la tabla.
	tx_alt_email	E-mail alternativo del usuario.
	nm_user	Nombre de pila del usuario.

tx_surname1	Primer Apellido o Inicial intermedia.
tx_surname2	Segundo Apellido o Apellidos si tx_surname1 es la inicial intermedia.
tx_challenge	Pregunta para recuperar la clave de acceso olvidada.
tx_reply	Respuesta a la pregunta para recuperar la clave de acceso olvidada.
dt_pwd_expires	Fecha de expiración de la clave de acceso.
gu_category	GUID de la Categoría inicial (home) del Usuario.
gu_workarea	GUID del Área de Trabajo por defecto a la que pertenece el usuario. Un usuario puede pertenecer a varias áreas de trabajo. De ellas la referencia en la tabla k_users es la que se utiliza cuando el usuario se conecta el sistema usando sólo su e-mail y clave, sin especificar parámetros adicionales para el área de trabajo en la que quiere entrar.
nm_compay	Nombre de la compañía a la que pertenece el Usuario
de_title	Cargo del Usuario en su compañía.
id_gender	Sexo. Seguir el convenio M=Masculino, F=Femenino.
dt_birth	Fecha de Nacimiento.
ny_age	Edad.
marital_status	Estado Civil.
tx_education	Nivel de Estudios.
icq_id	Identificación de ICQ.
sn_passport	Número de documento de Identidad.
tp_passport	Tipo de documento de Identidad.

tx_comments	Comentarios.
k_acl_groups	Grupos de Usuarios por dominio.
k_x_group_user	Establece qué usuarios pertenecen a qué grupos.
k_x_cat_group_acl	Mantiene la asociación entre categorías y grupos de permisos.
k_x_cat_user_acl	Mantiene la asociación entre categorías y usuarios que tienen permisos sobre ellas.

Aplicaciones

La suite está dividida en Aplicaciones. Cada aplicación admite que se definan 4 roles por área de trabajo. La combinación de grupos, áreas de trabajo y aplicaciones es necesaria porque los usuarios de un área de trabajo pueden tener acceso a diferentes aplicaciones que los usuarios de otra área.

Tablas **k_apps** Aplicaciones Instaladas.

Valores precargados en la instalación estándar

Id. Bit de Aplicación	Descripción
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 Grupos adscritos a cada rol por Aplicación y Área de Trabajo.

id_app Identificador de la Aplicación.

gu_workarea	Identificador del Área de Trabajo.
gu_admins	Identificador del Grupo que ejerce el rol de Administrador en esta Aplicación y Área de Trabajo.
gu_powusers	Identificador del Grupo que ejerce el rol de Usuario Avanzado en esta Aplicación y Área de Trabajo.
gu_users	Identificador del Grupo que ejerce el rol de Usuario Estándar en esta Aplicación y Área de Trabajo.
gu_guest	Identificador del Grupo que ejerce el rol de Invitado en esta Aplicación y Área de Trabajo.
gu_other	Identificador del Grupo que ejerce el rol adicional definible por el programador.
path_logo	Ruta a ficheros físicos asociados a la Aplicación en esta Área de Trabajo.
len_quota	Cuota de disco actualmente utilizada por el Área de Trabajo.
max_quota	Máxima cuota de disco permitida para el Área de Trabajo.

Máscaras estándar de permisos

Cuando se establece el nivel de acceso de un usuario o grupo de usuarios sobre una categoría, se hace especificando una máscara de acceso para dicha categoría.


Los permisos predefinidos son:

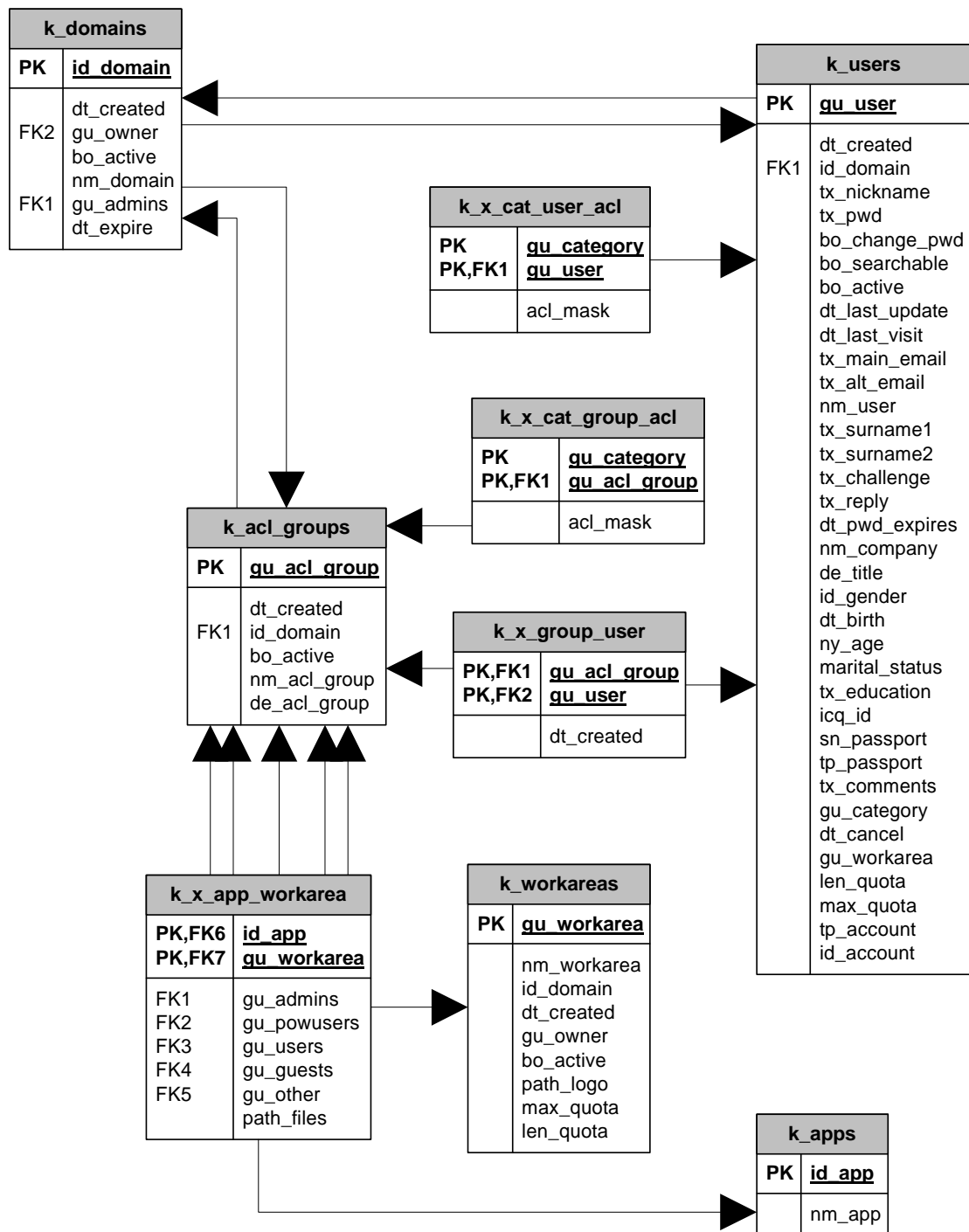
Tabla k_lu_permissions

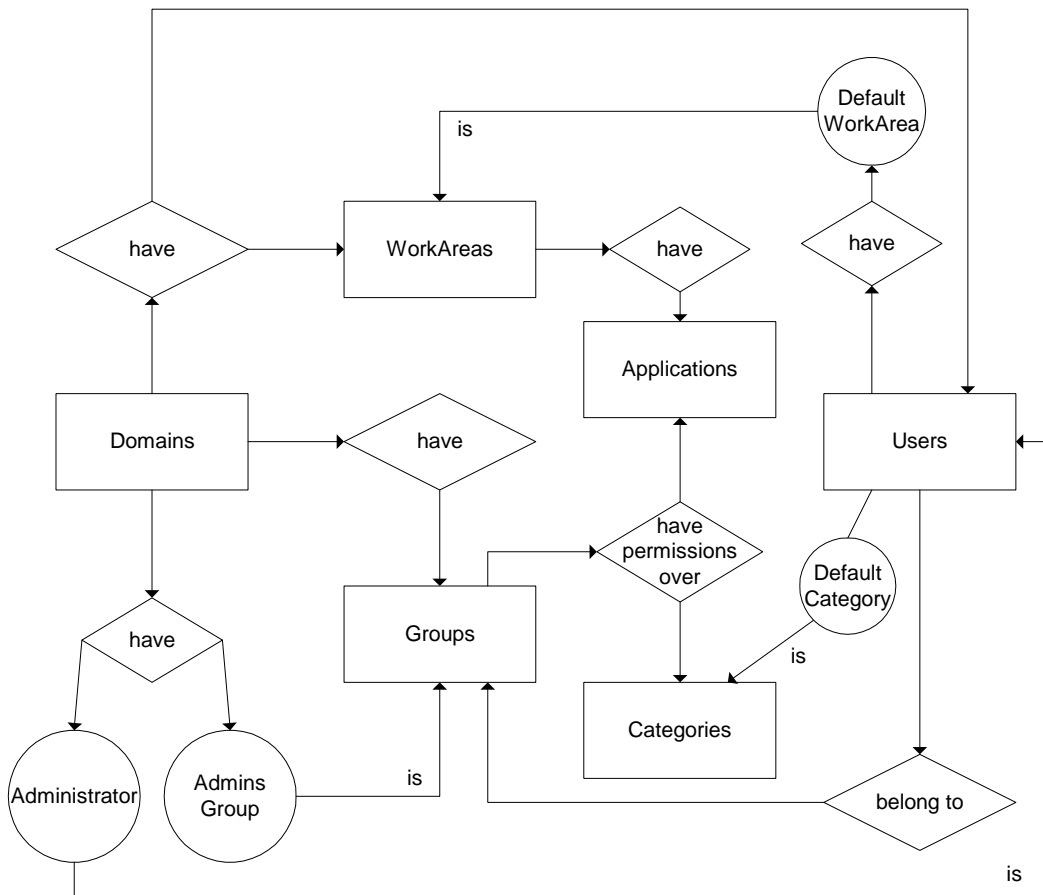
Listar	Leer	Agregar	Eliminar	Modificar	Moderar	Redactar	Gestionar Permisos	Control Total
1	2	4	8	16	32	64	128	2147483647

Carga Inicial de Datos para el Submodelo de Seguridad

El submodelo de seguridad viene cargado por defecto con 5 dominios en la tabla `k_domains`:

- **Dominio 1024 (SYSTEM):** Este es un dominio especial del sistema. Algunas aplicaciones lo reconocen y se comportan de forma especial respecto a él. Para el dominio SYSTEM se crea un único usuario administrador y un único grupo de usuarios administradores.
- **Dominio 1025 (MODEL):** Este dominio se utiliza como plantilla para crear otros dominios. Contiene 4 usuarios: Administrador, Usuario Avanzado, Usuario e Invitado; y 4 grupos de usuarios: Administradores, Usuarios Avanzados, Usuarios e Invitados.
 Se recomienda no escribir ni modificar nada sobre el Dominio MODEL. Este Dominio se utiliza como plantilla para generar nuevos Dominios mediante un proceso de clonación.
- **Dominio 1026 (TEST1):** Es un dominio precargado para poder hacer rápidamente pruebas de desarrollo sin necesidad de rellenar ningún dato.
- **Dominio 1027 (ACEP1):** Dominio precargado pruebas de Aceptación.
- **Dominio 1028 (PROD1):** Es un dominio precargado para un entorno de Producción.





Auditoría de logins

Desde la versión 2.2, los intentos de conexión con la aplicación se graban en la tabla **k_login_audit**.

bo_success '1' si la conexión tuvo éxito o '0' en caso contrario
nu_error 0 si la conexión tuvo éxito o el código del error si falló:

-1	Usuario no encontrado
-2	Clave inválida
-3	Cuenta de facturación desactivada
-4	Sesión expirada
-5	Dominio no encontrado
-6	Área de Trabajo no encontrada
-7	Área de Trabajo no establecida
-8	Cuenta de usuario cancelada

-9	Clave caducada
-10	El captcha no coincide
-11	El periodo de validez del captcha ha expirado
-255	Error interno del servidor

<code>dt_login</code>	Fecha y hora del login
<code>gu_user</code>	GUID del usuario que intentó la conexión
<code>tx_email</code>	e-mail con el que se intentó la conexión
<code>tx_pwd</code>	clave con la que se intentó la conexión
<code>gu_workarea</code>	Área de Trabajo a la que estaba adscrito el usuario
<code>ip_addr</code>	Dirección IP del origen

Submodelo de Portlets

Existe una única tabla que se encuentra en `security.ddl` necesaria para mantener la información relativa a los portlets.

<code>k_x_portlet_user</code>	Esta tabla contiene información de cómo presentar en pantalla los portlets para cada página, usuario y área de trabajo.
<code>id_domain</code>	Identificador del dominio al que pertenece el usuario.
<code>gu_user</code>	GUID del Usuario.
<code>gu_workarea</code>	Área de Trabajo para el Usuario. Puede ser la misma Área que indique el campo <code>gu_workarea</code> de <code>k_users</code> para el usuario <code>gu_user</code> o puede ser un Área de Trabajo diferente de la de por defecto.
<code>nm_portlet</code>	Nombre de la clase Java que implementa el interfaz <code>javax.portlet.GenericPortlet</code> y que generará el contenido del portlet. Por ejemplo <code>com.knowgate.http.portlets.CalendarTab</code> .
<code>nm_page</code>	Nombre de la página JSP que contiene el portlet.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

<code>nm_zone</code>	Nombre de la zona de la página JSP donde está ubicado el portlet. Los nombres de zona son arbitrarios y pueden cambiar de una página a otra, aunque comunmente son: <code>top</code> , <code>bottom</code> , <code>left</code> y <code>right</code> .
<code>op_position</code>	Posición del portlet dentro de la zona. Los portlets con posiciones menores se pintan en la parte superior por encima de aquellos con posiciones más altas.
<code>id_state</code>	Estado del portlet: <code>NORMAL</code> o <code>MINIMIZED</code> .
<code>dt_modified</code>	Fecha en la cual los datos utilizados por el portlet se modificaron por última vez. Esta fecha no se refiere a la información de <code>k_x_portlet_user</code> sino a los datos leídos por el portlet durante el proceso de composición. El dato se utiliza para servir el portlet desde un archivo cacheado o volverlo a componer. Si la fecha del archivo cacheado es posterior a la fecha de última modificación, entonces se sirve el archivo cacheado, en otro caso se descarta el archivo y se recompone todo el contenido del portlet.
<code>nm_template</code>	Nombre de la plantilla XSL que se utilizará para componer la salida del portlet como XHTML.

Submodelo de Tesoros, Direcciones e Imágenes

Tesoros

Un tesoro está compuesto básicamente de los siguientes elementos:

- Una lista de palabras o conceptos representados por combinaciones de palabras (términos).

- Un ámbito de conocimiento al cual dichos términos.
- Una lista de sinónimos para cada términos.
- Una relación jerárquica entre palabras que concreta o amplía el significado de un término.
- Opcionalmente, una descripción del término dentro del ámbito de conocimiento al que pertenece.

Para hacer simple la implementación, hipergate impone tres restricciones al modelo genérico de tesauros:

1ª) No todos los términos pueden tener sinónimos. En el modelo de hipergate hay términos primarios (que son los que representan el concepto) y términos sinónimos de un término primario. No hay sinónimos de sinónimos.

2ª) La relación jerárquica va del término más general al más específico. No hay relaciones laterales o de conceptos relacionados

3ª) Sólo se admiten hasta 10 niveles jerárquicos.

Ejemplo:

Término Primario:	Vehículo
Término Sinónimo:	Medio de Transporte
Término más restrictivo nivel 1:	Coche
Término más restrictivo nivel 2:	Coche de Alquiler
Término más restrictivo nivel 3:	Coche de Alquiler con Conductor
Término más restrictivo nivel 4:	Limusina de Alquiler con Conductor
Término más restrictivo nivel 1:	Moto
Término Sinónimo nivel 1:	Motocicleta
Término más restrictivo nivel 2:	Motocicleta con sidecar

Tablas

k_thesauri_root	Términos de nivel superior.
gu_rootterm	GUID del Término.
tx_term	Texto del Término
id_scope	Ámbito de Conocimiento.
id_domain	Dominio de seguridad de hipergate al que pertenece el Término.

<code>gu_workarea</code>	Área de Trabajo a la que pertenece el Término.
<code>k_thesauri</code>	Términos.
<code>gu_rootterm</code>	GUID del Término de nivel superior para este Término.
<code>gu_term</code>	GUID del Término.
<code>dt_created</code>	Fecha de Creación del Registro.
<code>id_language</code>	Idioma. Ver <code>k_lu_languages</code> .
<code>bo_mainterm</code>	1 si es Término Primario, 0 si es un sinónimo.
<code>tx_term</code>	Texto del Término. Por convenio todo mayúsculas y texto singular.
<code>tx_term2</code>	Texto en Plural para el Término.
<code>id_scope</code>	Ámbito de Conocimiento.
<code>id_domain</code>	Dominio de seguridad de hipergate.
<code>gu_synonym</code>	GUID del Término Primario para este Sinónimo.
<code>de_term</code>	Descripción del Término. Explica el significado del Término dentro de ámbito de conocimiento al que pertenece.
<code>id_term[0..9]</code>	Clave para todos los padres del término actual. Aparte de su GUID, a cada término se le asigna una clave primaria alternativa de tipo entero. En cada registro se almacena la lista completa de todos los padres del término. Los términos de nivel superior tienen su <code>id_term0</code> asignado a un número y el resto de los <code>id_term[1..9]</code> a NULL. Los términos del primer subnivel tienen su <code>id_term0</code> asignado al número de su padre, su <code>id_term1</code> con un número propio y el resto de los <code>id_term[2..9]</code> a NULL. Así sucesivamente para cada siguiente subnivel.

Direcciones Postales

Para todas las direcciones postales de todas las aplicaciones se usa una única tabla `k_addresses`. Cada submodelo engancha tablas de relaciones de claves foráneas estilo `k_x_addr_myclass` que relacionan las direcciones con otras entidades del modelo.

Tablas	<code>k_addresses</code>	Direcciones.
	<code>gu_address</code>	GUID de la Dirección.
	<code>ix_address</code>	Ordinal de la Dirección. Sirve para ordenar las direcciones arbitrariamente en aquellos casos en los que hay varias direcciones asociadas a la misma entidad externa.
	<code>gu_workarea</code>	Área de Trabajo a la que pertenece la Dirección.
	<code>dt_created</code>	Fecha de Creación del Registro.
	<code>bo_active</code>	1 si la dirección está activa, 0 en caso contrario.
	<code>dt_modified</code>	Fecha de Modificación del Registro.
	<code>gu_user</code>	Usuario al que está asociada la Dirección.
	<code>tp_location</code>	Tipo de Dirección: ej. "Oficina Central", "Sucursal", "Almacén", "Envío", "Facturación", "Personal", etc
	<code>nm_company</code>	Nombre de la Compañía.
	<code>tp_street</code>	Tipo de Vía.
	<code>nm_street</code>	Nombre de la Vía.
	<code>nu_street</code>	Número de la Vía.
	<code>tx_addr1</code>	Dirección Línea 1.
	<code>tx_addr2</code>	Dirección Línea 2.
	<code>id_country</code>	Código del País. Ver <code>k_lu_countries</code> .

nm_country	Nombre del País.
id_state	Código de la Provincia/Estado.
nm_state	Nombre de la Provincia/Estado.
mn_city	Nombre de la Ciudad.
zipcode	Código Postal.
work_phone	Teléfono del Trabajo (Centralita).
direct_phone	Teléfono Directo.
home_phone	Teléfono Personal.
mov_phone	Teléfono Móvil.
fax_phone	Fax.
other_phone	Teléfono de Contacto Adicional.
po_box	Apartado de Correos.
tx_email	E-mail.
url_addr	URL.
coord_x	Coordenada X en Plano.
coord_y	Coordenada Y en Plano.
contact_person	Persona de Contacto.
tx_salutation	Saludo.
id_ref	Referencia Externa para interfaz con otras aplicaciones.
tx_remarks	Comentarios.

k_addresses_lookup

gu_owner	GUID del Área de Trabajo.
id_section	Nombre del Campo en la tabla k_addresses.
pg_lookup	Progresivo del valor de remonte.
vl_lookup	Valor de remonte.
tr_es	Etiqueta de traducción al español.
tr_en	Etiqueta de traducción al inglés.

k_distances_cache Cache de distancias entre dos puntos.

lo_from	Localizador del punto de origen.
lo_to	Localizador del punto de destino.
nu_km	Distancia en kilómetros entre ambos puntos.
id_locale	Locale en el cual se expresan los puntos de origen y destino.
coord_x	Longitud.
coord_y	Latitud.

Imágenes

La tabla k_images guarda referencias en base de datos a las imágenes en disco. Estas referencias sirven para acelerar los procesos de listado de imágenes disponibles y para facilitar el cálculo de cuotas de disco consumidas por Usuario y por Área de Trabajo sin necesidad de recorrer los directorios del disco.

Para reducir el número de tablas de relación entre las imágenes y las entidades que las manejan, la tabla k_images tiene incluidos GUIDs para las dos entidades más típicas que contienen imágenes: PageSets y Productos.

Tablas	k_images	Imágenes.
	gu_image	GUID de la Imagen.
	path_image	Ruta absoluta hasta la Imagen.
	dt_created	Fecha de Creación del Registro.
	gu_writer	Usuario propietario de la Imagen.
	gu_workarea	Área de Trabajo.
	dt_modified	Fecha de Modificación del Registro.
	nm_image	Nombre del archivo de la Imagen.
	tl_image	Título de la Imagen (ALT de HTML).
	tp_image	Tipo de Imagen: "Thumbnail", "Detalle", etc.
	dm_width	Ancho en pixels.
	dm_height	Alto en pixels.
	id_img_type	Extensión de la Imagen.
	len_file	Longitud del archivo en bytes.
	gu_pageset	PageSet al que pertenece la Imagen.
	gu_block	Bloque del PageSet al que pertenece la Imagen.
	gu_product	Producto al que pertenece la Imagen.
	url_addr	Hiperenlace asociado a la Imagen.

Cuentas Bancarias

Tablas	k_bank_accounts	Cuentas Bancarias y Tarjetas de Crédito.
	nu_bank_acc	Número de cuenta bancaria sin guiones ni espacios.

gu_workarea	Área de Trabajo a la que pertenece la cuenta.
dt_created	Fecha de Creación del Registro.
bo_active	1 si la cuenta esta activa, 0 si está inactiva.
tp_account	Tipo de cuenta bancaria.
nm_bank	Nombre del Banco.
nm_cardholder	Nombre que aparece en la tarjeta.
nu_card	Número de la tarjeta de crédito sin guiones ni espacios.
tp_card	Tipo de Tarjeta de Crédito.
tx_expire	Fecha de expiración en formato texto MM/YYYY.
nu_pin	PIN de la tarjeta.
nu_cvv2	Código de verificación CVV2 de la tarjeta.
im_credit_limit	Límite de crédito de la cuenta o tarjeta.
de_bank_acc	Descripción de la cuenta bancaria.

Submodelo del Planificador de Tareas

El modelo de datos del planificador de tareas está pensado para dar soporte a una gama genérica de tareas de proceso por lotes.

Cada tarea (job, en inglés) está compuesta por una serie de unidades atómicas de proceso llamadas átomos.

Comandos

Cada tarea lleva asociado un comando que identifica en qué consiste dicha tarea. Los comandos determinan cómo actuará el ejecutor de tareas Java al encontrar una nueva tarea en la cola.

Tablas

k_lu_job_commands Comandos permitidos por tarea.

id_command Comando a ejecutar para la tarea.
[versión 1.0] MAIL → Enviar átomos por e-mail.
SAVE → Guardar ficheros de texto.
FTP → Enviar por FTP.
FAX → Enviar por FAX.

tx_command Texto descriptivo del comando.

nm_class Nombre de la subclase Java de com.knowgate.scheduler.Job que contiene el código ejecutor del comando.

k_lu_job_status Estados permitidos para las tareas.

k_jobs Lista de tareas, todas, pendientes y procesadas. Esta tabla se utiliza como una cola de procesamiento, aunque el orden de ejecución no está explícitamente forzado por el modelo sino que depende de la implementación del ejecutor de tareas que se utilice.

gu_job Identificador Único de la Tarea.

gu_workarea Identificador del [Área de Trabajo](#) a la que pertenece la Tarea.

gu_writer GUID del [Usuario](#) que creó la Tarea.

id_command Identificador del comando.

id_status Estado de la tarea.

dt_created Fecha de Creación.

dt_execution Fecha Programada de Ejecución.

dt_finished Fecha de Finalización.

dt_modified Fecha de Modificación del Registro.

tl_job Título descriptivo de la Tarea.

gu_job_group	Grupo de Tareas.
tx_parameters	Parámetros adicionales para la ejecución. Este campo contiene un número indefinido de parámetros. El formato de cada parámetro es "nombre : valor" separando el nombre del valor por dos puntos. Cada parámetro se separa del siguiente mediante una coma. Así un par de parámetros pueden tener la forma: "id_pageset:123456,id_list:26879877".
k_job_atoms	Átomos por tarea, para las tareas en curso. Los átomos llevan precargados todos los campos necesarios para su procesamiento ulterior. Esto es una táctica para reducir al mínimo la cantidad de consultas que se ejecutan contra la base de datos una vez que se arranca la ejecución de una tarea.
k_job_atoms_archived	Átomos por tarea, para las tareas finalizadas o canceladas. Cuando una tarea termina sus átomos pasan de la tabla k_job_atoms a k_job_atoms_archived.

Consultas por Formulario (QBF)

Aunque las Consultas por Formulario (*Query By Form*) están incluidas en el submodelo del planificador, no son una parte del mismo propiamente dicha.

Tablas	k_queries	Esta tabla almacena las consultas generadas mediante formularios. Ver Consultas por Formulario en el capítulo de clases Java para una explicación más detallada sobre los QBFs.
	gu_query	GUID de la Consulta.
	dt_created	Fecha de Creación.
	gu_workarea	GUID del Área de Trabajo a la que pertenece la Consulta.

tl_query	Nombre de la Consulta.
nm_queryspec	Nombre (sin extensión) del archivo XML que contiene la definición (metadatos) de la Consulta. Estos archivos se encuentran en el subdirectorío /storage/qbf.
dt_modified	Fecha de Modificación.
nm_field[1..3]	Nombre del Campo n-ésimo de la Consulta.
tx_value[1..3]	Etiqueta de remonte para el campo n-ésimo de la Consulta.
vl_code[1..3]	Valor real para el Campo n-ésimo de la Consulta.
nm_operator[1..3]	Operador de comparación del Campo con el Valor. Los operadores de comparación permitidos son:

=	Igual a
<>	Distinto de
>	Mayor que
<	Menor que
S	Empieza por (subcadenas)
C	Contiene (subcadenas)
N	Es Nulo (IS NULL)
M	Es No Nulo (IS NOT NULL)

☞ Cuando se definen las Consultas con valores nulos hay que tener presente que la condición “campo=NULL” es diferente de “campo IS NULL”. En general, la mayoría de los SGBDR interpretan que no hay dos valores nulos iguales.

tx_condition[1..2]	Fecha de Modificación. Condición lógica a aplicar. AND u OR.
tx_columns	Columnas a Recuperar.

Ejemplo:

Nombre de la Consulta : Tareas Pendientes de Alta Prioridad.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

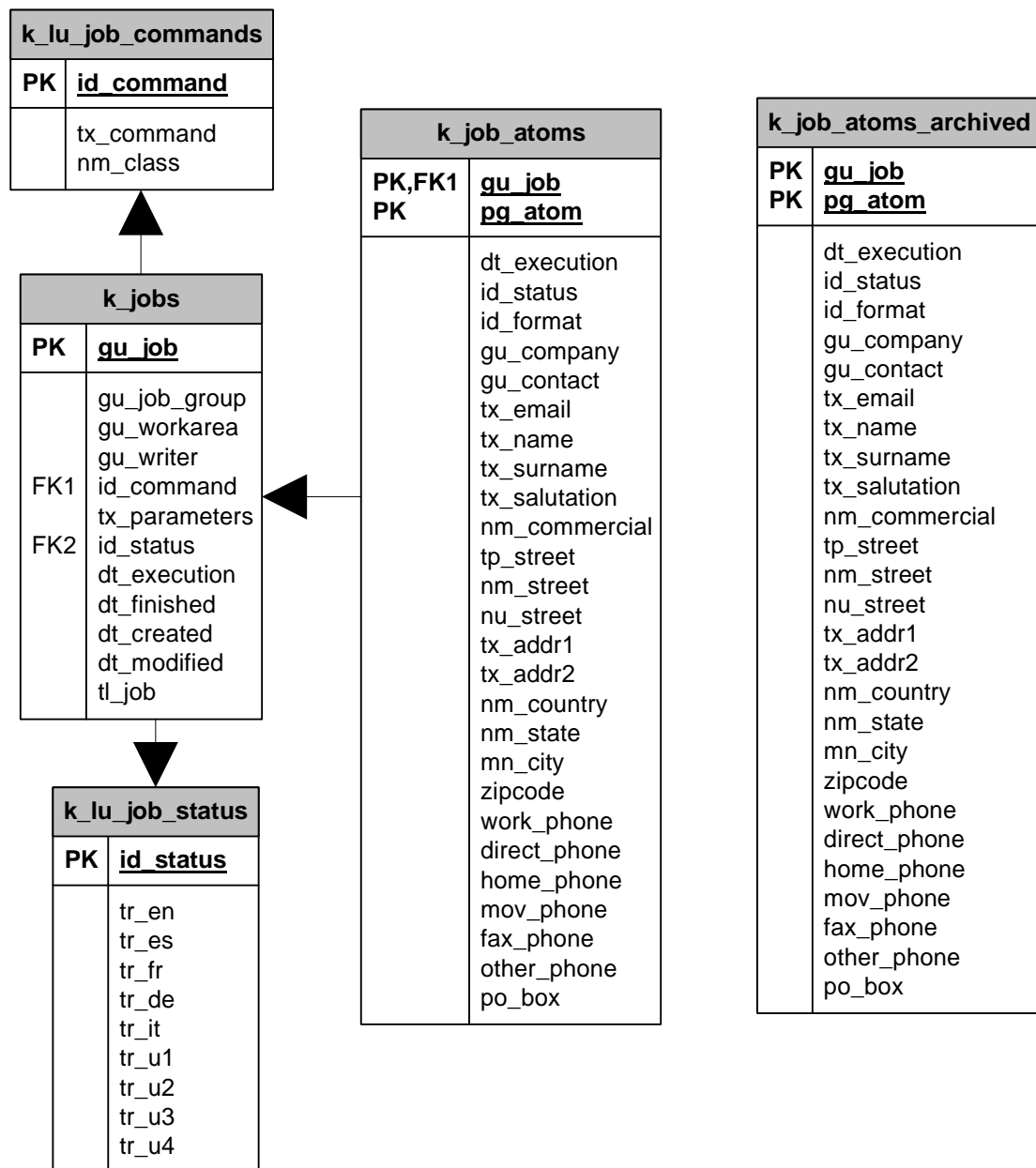
QBF : duties [.xml]

SQL : SELECT * FROM v_duty_resource WHERE od_priority=3 AND
tx_status='PENDIENTE'

Quedaría almacenada en la tabla como

tl_query	Tareas Pendientes de Alta Prioridad
nm_queryspec	duties
nm_field1	od_priority
nm_field2	tx_status
nm_field3	NULL
nm_operator1	=
nm_operator2	=
tx_value1	ALTA (campo k_duties_lookup.tr_es)
tx_value2	PENDIENTE
vl_code1	3 (campo k_duties_lookup.vl_lookup)
vl_code2	PENDIENTE
tx_columns	NULL

☞ El nombre de la vista y los filtros adicionales por área de trabajo no se almacenan en la tabla porque son comunes a todas las Consultas generadas y están guardados en los archivo XML de definición de los QBF.



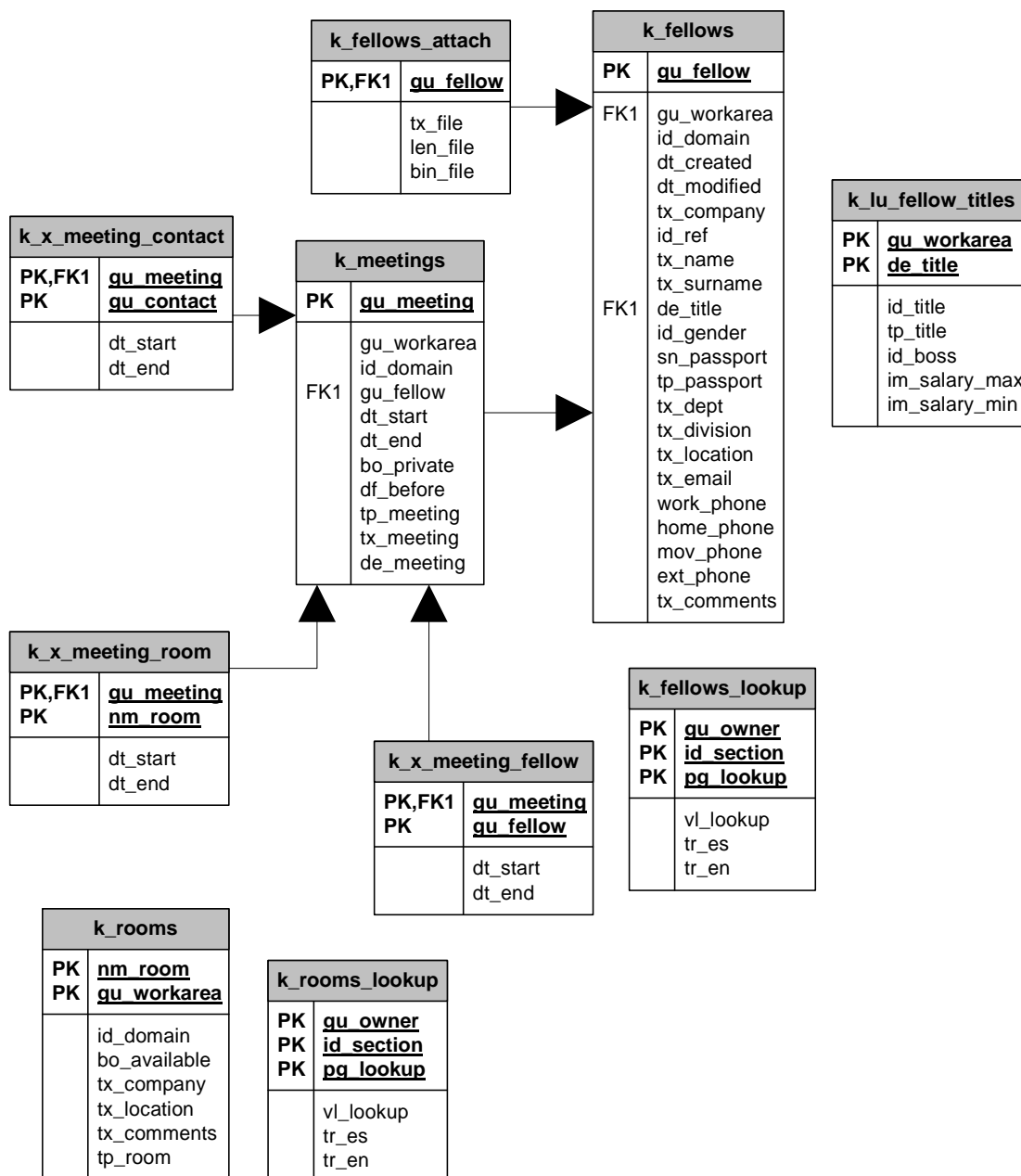
Submodelo de Trabajo en Grupo

Esta parte del modelo mantiene el directorio de empleados y la agenda de actividades programadas. La agenda es personal, existe una por cada usuario de la aplicación.

Los usuarios y los empleados son cosas independientes, pero, si un usuario entra en la parte de empleados la aplicación generará automáticamente un empleado clónico del usuario para que el usuario en cuestión pueda tener acceso a su agenda. Dado que las agendas están vinculadas a los empleados y no a los usuarios se necesita una copia del usuario como empleado para que pueda programar actividades.

Tablas	k_lu_fellow_titles	Contiene el árbol de empleos o cargos para un área de trabajo. Sólo es posible definir una estructura organizativa dentro de cada área de trabajo.
	gu_workarea	Identificador del Área de Trabajo a la que pertenece el Empleo/Cargo.
	de_title	Descripción del Empleo/Cargo.
	id_title	Código interno del Empleo/Cargo (opcional).
	tp_title	Grupo de Categoría Profesional (opcional).
	id_boss	Referencia de dependencia jerárquica de otro Empleo/Cargo superior.
	im_salary_max	Salario Máximo (HR).
	im_salary_min	Salario Mínimo (HR).
	k_lu_fellows	Listado de empleados.
	k_fellows_attach	Imágenes adjuntas a los empleados.
	k_fellows_lookup	Valores de remonte de empleados.
	k_rooms	Salas y Recursos Reservables.

k_rooms_lookup	Valores de remonte de Salas y Recursos.
k_meetings	Actividades y Reuniones Programadas.
gu_meeting	Identificador Único de la Actividad.
gu_workarea	Área de Trabajo a la que pertenece la Actividad.
id_domain	Dominio al que pertenece la Actividad.
gu_fellow	Identificador del Empleado propietario de la Actividad.
dt_start	Fecha y Hora de Inicio de la Actividad.
dt_end	Fecha y Hora de Fin de la Actividad.
bo_private	Indica si la actividad es privada. Las actividades privadas sólo son visibles para los empleados convocados a ella.
df_before	Avisar n-minutos antes de la actividad.
tp_meeting	Tipo de Actividad. Es un remonte con valores definibles por el administrador del área de trabajo.
tx_meeting	Texto que aparecerá como Asunto de la Actividad.
de_meeting	Descripción detallada de la Actividad.
k_x_meeting_room	Salas y Recursos por Actividad.
k_x_meeting_fellow	Empleados por Actividad.
k_x_meeting_contact	Contactos Externos por Actividad.



Calendarios laborables

A partir de la versión 4.0, hipergate incluye la capacidad para gestionar calendarios laborables. Cada calendario define un conjunto de días laborables o festivos entre dos fechas determinadas.

Los calendarios son jerárquicos y pueden combinarse de tal manera que para un determinado recurso se superpongan varios de ellos: el de su

país, región, y centro de trabajo, por ejemplo. De esta forma, es posible definir desde un calendario global aplicable a todo el mundo hasta un calendario individualizado para cada usuario.

Cada día en el calendario está marcado como laborable o festivo, para los laborables, además, es posible definir dos franjas horarias de horario de trabajo una para por la mañana y otra para por la tarde.

Los calendarios se almacenan en las tablas `k_working_calendar` y `k_working_time`.

Tablas	k_working_calendar	Maestro de calendarios
	gu_calendar	GUID del calendario.
	gu_workarea	GUID del Área de Trabajo a la que pertenece el calendario.
	id_domain	Identificador numérico del dominio al que pertenece el calendario.
	nm_calendar	Nombre descriptivo del calendario.
	dt_created	Fecha de creación.
	dt_modified	Fecha de última modificación.
	dt_from	Primer día definido en el calendario.
	dt_to	Último día definido en el calendario.
	gu_user	GUID del usuario al cual se aplica el calendario.
	gu_acl_group	GUID del grupo al cual se aplica el calendario.
	gu_geozone	GUID de la zona geográfica a la cual se aplica el calendario.
	id_country	Identificador del país al cual se aplica el calendario.
	id_state	Identificador del estado/provincia al cual se aplica el calendario.

k_working_time	Dias laborables y horario diario por calendario.
dt_day	Dia. Es un entero en formato yyyyymmdd.
gu_calendar	GUID del calendario.
bo_working_time	Entero corto. 1 si el día es laborable ó 0 si es festivo.
hh_start1	Entero corto. Hora de inicio del horario de trabajo de mañana [0..23] o -1 si no está definido el horario de trabajo de mañana.
mi_start1	Entero corto. Minuto de inicio del horario de trabajo de mañana [0..59] o -1 si no está definido el horario de trabajo de mañana.
hh_end1	Entero corto. Hora de fin del horario de trabajo de mañana [0..23] o -1 si no está definido el horario de trabajo de mañana.
mi_end1	Entero corto. Minuto de fin del horario de trabajo de mañana [0..59] o -1 si no está definido el horario de trabajo de mañana.
hh_start2	Entero corto. Hora de inicio del horario de trabajo de tarde [0..23] o -1 si no está definido el horario de trabajo de tarde.
mi_start2	Entero corto. Minuto de inicio del horario de trabajo de tarde [0..59] o -1 si no está definido el horario de trabajo de tarde.
hh_end2	Entero corto. Hora de fin del horario de trabajo de tarde [0..23] o -1 si no está definido el horario de trabajo de tarde.
mi_end2	Entero corto. Minuto de fin del horario de trabajo de tarde [0..59] o -1 si no está definido el horario de trabajo de tarde.

Submodelo de Productos y Documentos

La mayor parte de objetos susceptibles de ser alojados en Categorías (con la notable excepción de las Compañías y los Individuos) se manejan de forma unificada a través de la tabla `k_products`. Esta tabla contiene los productos de las tiendas virtuales, los documentos y los enlaces.

Definiciones y Conceptos

Producto	Es una entidad que representa de forma genérica objetos contenidos en Categorías. Los objetos pueden ser ítems físicos para el modelo de tienda virtual, o documentos simples, documentos compuestos o hiperenlaces.
Ubicación	Cada producto puede contener copias de si mismo en diferentes ubicaciones. Las ubicaciones pueden representar: <ul style="list-style-type: none">- almacenes de material (cada uno con su stock correspondiente).- diferentes versiones de un mismo documento.- sitios alternativos desde los que descargar versiones electrónicas del documento.- piezas para un documento compuesto por otros documentos.
Atributo Predefinido	Los Productos tienen una colección de atributos predefinidos en el modelo.
Atributo a Media	Para los valores asociados a un Producto que no encajen de forma natural en ninguno de los predefinidos, pueden definirse atributos adicionales.
Palabras Clave	Para cada producto es posible definir un conjunto limitado de palabras clave.

Elementos del modelo

Tablas

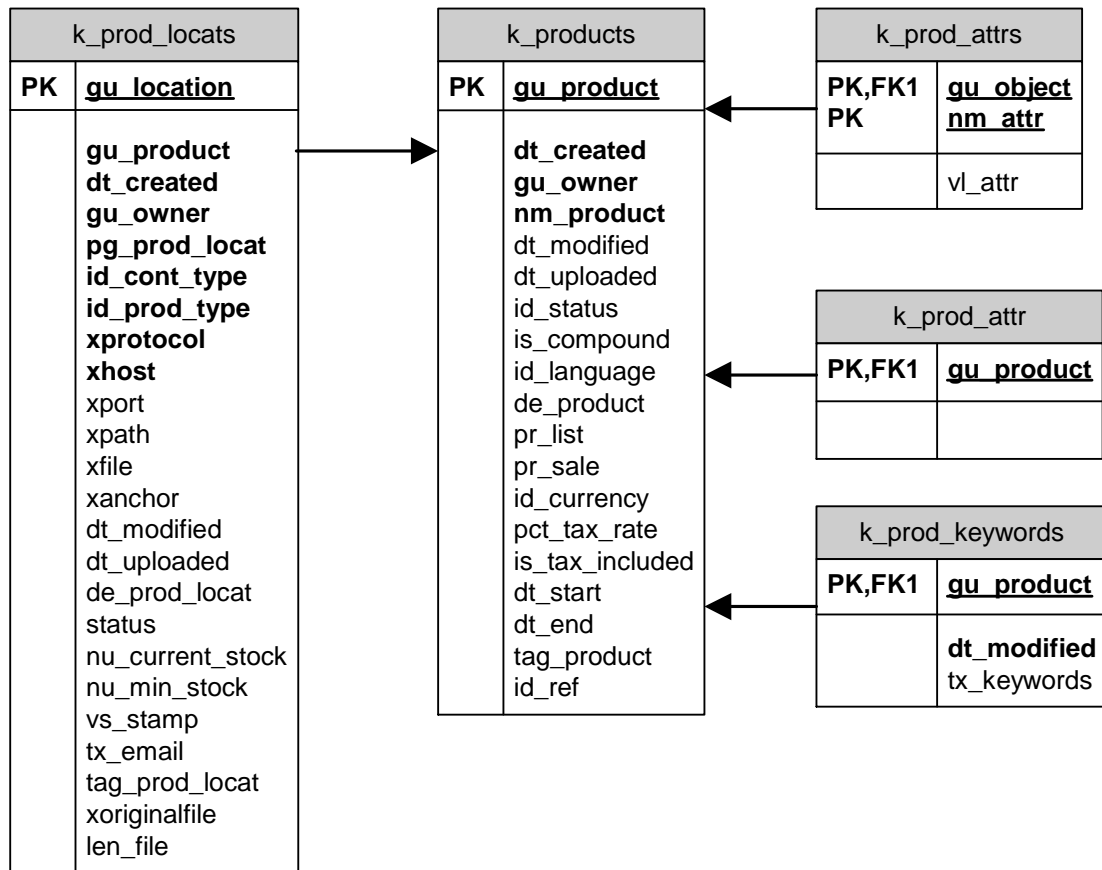
k_products

gu_product	GUID del Producto.
dt_created	Fecha de Creación del Registro.
gu_owner	GUID del Usuario propietario del Producto.
nm_product	Nombre del Producto.
id_status	Estado. Ver k_lu_status .
is_compound	0 si el producto es simple, ó 1 si es compuesto.
gu_blockedby	GUID del Usuario que actualmente tiene bloqueado el Producto.
dt_modified	Fecha de Modificación del Registro.
dt_uploaded	Fecha de la Última Carga.
id_language	Idioma.
de_product	Descripción del Producto.
pr_list	Precio de venta en lista.
pr_sale	Precio de venta en oferta.
id_currency	Código numérico para la moneda para el precio de venta. Ver k_lu_currencies .
pct_tax_rate	Porcentaje de impuestos aplicable. Esta cantidad debe estar expresada en porcentaje. Es decir, si los impuestos aplicables son del 15% entonces pct_tax_rate debe ser el valor entero 15, y no 0,15.

is_tax_included	1 si los impuestos están incluidos en el precio de lista, 0 en caso contrario.
dt_start	Fecha de Inicio de Oferta.
dt_end	Fecha de Fin de Oferta.
tag_product	Texto adicional para el producto.
id_ref	Referencia externa para interfaz con otros sistemas.
k_prod_locats	Cada Producto admite tener diferentes copias de si mismo en diferentes Ubicaciones.
gu_location	GUID de la Ubicación del Producto.
gu_product	GUID Producto ubicado.
pg_prod_locat	Progresivo de la Ubicación. El GUID del Producto junto con el Progresivo de la Ubicación constituyen una clave primaria alternativa para la tabla k_prod_locats. Es responsabilidad de la aplicación cliente asignar correctamente los progresivos de ubicación.
dt_created	Fecha de Creación del Registro.
gu_owner	GUID del Usuario propietario de la Ubicación.
id_cont_type	Tipo de Contenedor. El tipo de contenedor debe ser consistente con el protocolo de acceso (xprotocol). Ver k_lu_cont_types .
id_prod_type	Tipo de Producto. Ver k_lu_prod_types .
len_file	Longitud en bytes del archivo asociado. Para los hiperenlaces y los ítems físicos debe ser cero.
xprotocol	Protocolo de acceso. Los valores utilizados de forma estándar son: file:// Para rutas de archivos locales.

ftp://	Para protocolo FTP.
http://	Hiperenlaces HTML.
https://	Hiperenlaces HTML.
jdbc://	URLs de acceso a base de datos.
ware://	Items físicos ubicados en Almacenes.
xhost	Nombre del host. Por convenio, sin separadores al principio ni al final. Por ejemplo: "files.hipergate.org". Si el producto es un ítem físico el nombre del host es "warehouse"
xport	Puerto (opcional).
xpath	Ruta. Precedida de un separador de archivos y sin separador al final. Ejemplos: SI /opt/knowngate/knowngate NO opt/knowngate/knowngate NO /opt/knowngate/knowngate/
xfile	Nombre interno del archivo.
xoriginalfile	Nombre original del archivo (caso de que fuese renombrado internamente).
xanchor	Ancla en el documento (para URLs) sin almohadilla
dt_modified	Fecha de Modificación del Registro.
dt_uploaded	Fecha de Carga.
de_prod_locat	Descripción de la Ubicación.
status	Estado.
nu_current_stock	Stock actual.
nu_reserved_stock	Stock reservado.
nu_min_stock	Stock mínimo del Producto en esta Ubicación.
vs_stamp	Etiqueta de versión.

	<code>tx_email</code>	E-mail.
	<code>tag_prod_locat</code>	Texto adicional para la Ubicación.
	<code>k_prod_attr</code>	El subregistro de <code>k_products</code> en la tabla <code>k_prod_attr</code> contiene una colección opcional de campos fijos para cada producto. En <code>k_prod_attr</code> es donde se almacenan, por ejemplo, las propiedades de documentos OLE2.
	<code>k_prod_attrs</code>	Contiene un conjunto variable de pares atributo-valor adicionales por Producto.
	<code>gu_object</code>	GUID del Producto.
	<code>nm_attr</code>	Nombre del Atributo.
	<code>vl_attr</code>	Valor del Atributo.
	<code>k_prod_keywords</code>	Claves asociadas a un producto.
	<code>gu_product</code>	GUID del Producto.
	<code>dt_modified</code>	Fecha de Modificación del Registro.
	<code>tx_keywords</code>	Texto libre hasta 4000 caracteres. El contenido depende de la aplicación que interprete las claves.
Vistas	<code>v_prods_with_attrs</code>	Vista de Ubicaciones y Atributos de Producto por cada Ubicación.



Control de Versiones

El control de versiones es un caso particular del almacenamiento de múltiples copias del mismo documento en diferentes Ubicaciones.

hipergate proporciona un modelo de control de versiones que permite bloquear archivos que están siendo modificados.

El proceso es el siguiente:

1º) Creación de Versión Inicial.

- 1.1 Se crea el GUID para el Producto/Documento.
- 1.2 Se asignar manualmente un nombre desde la aplicación cliente.
- 1.3 Se asigna manualmente el campo `k_products.gu_owner` al GUID del Usuario a la que pertenezca el Producto.
- 1.4 Se pone automáticamente el Estado del Producto a 1 (Activo).
- 1.5 Se pone automáticamente el campo `is_compound` a 0 (los documentos compuestos no son versionables).

- 1.6 La fecha de última modificación se asigna a NULL (no modificado).
- 1.7 La fecha de creación se asigna automáticamente a la fecha del servidor de base de datos.
- 1.8 La fecha de carga se asigna automáticamente a la fecha del sistema de la aplicación cliente.
- 1.9 Se crea el GUID de la primera ubicación.
- 1.10 Se asigna el progresivo de la nueva Ubicación a 1.
- 1.11 Se asigna manualmente el campo `k_prod_locats.gu_owner` al GUID del Usuario que crea al Producto/Documento.
- 1.12 Se pone automáticamente el Estado de la Ubicación a 1 (Activa).
- 1.13 Se asigna la fecha de carga de la Ubicación exactamente al mismo valor que la fecha de carga del Producto.
- 1.14 Se Asigna una etiqueta de versión en el campo `k_prod_locats.vs_stamp` (opcional).

2º) Bloqueo y Toma de posesión de un Producto/Documento.

- 2.1 Se toma como última versión aquella con fecha de carga (`dt_uploaded`) más reciente. Para encontrarla se lee el campo `dt_uploaded` de la tabla `k_products` y a continuación se busca el registro con la misma fecha de carga en la tabla `k_prod_locats`.
- 2.2 Se cambia el estado del Producto y de todas sus Ubicaciones a 2 (Bloqueado).
- 2.3 Se asigna el campo `k_products.gu_blockedby` al GUID del Usuario que toma posesión del Producto/Documento.

3º) Desbloquear un Producto/Documento.

- 3.1 Se crea una nueva Ubicación.
- 3.2 Se actualiza el campo `k_products.dt_uploaded` con la fecha de carga de la última Ubicación.
- 3.3 Se actualizan los campos `dt_modified` de `k_products` y `k_prod_locats` con la fecha actual del sistema.
- 3.4 Se cambia el estado del Producto y de todas sus Ubicaciones a 1 (Activo).
- 3.5 Se asigna el campo `k_products.gu_blockedby` a NULL.

4º) Deshacer Bloqueo.

- 4.1 Se cambia el estado del Producto y de todas sus Ubicaciones a 1.
- 4.2 Se asigna el campo `k_products.gu_blockedby` a NULL.

6º) Eliminar definitivamente un Producto/Documento.

Consiste en el borrado físico de registros en la base de datos y sus archivos asociados. Sólo pueden hacerlo los usuarios con rol de Administrador.

Submodelo de Tienda Virtual

El submodelo de tienda virtual está compuesto por los siguientes elementos :

- Catálogos
- Categorías de productos
- Productos
- Pedidos
- Albaranes
- Facturas

Catálogos

Cada Área de Trabajo puede contener uno o más catálogos.

Los catálogos son útiles para separar conjuntos de productos que pertenecen a diferentes tiendas o, simplemente, para establecer una división arbitraria de productos por familias.

Existe un único juego de plantillas XSL para Pedidos, Albaranes y Facturas para cada Catálogo.

Los catálogos se almacenan en la tabla `k_shops` y se manipulan con la clase Java `com.knowgate.hipergate.Shop`.

Categorías

Cada Catálogo tiene una Categoría Raíz de la cual cuelgan el resto de las Categorías de Productos. Las categorías son jerárquicas pudiendo haber un número ilimitado de niveles.

En el modelo de datos un producto puede pertenecer a varias categorías simultáneamente, aunque el interfaz gráfico estándar sólo permite manejar una categoría por producto.

Los categorías se almacenan en la tabla `k_categories` shops y se manipulan con la clases Java


```
com.knowgate.hipergate.Categories,  
com.knowgate.hipergate.Category y  
com.knowgate.hipergate.CategoryLabel.
```

Productos

Cada producto está compuesto por los siguientes elementos:

- Información básica
- Imágenes
- Tarifas
- Ubicaciones del stock
- Atributos fijos
- Atributos definidos por el usuario
- Archivos adjuntos
- Palabras clave

La información básica de los productos se almacena en la tabla `k_products`. La clase base para manejar las instancias de producto es `com.knowgate.hipergate.Product`.

Cada producto puede tener una cantidad ilimitada de imágenes asociadas en la tabla `k_images`. Cada imagen está marcada con un tipo específico en el campo `tp_image` de la tabla `k_images`. Los tipos predefinidos son: `thumbview`, `normalview`, `frontview` y `rearview`. Se pueden utilizar otros tipos si se modifica el interfaz estándar para manejarlos.

Los archivos gráficos para las imágenes de los productos se almacenan bajo el directorio indicado en la propiedad `workareasput` del archivo `hipergate.cnf` usando el GUID del Área de Trabajo y el Nombre de la tienda de la siguiente forma:

```
workareasput/apps/Shop/k_shops.nm_shop/gu_product_tp_image
```

Puede haber opcionalmente varias tarifas asociadas a un producto. La información básica del producto ya contiene un precio de listado y un precio de oferta con fechas de inicio y fin para dicha oferta, de modo que sólo es necesario usar las tarifas cuando realmente se deseen vender el mismo producto a diferente precio según el segmento de cliente o la época del año.

Las tarifas se almacenan en la tabla `k_prod_fares`. Y se manejan con la clase Java `com.knowgate.hipergate.ProductFare`.

El stock de un producto puede estar ubicado en uno o varios almacenes.

A diferencia de las tarifas, que son opcionales, siempre hay al menos una

ubicación para cada producto en la tabla `k_prod_locats`. Cada ubicación mantiene su propio stock en unidades de producto. La clase Java para las ubicaciones de producto es `com.knowgate.hipergate.ProductLocation`.

Para cada producto existe un conjunto fijo de atributos y otro conjunto variable de atributos. Los atributos fijos se almacenan en la tabla `k_prod_attr` y los atributos variables en `k_prod_attr_s`.

Los archivos adjuntos a los productos se manejan como si fuesen ubicaciones del mismo producto.

Pedidos

Los pedidos se almacenan en la tabla `k_orders`. Cada pedido tiene un conjunto de líneas de pedido en la tabla `k_order_lines`.

Los pedidos tienen una numeración correlativa `pg_order` asignada mediante la secuencia `seq_k_orders`.

Albaranes

Los albaranes se almacenan en la tabla `k_despatch_advices`. Cada albarán tiene un conjunto de líneas en la tabla `k_despatch_lines`. La asociación entre los pedidos y los albaranes se mantiene en la tabla `k_x_orders_despatch`.

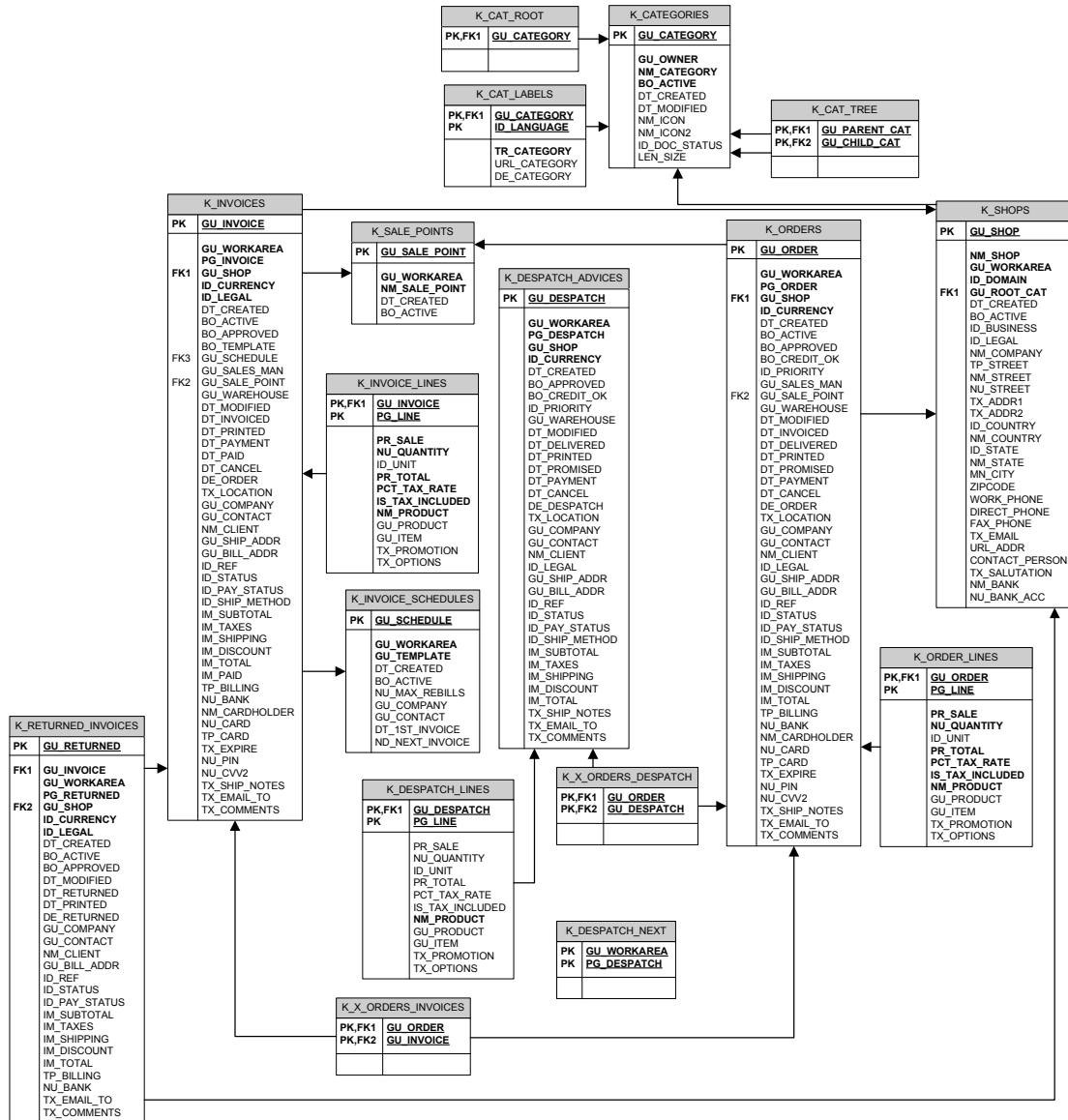
Los albaranes tienen una numeración correlativa por Área de Trabajo asignada en el campo `pg_despatch` mediante la tabla `k_despatch_next`.

Facturas

Las facturas se almacenan en la tabla `k_invoices`. Cada factura tiene un conjunto de líneas en la tabla `k_invoice_lines`. La asociación entre los pedidos y las facturas se mantiene en la tabla `k_x_orders_invoices`.

Las facturas tienen una numeración correlativa por Área de Trabajo asignada en el campo `pg_invoice` mediante la tabla `k_invoices_next`.

Las facturas devueltas se almacenan en la tabla `k_returned_invoices`.



Submodelo de Gestión de Ventas y Relaciones con Clientes

Definiciones y Conceptos

Vendedor

Un vendedor es un tipo particular de usuario del Dominio. Es especial en el sentido de que puede tener objetivos de venta asociados.

Compañía	Una compañía puede ser cliente, proveedor, partner, competencia u otro tipo de empresa u organización.
Contactos	Los contactos son individuos personales dentro de las Compañías. Los Contactos son individuos diferentes de los Usuarios del submodelo de Seguridad y de los Empleados del submodelo de Trabajo en Grupo.
Anotaciones	Texto fechados que pueden asociarse a un Contacto para mantener su historial de llamadas, visitas, envíos, etc.
Archivos Adjuntos	Archivos asociados a un Contacto. Pueden ser documentos, presentaciones, e-mails, etc.
Oportunidades	Representan oportunidades de venta abiertas o cerradas para una Compañía o Individuo.
Cuentas Bancarias	Las cuentas bancarias se definen en el submodelo de tesauros y se pueden vincular a Compañías o a Contactos.

Elementos del modelo

Tablas	k_sales_men	Vendedores.
	k_sales_objectives	Objetivos de Venta.
	k_companies	Compañías.
	gu_company	GUID de la Compañía.
	dt_created	Fecha de Creación del Registro.
	nm_legal	Razón Social.
	gu_workarea	GUID del Área de Trabajo a la que pertenece la Compañía.
	nm_commercial	Nombre Comercial.
	dt_modified	Fecha de Modificación del Registro.

dt_founded	Fecha de Constitución de la Compañía.
id_legal	Nº de identificación legal.
id_sector	Código del Sector.
id_status	Estado.
id_ref	Referencia para interfaz con sistemas externos.
tp_company	Tipo de Compañía.
gu_geozone	Zona geográfica a la que pertenece la Compañía.
nu_employees	Número de empleados.
im_revenue	Facturación Anual.
de_company	Descripción de la Compañía.
k_x_company_bank	Cuentas Bancarias por Compañía.
gu_company	GUID de la Compañía.
nu_bank_acc	Nº de cuenta bancaria.
k_x_company_addr	Direcciones por Compañía.
gu_company	GUID de la Compañía.
gu_address	GUID de la Dirección.
k_companies_lookup	Tabla de valores de remonte para Compañías.
k_companies_attrs	Atributos definidos por el Usuario para las Compañías.
k_contacts	Contactos.
gu_contact	GUID del Contacto.
gu_workarea	GUID del Área de Trabajo a la que pertenece el Contacto.

dt_created	Fecha de Creación del Registro.
bo_private	1 si el Contacto es privado del Usuario que lo creó, 0 si es un Contacto público dentro del Área de Trabajo a la que pertenece.
nu_notes	Cuenta de notas asociadas al Contacto.
nu_attachs	Cuenta de archivos asociados al Contacto.
dt_modified	Fecha de Modificación del Registro.
gu_writer	GUID del Usuario propietario del registro.
gu_company	GUID de la Compañía a la que pertenece el Contacto.
id_status	Estado.
id_ref	Referencia para interfaz con sistemas externos.
tx_name	Nombre de Pila.
tx_surname	Apellidos.
de_title	Empleo/Cargo.
id_gender	Sexo ('M'=Masculino, 'F'=Femenino)
dt_birth	Fecha de Nacimiento
ny_age	Edad.
sn_passport	Nº de pasaporte o documento de identidad.
tp_passport	Tipo de documento de identidad.
tx_dept	Departamento.
tx_division	División.
gu_geozone	Zona Geográfica.

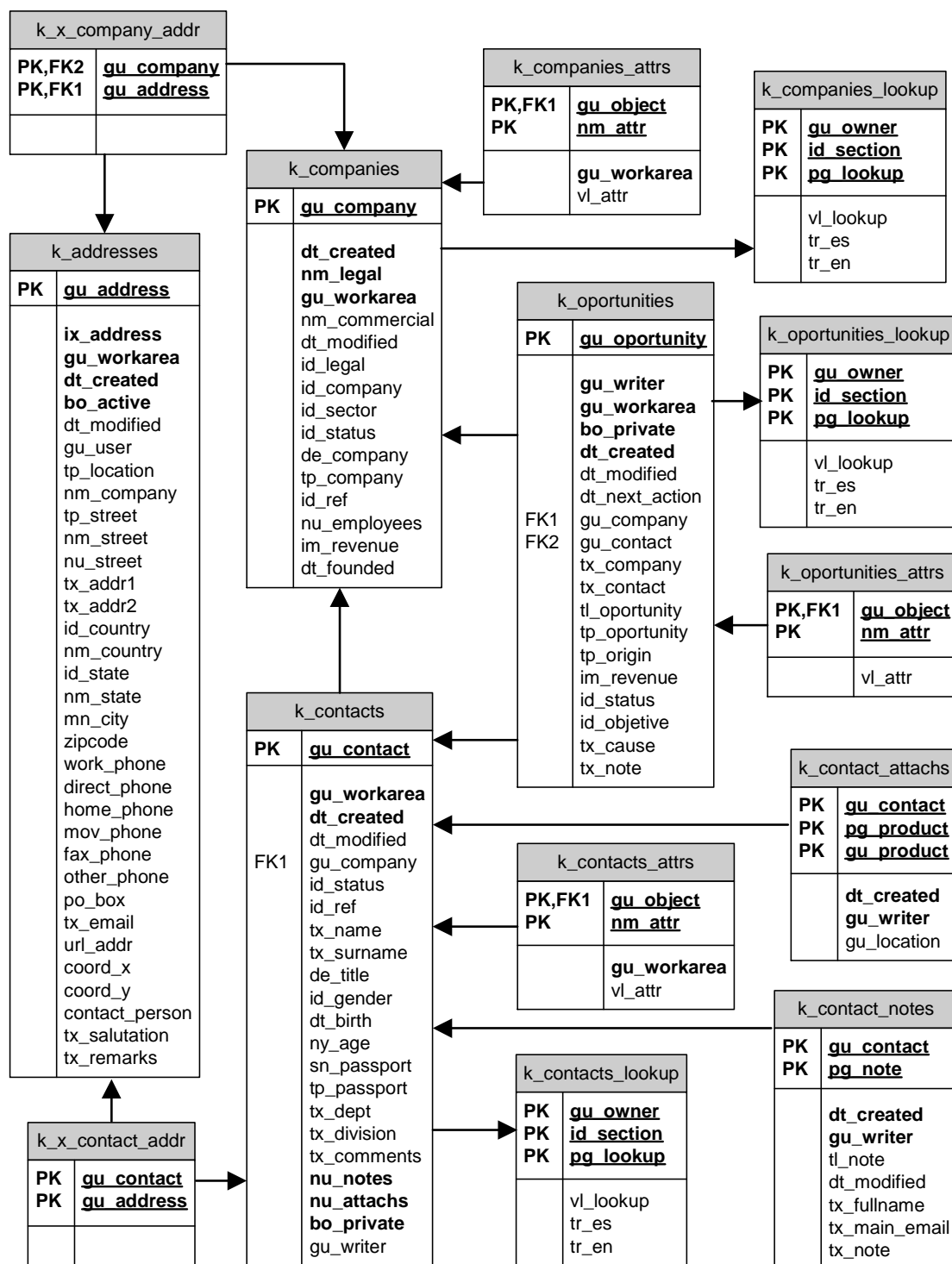
tx_comments	Comentarios.
k_contact_notes	Anotaciones para un Contacto.
gu_contact	GUID del Contacto.
pg_note	Progresivo de la Anotación.
dt_created	Fecha de Creación del Registro.
gu_writer	GUID del Usuario que escribió o modificó la Anotación.
tl_note	Título de la Anotación.
dt_modified	Fecha de Última Modificación de la Anotación.
tx_fullname	Nombre completo del Usuario que escribió o modificó la Anotación.
tx_main_email	E-mail del Usuario que escribió o modificó la Anotación.
tx_note	Texto de la Anotación (hasta 4000 caracteres).
k_contact_attachs	Archivos Adjuntos a un Contacto. Los Archivos Adjuntos se gestionan como archivos referenciados en la tabla k_products .
gu_contact	GUID del Contacto.
pg_product	Progresivo del Archivo Adjunto.
gu_product	GUID del Producto que mantiene la referencia al archivo en disco.
dt_created	Fecha de Creación del Registro.
gu_writer	GUID del Usuario que adjuntó el archivo.

k_x_contact_bank	Cuentas Bancarias por Contactos.
gu_contact	GUID del Contacto.
nu_bank_acc	Nº de cuenta bancaria.
k_x_contact_addr	Direcciones por Contacto.
k_x_contacts_lookup	Tabla de valores de remonte para Contactos.
k_x_contacts_attrs	Atributos definidos por el Usuario para los Contactos.
k_opportunities	Oportunidades.
gu_opportunity	GUID de la Oportunidad.
gu_writer	GUID del último Usuario que escribió la Oportunidad.
gu_workarea	GUID del Área de Trabajo a la que pertenece la Oportunidad.
bo_private	1 si la Oportunidad es privada del Usuario que la creó, 0 si es una Oportunidad pública dentro del Área de Trabajo a la que pertenece.
dt_created	Fecha de Creación del Registro.
dt_modified	Fecha de Modificación del Registro.
dt_next_action	Fecha para la Siguiente Acción.
gu_company	GUID de la Compañía de la Oportunidad.
gu_contact	GUID del Contacto de la Oportunidad.
tx_company	Nombre de la Compañía.
tx_contact	Nombre completo del Contacto.

tl_opportunity	Título de la Oportunidad.
tp_opportunity	Tipo de Oportunidad.
tp_origen	Origen de la Oportunidad.
im_revenue	Ingresos previstos o conseguidos.
id_status	Estado de la Oportunidad.
id_objective	Objetivo de la Oportunidad.
tx_cause	Causa de Cierre.
tx_notas	Comentarios.
k_opportunities_lookup	Valores de remonte para Oportunidades.
k_opportunities_attrs	Atributos definidos por el Usuario para las Oportunidades.
k_opportunities_changelog	Registro de cambios realizados en la tabla k_opportunities.
gu_opportunity	GUID de la Oportunidad.
nm_column	Nombre de la columna modificada en k_opportunities.
gu_writer	GUID del usuario que realizó la modificación.
dt_modified	Fecha de la modificación.
id_former_status	Estado de la oportunidad previo a la modificación.
id_new_status	Estado de la oportunidad posterior a la modificación.

tx_value	Si nm_column='id_status' entonces esta columna tiene el valor de tx_cause en k_opportunities posterior a la modificación.
----------	---

Vistas	v_company_address	Direcciones Activas por Compañía. (k_address.bo_active<>0).
	v_contact_titles	Valores de Empleo/Cargo para la tabla de remonte de Contactos (k_contacts_lookup.id_section='de_title').
	v_contact_company	Contactos por Compañía.
	v_active_contact_address	Direcciones Activas por Contacto.
	v_contact_address	Direcciones Activas por Contacto, incluyendo la información de la Compañía para cada Contacto.
	v_contact_list	Contactos con la información de la Compañía a la que pertenecen.



Restricciones de acceso para compañías e individuos

A partir de la versión 4.0, es posible establecer restricciones de acceso a compañías e individuos sólo para determinados grupos de usuarios.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

Submodelo de Producción de Contenidos

La mayor parte de la información necesaria para la producción de contenidos se almacena en archivos XML fuera de la base de datos.

No obstante, el SGBDR se utiliza para mantener índices de referencia que permitan listar, buscar y recuperar rápidamente los archivos XML de cada Área de Trabajo.

Tablas	k_microsites	Lista de Microsites disponibles por Área de Trabajo o Microsites comunes a todas las Áreas de Trabajo. Todas las encuestas comparten el mismo Microsite que viene cargado por defecto a partir de la versión 2.2 del producto con el GUID "SURVEYMICROSITEJIXBXMLDEFINITION".						
	gu_microsite	GUID del Microsite.						
	dt_created	Fecha de Creación del Registro.						
	tp_microsite	Tipo de Microsite : <table><tr><td>1</td><td>Microsites basados en plantillas XSL</td></tr><tr><td>2</td><td>Microsites basados en HTML libre</td></tr><tr><td>4</td><td>Encuestas</td></tr></table>	1	Microsites basados en plantillas XSL	2	Microsites basados en HTML libre	4	Encuestas
1	Microsites basados en plantillas XSL							
2	Microsites basados en HTML libre							
4	Encuestas							
	nm_microsite	Nombre del Microsite.						
	path_metadata	Ruta relativa al archivo XML de definición del Microsite desde el directorio /storage/xslt/templates. Para las encuestas este campo es siempre xslt/schemas/survey-def-jixb.xml						
	id_app	Identificador numérico de la Aplicación que maneja el Microsite : 13 Mailwire. Para las newsletters y otros documentos de 1 página remitibles por e-mail. 14 Web Builder. Para websites multipágina. 23 Surveys. Para encuestas.						

<code>gu_workarea</code>	GUID del Área de Trabajo propietaria del Microsite o NULL si el Microsite es visible desde todas las Áreas de Trabajo.
<code>k_pagesets</code>	Juegos de Páginas instanciados de Microsites.
<code>gu_pageset</code>	GUID de Juego de Páginas.
<code>dt_created</code>	Fecha de Creación del Registro.
<code>gu_microsite</code>	GUID del Microsite en que está basado el Juego de Páginas.
<code>gu_workarea</code>	GUID del Área de Trabajo a la que pertenece el Juego de Páginas.
<code>nm_pageset</code>	Nombre del Juego de Páginas.
<code>vs_stamp</code>	Etiqueta de versión del Juego de Páginas.
<code>id_language</code>	Idioma del Juego de Páginas. Ver k_lu_languages .
<code>path_data</code>	Ruta relativa al archivo XML con los datos de Juego de Páginas desde el directorio <code>/storage/domains</code> Para las encuestas este campo no apunta a un archivo sino a un directorio como <code>xslt/templates/Survey</code> . La ruta al archivo a cada página de definición de la encuesta para enlace JiXB se compone concatenando el campo <code>nm_pageset</code> con el progresivo de la página (<code>k_pageset_pages.pg_page</code>). Por consiguiente si <code>nm_pageset = 'Encuesta'</code> entonces el archivo de definición XML para la página dos de la encuesta estará en: <code>xslt/templates/Survey/Encuesta2.xml</code> Esto es debido a que las plantillas para Newsletters y WebSites contienen la composición de todas sus páginas en un único

archivo XML pero las encuestas están definidas en varios archivos, uno por cada página.

<code>id_status</code>	Estado del Juego de Páginas.
<code>dt_modified</code>	Fecha de Última Modificación del Registro.
<code>tx_comments</code>	Comentarios (hasta 255 caracteres).
k_pageset_pages	
<code>gu_page</code>	GUID de la Página en el Juego de Páginas.
<code>pg_page</code>	Progresivo de la Página.
<code>dt_created</code>	Fecha de Creación de la Página.
<code>dt_modified</code>	Fecha de Última Modificación de la Página.
<code>tl_page</code>	Título de la de Página.
<code>path_page</code>	Ruta a la Página.
k_pagesets_lookup	Valores de Remonte por Área de Trabajo para los Juegos de Páginas.
k_pageset_answers	
<code>gu_datasheet</code>	GUID del conjunto de respuestas.
<code>gu_page</code>	GUID de la Página en el Juego de Páginas.
<code>pg_answer</code>	Progresivo de la Respuesta [1..n].
<code>dt_created</code>	Fecha de Creación de la Respuesta.
<code>dt_modified</code>	Fecha de Última Modificación de la Respuesta.
<code>gu_pageset</code>	GUID de la encuesta para la cual son las respuestas.

<code>tp_answer</code>	Tipo de respuesta {TEXT, MEMO, CHOICE, MULTICHOICE, LICKERT, BOOLEAN, LIST, MATRIX}.
<code>nm_answer</code>	Nombre único para la respuesta dentro del juego de respuestas.
<code>gu_writer</code>	GUID del usuario que escribió la respuesta (de <code>k_users.gu_user</code>)
<code>tx_answer</code>	Contenido de la respuesta. Para respuestas con múltiples opciones simultáneas cada opción selecciona viene separada de las demás típicamente por punto y coma.

Submodelo de Foros

El modelo de Foros permite tener un número ilimitado de Grupos de Mensajes organizados de forma jerárquica.

El Grupo de Mensajes es un subregistro de la Categoría, de modo que, a la postre, los Mensajes se almacenan dentro de Categorías.

k_categories

Los grupos de mensajes son una subclase de las categorías. Para cada grupo existe una única categoría con el mismo GUID.

k_newsgroups Grupos de Mensajes.

`gu_newsgrp` GUID del Grupo de Mensajes. Debe corresponder con el GUID de una Categoría.

`id_domain` Dominio al que pertenece el Grupo de Mensajes.

`gu_workarea` Área de Trabajo a la que pertenece el Grupo de Mensajes.

`dt_created` Fecha de Creación del Grupo de Mensajes.

`bo_binaries` 1 si el Grupo de Mensajes admite archivos binarios adjuntos, 0 en caso contrario.

dt_expire	Tiempo de expiración por defecto de los mensajes en días a partir de su fecha de publicación o NULL si los mensajes no expiran nunca.
de_newsgroup	Descripción del Grupo de Mensajes.
k_newsmgs	Mensajes.
gu_msg	GUID del mensaje.
nm_author	Nombre completo del Remitente.
gu_writer	GUID del Usuario que remitió el mensaje.
dt_published	Fecha de Remisión para Publicación.
dt_start	Fecha de Inicio de Visibilidad.
id_language	Idioma del Mensaje.
id_status	Estado.
id_msg_type	Tipo de Mensaje. TXT Texto Plano HTML HTML
nu_thread_msgs	Número de mensajes en la Conversación.
gu_thread_msg	GUID del primer mensaje en la Conversación.
gu_parent_msg	GUID del Mensaje Anterior en la Conversación.
tx_email	Dirección de e-mail del Remitente.
tx_subject	Asunto.
dt_expire	Fecha de Expiración.
dt_validated	Fecha en la que el Mensaje fue validado.
gu_validator	GUID del Usuario que validó el mensaje.

`gu_product` GUID del Producto que mantiene los Archivos Adjuntos al mensaje.

`tx_msg` Texto del Mensaje (hasta 16Mb).

k_x_cat_objs

Los mensajes se asocian a grupos como objetos dentro de categorías.

`gu_category` GUID del grupo (`gu_newsgrp`).

`gu_object` GUID del mensaje (`gu_msg`).

`id_class` Identificador numérico de la clase `NewsMessage`. Este campo siempre tiene el valor 31 para los mensajes.

k_products

Si el mensaje contiene archivos adjuntos existe un único registro en `k_products` para dicho mensaje.

k_prod_locats

Existe un registro en `k_prod_locats` para cada archivo adjunto. Los archivos se almacenan fuera de la base de datos en `/web/workareas/gu_workarea/apps/Forums/nm_category`

☞ No se deben borrar mensajes con archivos adjuntos directamente de la base de datos. Hay que utilizar el API Java o de lo contrario los archivos se quedarán sin referencias perdidos indefinidamente en el disco duro.

Submodelo de Gestión de Proyectos e Incidencias

Este submodelo permite crear Proyectos o Contratos de Mantenimiento asociados a Compañías o Individuos.

Para cada Proyecto es posible definir Tareas Pendientes y reportar Incidencias.

Tablas **k_projects** Proyectos.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

<code>gu_project</code>	GUID del Proyecto.
<code>dt_created</code>	Fecha de Creación del Proyecto.
<code>nm_project</code>	Nombre del Proyecto.
<code>gu_owner</code>	Área de Trabajo a la que pertenece el Proyecto.
<code>id_parent</code>	Proyecto Padre.
<code>id_dept</code>	Departamento asignado al Proyecto.
<code>id_status</code>	Estado del Proyecto.
<code>dt_start</code>	Fecha de Inicio del Proyecto.
<code>dt_end</code>	Fecha de Finalización del Proyecto.
<code>pr_cost</code>	Coste.
<code>gu_company</code>	Compañía Cliente.
<code>gu_contact</code>	Individuo Cliente.
<code>id_ref</code>	Individuo Cliente. Referencia Externa (para interfaz con otras aplicaciones).
<code>de_project</code>	Descripción (hasta 1000 caracteres).
<code>k_projects_lookup</code>	Valores de Remonte por Proyecto.
<code>k_project_expand</code>	Guarda una lista pre-expandida de todos los hijos de cada proyecto a todos los niveles.
<code>gu_rootprj</code>	GUID del Proyecto Raiz.
<code>gu_project</code>	GUID del Proyecto hijo, nieto, bis nieto, etc. del Raiz.
<code>gu_parent</code>	Padre inmediato del Proyecto.
<code>nm_project</code>	Nombre del Proyecto hijo, nieto, bis nieto, etc.

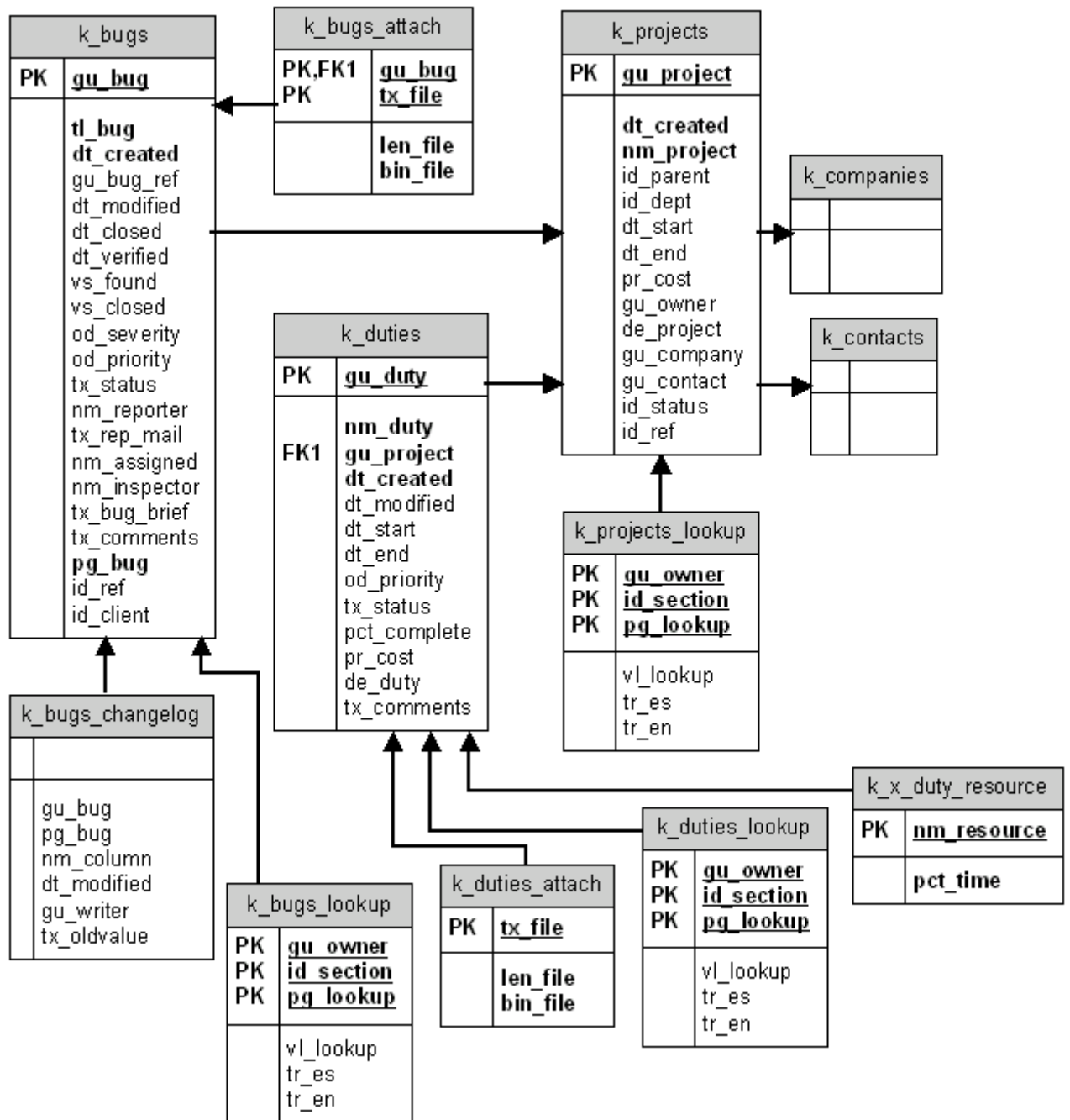
od_level	Nivel de profundidad en la jerarquía (1 = Proyecto Raiz).
od_walk	Orden absoluto de recorrido en profundidad.
k_project_costs	Costes asociados al Proyecto.
gu_project	GUID del Proyecto.
dt_created	Fecha de Creación del Coste.
dt_modified	Fecha de Última Modificación del Coste.
dt_cost	Fecha en la que deba aplicarse el Coste.
gu_user	Usuario a quien debe imputarse el Coste.
gu_writer	Usuario que creó o modificó el Coste por última vez.
tp_cost	Tipo de Coste.
pr_cost	Importe del Coste (moneda, horas, etc).
tl_cost	Título corto del Coste.
de_cost	Descripción detallada del Coste.
k_project_snapshots	Instantáneas de proyecto.
gu_snapshot	GUID de la Instantánea.
gu_project	GUID del Proyecto sobre el que se tomó la Instantánea.
gu_writer	GUID del Usuario que tomó la Instantánea.
dt_created	Fecha de Creación de la Instantánea.
tl_snapshot	Título de la Instantánea.
tx_snapshot	Contenido XML de la Instantánea.

k_duties	Tareas por Proyecto.
gu_duty	GUID de la Tarea.
nm_duty	Nombre de la tarea
gu_project	GUID del Proyecto al que pertenece la Tarea.
dt_created	Fecha de Creación de la Tarea.
dt_modified	Fecha de Última Modificación de la Tarea.
dt_start	Fecha de Inicio.
dt_end	Fecha de Finalización.
od_priority	Prioridad.
tx_status	Estado.
pct_complete	Porcentaje de la Tarea Completado.
pr_cost	Coste (moneda, horas, etc.).
de_duty	Descripción (hasta 1000 caracteres).
k_x_duty_resource	Recursos Asignados por Tarea.
gu_duty	GUID de la Tarea.
nm_resource	Nombre del Recurso.
pct_time	Porcentaje del tiempo del recurso asignado.
k_duties_lookup	Valores de Remonte por Tarea.
k_duties_attach	Documentos Adjuntos por Tarea.
gu_duty	GUID de la Tarea.

<code>tx_file</code>	Nombre del Documento.
<code>len_file</code>	Longitud del Documento en bytes.
<code>bin_file</code>	Contenido binario largo del Documento.
<code>k_duties_workreports</code>	Partes de Trabajo.
<code>gu_workreport</code>	GUID del Parte de Trabajo.
<code>gu_project</code>	GUID del Proyecto al que pertenece el Parte.
<code>gu_writer</code>	GUID del Usuario que realizó el Parte.
<code>dt_created</code>	Fecha de Creación del Parte.
<code>tl_workreport</code>	Título del parte.
<code>de_workreport</code>	Resumen y notas acerca del Parte.
<code>tx_workreport</code>	Contenido XML del Parte.
<code>k_bugs</code>	Incidencias por Proyecto.
<code>gu_bug</code>	GUID de la Incidencia.
<code>pg_bug</code>	Progresivo de la Incidencia.
<code>tl_bug</code>	Título de la Incidencia.
<code>gu_project</code>	Proyecto al que pertenece la Inidencia.
<code>dt_created</code>	Fecha de Creación de la Incidencia.
<code>gu_bug_ref</code>	Referencia a otra Incidencia igual que esta (para incidencias repetidas).
<code>dt_modified</code>	Fecha de Última Modificación de la Incidencia.
<code>dt_verified</code>	Fecha de Verificación de la Incidencia.
<code>dt_closed</code>	Fecha de Cierre de la Incidencia.

vs_found	Versión o N° de Serie del Producto en la que se encontró la Incidencia.
vs_closed	Versión del Producto en la que se cerró la Incidencia.
od_severity	Severidad.
od_priority	Prioridad.
tx_status	Estado.
nm_reporter	Nombre de la persona que reportó la Incidencia.
tx_rep_mail	E-Mail de la persona que reportó la Incidencia.
nm_assigned	Persona asignada al seguimiento.
nm_inspector	Persona que inspeccionó la Incidencia.
id_ref	Referencia para interfaz con otras aplicaciones.
id_client	Referencia del cliente.
gu_writer	GUID del último usuario que grabó la incidencia.
tx_bug_brief	Descripción de la Incidencia (hasta 2000 caracteres).
tx_comments	Comentarios. (hasta 1000 caracteres).
k_bugs_lookup	Valores de Remonte por Incidencia.
k_bugs_attach	Documentos Adjuntos por Incidencia.
gu_bug	GUID de la Incidencia.
tx_file	Nombre del Documento.
len_file	Longitud del Documento en bytes.
bin_file	Contenido binario largo del Documento.

	k_bugs_changelog	Registro de cambios en incidencias.
	gu_bug	GUID de la Incidencia.
	pg_bug	Progresivo de la Incidencia.
	dt_modified	Fecha de la modificación.
	gu_writer	GUID del último usuario que realizó la modificación o null si fue un cambio anónimo.
	nm_column	Nombre de la columna modificada en la tabla k_bugs.
	tx_oldvalue	Primeros 255 caracteres del valor previo en la tabla k_bugs.
Vistas	v_project_company	Proyectos por Compañía.
	v_duty_resource	Recursos por Tarea.
	v_duty_project	Tareas por Proyecto.
	v_duty_company	Tareas por Compañía.



Submodelo de Listas de Distribución

Definiciones y Conceptos

Las Listas de Distribución se utilizan para enviar un correo (genérico o personalizado) a un conjunto de direcciones de e-mail, en adelante llamaremos Miembros de la Lista a cada una de las direcciones de e-mail de envío.

Las direcciones de e-mail (Miembros) pueden extraerse y mantenerse de varias formas. Existen 3 clases de direcciones de Miembros:

- b. Los obtenidos del campo `tx_email` de la tabla `k_addresses` de Direcciones asociadas a Compañías de la tabla `k_companies`.
- c. Los obtenidos del campo `tx_email` de la tabla `k_addresses` de Direcciones asociadas a Contactos de la tabla `k_contacts`.
- d. Los obtenidos cargando directamente un archivo de texto delimitado conteniendo direcciones de e-mail o importando un archivo Windows Address Book de direcciones de Outlook Express.

Por otra parte las Listas de Distribución pueden ser de 4 tipos:

- a. **Listas Dinámicas.** Las Listas Dinámicas obtienen su conjunto de Miembros mediante una Consulta predefinida y almacenada en la tabla `k_queries`. Cada vez que se realiza un envío a los miembros de una Lista Dinámica, el conjunto de destinatarios se actualiza lanzando nuevamente la sentencia SQL asociada a la Consulta predefinida.
- b. **Listas Estáticas.** Al igual que las Listas Dinámicas, las Listas Estáticas obtienen su conjunto de miembros de una Consulta predefinida, pero, a diferencia de las anteriores, el conjunto de miembros se carga una vez en la tabla `k_x_list_members` y luego no se actualiza nunca.
- c. **Listas Directas.** Las Listas Directas están formadas por conjuntos de miembros cuyas direcciones de e-mail no están presentes en la tabla `k_companies` ni en la tabla `k_contacts`. Los miembros de las listas directas se cargan desde archivos de texto delimitado.

- d. **Listas Negras.** Las listas negras son un tipo especial de Listas Directas que se utiliza para bloquear el envío de e-mails a miembros de una determinada Lista. Una Lista Negra está siempre asociada a una lista Dinámica, Estática o Directa. La lista negra asociada es aquella cuyo campo `gu_query` es el GUID de la lista de distribución base. El tipo de las lista negras es 4. Por consiguiente, para hallar la lista negra asociada a una lista dada es posible emplear la sentencia SQL : **SELECT** `gu_list` **FROM** `k_lists` **WHERE** `gu_query='guid de la lista base'` **AND** `tp_list=4`

Tablas y Vistas

Tablas	k_lists	Listas de Distribución.
	gu_list	GUID de la Lista de Distribución.
	dt_created	Fecha de Creación de la Lista.
	gu_workarea	GUID del Área de Trabajo a la que pertenece la Lista.
	tp_list	Tipo de Lista de Distribución. 1 Estática 2 Dinámica 3 Directa 4 Negra
	gu_query	GUID de la consulta Asociada a la Lista (sólo en el caso de que sea una Lista Dinámica) o GUID de la Lista Base (en el caso de que ésta sea una Lista Negra).
	de_list	Descripción de la Lista.
	tx_sender	Nombre Completo del Remitente.
	tx_from	E-Mail del Remitente.
	tx_reply	Dirección de E-Mail de retorno.
	tx_subject	Asunto.
	k_x_list_members	Miembros de Listas de Distribución Estáticas, Directas o Negras.
	gu_list	GUID de la Lista de Distribución.
	tx_email	E-Mail del Miembro.
	tx_name	Nombre del Miembro.
	tx_surname	Apellidos del Miembro.

<code>tx_salutation</code>	Saludo aplicable al Miembro.
<code>bo_active</code>	1 si el miembro está activo, 0 en caso contrario.
<code>dt_created</code>	Fecha de Creación del Registro.
<code>dt_modified</code>	Fecha de Modificación del Registro.
<code>tp_member</code>	Tipo de Miembro. 90 Contacto tomado de la tabla <code>k_contacts</code> . 91 Compañía tomada de la tabla <code>k_companies</code> . 95 Miembro cargado directamente desde un archivo.
<code>gu_company</code>	GUID de la Compañía (si el miembro es de tipo 91).
<code>gu_contact</code>	GUID del Contacto (si el miembro es de tipo 90).
<code>id_format</code>	Formato preferido de recepción de mensajes. TXT Texto Simple. HTML HTML.

`k_member_address` El modelo de datos de hipergate permite tener múltiples contactos por compañía y múltiples direcciones por contacto.

La flexibilidad de tener un número arbitrario de contactos y direcciones acarrea la necesidad de usar al menos 5 tablas: `k_companies` (Compañías), `k_contacts` (Contactos), `k_addresses` (Direcciones), `k_x_company_addr` (Direcciones por Compañía) y `k_x_contact_addr` (Direcciones por Contacto).

En muchas ocasiones es conveniente acceder a todas las direcciones como si estuviesen en una única tabla. En las versiones 1.x esto se consigue utilizando la vista `v_member_address`. Se trata de una vista compleja y costosa de ejecutar para el gestor de base de datos.

Para mejorar el rendimiento, a partir de la versión 2.0 se incluyó la tabla

k_member_address. Esta es una tabla que se mantiene automáticamente con disparadores (triggers) en la base de datos. Cada vez que se inserta, actualiza, o borra una dirección, una compañía o un contacto, los disparadores modifican la tabla k_member_address para reflejar los cambios actualizados. Dado que todo el trabajo lo hacen los diparadores no es necesario (ni recomendable) escribir directamente en esta tabla.

Si la tabla se borra por error, es posible restaurarla simplemente ejecutando la sentencia

```
SQL:  INSERT INTO k_member_address
      SELECT * FROM k_member_address.
```

Vistas	v_member_address	Esta vista es la unión de 3 conjuntos : { Todas las direcciones activas de las Compañías ∪ Todas las direcciones activas de los Contactos asociados a alguna Compañía ∪ Todas las direcciones activas de los Contactos no asociados a ninguna Compañía }
---------------	-------------------------	--

Submodelo del Planificador de Tareas

Tablas y Vistas

Tablas	k_jobs	Tareas.
	gu_job	GUID de la Tarea.
	gu_workarea	GUID del Área de Trabajo de la Tarea.
	gu_writer	GUID del Usuario que creó la Tarea.
	id_command	Comando para ejecutar asociado a la Tarea (de la tabla k_lu_job_commands).

id_status	Estado de ejecución de la Tarea (de la tabla k_lu_job_status).
dt_created	Fecha de Creación de la Tarea.
tl_job	Título Descriptivo de la Tarea.
gu_job_group tx_parameters	GUID del lote de tareas (actualmente no se usa). Parámetros adicionales para el comando. Se trata de una lista de variables separadas por comas con sus valores correspondientes. Por ejemplo: "gu_pageset:P012345,gu_list:Lista1". El valor para cada variable se especifica tras los dos puntos. Es responsabilidad de las aplicaciones clientes parsear esta cadena de parámetros para extraer los valores de las variables.
dt_execution	Fecha programada de ejecución o NULL si la ejecución debe empezar lo antes posible.
dt_finished	Fecha de finalización o NULL si la ejecución aún no ha finalizado.
dt_modified	Fecha de última modificación del registro.
k_job_atoms	Átomos por tarea pendientes de ejecutar. Cada Átomo representa típicamente una dirección de correo a la cual enviar un e-mail, o un número para mandar un fax, o una ubicación para subir un fichero por FTP. Por comodidad, y para reducir el número de accesos a la base de datos durante la ejecución de tareas planificadas; los campos de uso típicos se insertan dentro del registro del Átomo.
gu_job	GUID de la Tarea.
pg_atom	Progresivo del Átomo.
dt_execution	Fecha programada de ejecución.

id_status	Estado de proceso del Átomo.
id_format	Formato de envío de datos (nomalmente "TXT" o "HTML").
gu_company	GUID de la Compañía para envíos postales, de fax y correo electrónico.
nm_commercial	Nombre Comercial de la Compañía.
gu_contact	GUID del Contacto Individual.
tx_email	Dirección de e-mail del destinatario.
tx_name	Nombre de pila del destinatario.
tx_surname	Apellidos del destinatario.
tx_salutation	Saludo del destinatario.
tp_street	Tipo de Vía.
nu_street	Número de Vía.
tx_addr1	Línea de Dirección 1.
tx_addr2	Línea de Dirección 2.
nm_country	Nombre del Pais.
nm_state	Nombre de la Provincia/Estado.
mn_city	Nombre de la Ciudad.
zipcode	Código Postal.
work_phone	Teléfono del Trabajo (Centralita).
direct_phone	Teléfono Directo.
home_phone	Teléfono Personal.

mov_phone	Teléfono Móvil.
fax_phone	Fax.
other_phone	Teléfono Adicional.
po_box	Apartado de Correos.
k_job_archived	Átomos por tarea ya ejecutados.
k_lu_job_status	Remonte de estados de tareas.

Código de Estado	Descripción
-1	Cancelada
0	Pendiente
1	Terminada
2	Suspendida
3	En Ejecución

k_lu_job_commands Comandos permitidos para una Tarea.

id_command Código del Comando.

Código Comando	Descripción
VOID	No hacer nada
MAIL	Enviar newsletter por mail
SEND	Enviar un mail ordinario
NTFY	Notificar evento por mail
FAX	Enviar un fax
SAVE	Guardar en disco local
FTP	Guardar por FTP

tx_command Descripción del Comando.

nm_class Nombre completo cualificado de la subclase Java que implementa la ejecución del Comando.

k_job_atoms_tracking	Tracking de átomos. Esta tabla se usa para recibir notificaciones de lectura procedentes de los web beacons ocultos en e-mails enviados.
gu_job	GUID de la Tarea.
pg_atom	Progresivo del Átomo.
gu_company	GUID de la Compañía destinataria.
gu_contact	GUID del Contacto destinatario.
ip_addr	Dirección IP del la máquina cliente.
tx_email	Dirección de correo del destinatario.

Almacenamiento de correo

hipergate 2.1 incluye un sistema de almacenamiento local compatible con JavaMail.

hipergate emplea un método híbrido de almacenamiento utilizando archivos planos y una base de datos relacional. Los mensajes se almacenan en archivos planos en format MBOX. Los archivos MBOX son sencillamente una concatenación de mensajes en formato RFC 822. MBOX no proporciona ningún mecanismo de indexación de mensajes, de modo que las búsquedas y recuperación deben implementarse separadamente.

hipergate usa una base de datos relacional y Jakarta Lucene para indexar los mensajes.

hipergate puede almacenar los mensajes en archivos MBOX o en campos LONGVARBINARY de la base de datos. Por defecto, se utiliza MBOX. No se recomienda almacenar los mensajes completos dentro de la base de datos debido a que los buzones de correo pueden crecer mucho y afectar al rendimiento de la base de datos y al tiempo necesario para hacer los backups.

Incluso si se utiliza MBOX para almacenar los mensajes, la base de datos sigue siendo necesaria para la indexación.

Existe un archivo MBOX para cada carpeta de mensajes. Las carpetas son una subclase de las categorías de hipergate. De hecho no existe ninguna entrada explícita para las carpetas en el modelo de datos sino que se reutiliza el modelo de categorías.

Los archivos MBOX se ubican en el directorio de su categoría correspondiente bajo el subdirectorio /storage. Por ejemplo para el usuario con alias *ad6148* del dominio TEST la ruta a su bandeja de entrada de mensajes tendría el aspecto:

```
.../storage/domains/2049/workareas/c0a801bffe3a42d52100000e12e2153/  
ROOT/DOMAINS/TEST/TEST_USERS/TEST_ad6148/TEST_ad6148_email/TEST_ad  
6148_inbox/ TEST_ad6148_inbox.mbox
```

2049 es el identificador numérico del dominio y c0a801... es el GUID del Área de Trabajo.

☞ Cuando se cambia el Área de Trabajo por defecto para un usuario, sus archivos MBOX no cambian de ubicación. El hecho de que el identificador del Área de Trabajo se quede cableado en la ruta al directorio del archivo MBOX debe tenerse en cuenta si se escriben rutinas que hagan uso de ello.

Enlace entre el correo y el gestor de contactos de hipergate

Cada vez que se envía o se recibe un mensaje, las direcciones de correo se chequean contra las tablas `k_users` y `k_member_address`. Si existe un Usuario, Contacto o Compañía con el e-mail dado en la misma Área de Trabajo que el mensaje entonces se establece una referencia en la tabla `k_inet_addrs` que permite trazar los mensajes enviados y recibidos para cada contacto de hipergate. Para que el seguimiento funcione es preciso enviar los mensajes desde el interfaz de hipermail. Los mensajes se escanéan cuando son abiertos por primera vez, para que un mensaje entrante quede asociado a un contacto es preciso leerlo al menos en una ocasión.


Cache de mensajes

Los mensajes no se almacenan localmente en hipergate hasta que son abiertos por primera vez. Una vez que el mensaje ha sido abierto, se descarga completamente y se almacena en la carpeta inbox.

Tablas y Vistas

<code>k_categories</code>	Carpetas de mensajes.
<code>k_mime_msgs</code>	Índice de mensajes.
<code>gu_mimemsg</code>	GUID del mensaje. Este es un identificador interno de hipergate de 32 caracteres, no el Id. de mensaje RFC 822.
<code>gu_workarea</code>	GUID del Área de Trabajo a la cual pertenece el mensaje.

<code>pg_message</code>	Posición ordinal del mensaje dentro de su carpeta.
<code>gu_category</code>	GUID de la carpeta que contiene el mensaje. Las carpetas son subclases de las categorías de la tabla <code>k_categories</code> . No hay registros especiales para las carpetas sino que se utiliza directamente la tabla de categorías. Para cada usuario se crean por defecto 6 carpetas bajo su categoría de inicio: <i>inbox</i> , <i>drafts</i> , <i>sent</i> , <i>received</i> , <i>delete</i> y <i>spam</i> .
<code>gu_parent_msg</code>	GUID del mensaje padre. Esta columna sólo se utiliza en mensajes adjuntos dentro de otro mensaje de nivel superior.
<code>nu_position</code>	Posición (en bytes) del mensaje dentro del archivo MBOX.
<code>id_message</code>	Id. del mensaje generado por el proveedor de correo.
<code>len_mimemsg</code>	Longitud del mensaje en bytes.
...	
<code>by_content</code>	Esta columna contiene el texto principal del mensaje, bien texto plano o HTML. El mismo texto se almacena en el archivo MBOX o en la parte del mensaje correspondiente en la tabla <code>k_mime_parts</code> pero está también presente en esta columna para acelerar el acceso al contenido básico del mensaje.
<code>k_mime_parts</code>	Partes del mensaje.
<code>gu_mimemsg</code>	GUID del mensaje.
<code>id_message</code>	Id. del mensaje generado por el proveedor de correo.
<code>pg_message</code>	Posición del mensaje en la carpeta.

<code>nu_offset</code>	Desplazamiento en bytes de esta parte desde el inicio del mensaje.
<code>id_disposition</code>	Puede ser inline, attachment, reference o pointer. inline y attachment se utilizan para mensajes terminados mientras que reference y pointer se utilizan mientras se está componiendo los borradores de los mensajes. reference significa que la parte no está contenida en el archivo MBOX ni en la columna <code>by_content</code> sino que es un archivo externo. Para las referencias la columna <code>file_name</code> contiene la ruta completa al archivo. pointer significa que la parte se encuentra dentro de un archivo MBOX.
<code>by_content</code>	Fuente de la parte. Si se utiliza el modo MBOX esta parte es siempre NULL.  Cuando se utiliza el modo BLOB el código fuente del mensaje no se almacena completo en ningún lugar sino fraccionado por partes en la table <code>k_mime_parts</code> . Esto permite ver o descargar cada archivo adjunto individualmente sin necesidad de procesar todo el mensaje.
<code>k_inet_addrs</code>	Destinatarios de correo.
<code>gu_mimemsg</code>	GUID del mensaje.
<code>id_message</code>	Id del mensaje generado por el proveedor de correo.
<code>pg_message</code>	Posición ordinal del mensaje en su carpeta.
<code>tx_email</code>	Dirección del destinatario.
<code>tp_recipient</code>	Tipo de destinatario { <code>from</code> <code>to</code> <code>cc</code> <code>bcc</code> }.
<code>tx_personal</code>	Nombre completo del destinatario.
<code>gu_user</code>	Usuario cuyo <code>tx_main_email</code> en la tabla <code>k_users</code> es el mismo que <code>tx_email</code> .

<code>gu_contact</code>	Contacto cuya dirección de correo en la tabla <code>k_member_address</code> es la misma que <code>tx_email</code> .
<code>gu_company</code>	Compañía cuya dirección de correo en la tabla <code>k_member_address</code> es la misma que <code>tx_email</code> .

Procedimientos Almacenados

4

Los procedimientos almacenados PL/SQL (Oracle), Transact-SQL (MS SQL Server) o PL/pgSQL (PostgreSQL) se utilizan con dos propósitos: 1º) para obtener el mejor rendimiento en algunas operaciones contra la base de datos y 2º) para externalizar el manejo de dependencias entre objetos fuera de las clases Java compiladas.

Dónde encontrar los fuentes de los procedimientos almacenados

En la carpeta `com/knowgate/hipergate/datamodel/procedures` del archivo `hipergate.jar`. Están divididos según los submodelos de datos sobre los que actúan.

Porqué procedimientos almacenados en el gestor

En muchas ocasiones, cuando se inicia el desarrollo de una aplicación multi-plataforma lo primero que se hace es eliminar el uso de características propietarias de cada gestor de base de datos. Esta decisión se toma para reducir el esfuerzo de transportar el código y para hacerlo más compatible con nuevos gestores de bases de datos.

Con la eliminación de los procedimientos almacenados se pierde el uso de uno de los mecanismos más potentes y rápidos de acceso a los gestores de bases de datos.

En hipergate se ha seguido la pauta de utilizar procedimientos almacenados allí donde pueden mejorar la eficiencia del programa. Se ha realizado un esfuerzo adicional de re-escribir los procedimientos almacenados para cada gestor. Ello contribuye a mejorar la calidad del producto final por los siguientes motivos:

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

- 1º) Los procedimientos almacenados son el método más rápido para ejecutar lotes de sentencias SQL precompiladas en el SGBDR.
- 2º) Los procedimientos almacenados reducen el tráfico de red entre el servidor web y la base de datos.
- 3º) Para funciones que buscan algo en la base de datos y devuelven un único valor, es más eficiente usar un parámetro de salida en un procedimiento almacenado que crear un ResultSet en cliente.
- 4º) El código de los procedimientos puede modificarse más fácilmente que las clases Java compiladas. Escribiendo algunas subrutinas como procedimientos se consigue que sean más fáciles de cambiar si se produce algún cambio o ampliación en el modelo subyacente sin necesidad de recompilar todo el proyecto Java.

Cómo llamar a los procedimientos desde código Java

Normalmente, no hay necesidad de llamar directamente a los procedimientos almacenados en el gestor de base de datos desde código Java, ya que la mayoría de los procedimientos están escritos específicamente para acelerar o externalizar algún método concreto de un objeto estándar que ya llama al procedimiento. Por consiguiente, lo habitual es llamar al método del objeto Java que lleva dentro la llamada al procedimiento almacenado y olvidarse de los detalles de la llamada.

No obstante, en este apartado veremos un ejemplo de cómo se llama al procedimiento `k_sp_authenticate` desde el método `authenticate()` de la clase `com.knowgate.ACL`.

Este ejemplo es interesante porque ilustra las diferencias que existen entre los gestores :

```

public static short authenticate
(JDBCConnection oConn, String sUserId, String sAuthStr, int iFlags)
throws SQLException, UnsupportedOperationException {

    /* Busca el usuario en la tabla k_users y verifica si la clave
       pasada como parámetro coincide con la almacenada en la tabla y
       si, en caso de coincidir el usuario está activado y la clave no
       ha expirado.
    */

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

    switch (oConn.getDataBaseProduct()) {

        case JDBCConnection.DBMS_ORACLE:

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

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

            oCall.execute();

            /* En Oracle no hay diferencia entre los tipos SMALLINT y los
               tipos NUMBER, de modo que aquí el resultado se recoge como
               un número decimal que luego se parsea y se convierte a un
               entero corto de 16 bits. Normalmente el driver JDBC puede
               llevar a cabo esta conversión de decimal a entero corto
               transparentemente si se solicita un valor de tipo Short
               para el retorno del procedimiento.
            */

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

            oCall.close();

            break;

        case JDBCConnection.DBMS_MSSQL:

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

            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 JDBCConnection.DBMS_POSTGRESQL:

```



```

/* En PostgreSQL el procedimiento k_sp_authenticate está
   definido como una función PL/pgSQL. En este caso, se usa
   un ResultSet que recoge el valor devuelto por la función
   */

oStmt = oConn.createStatement();

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

iStatus = oRSet.getShort(1);

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

default:
    throw new UnsupportedOperationException("proc. not found");
} // end switch

return iStatus;
} // authenticate

```

Procedimientos de Seguridad y Autenticación de Usuarios

Pueden encontrarse en el archivo security.ddl de la carpeta de cada SGBDR bajo com/knowgate/hipergate/datamodel/procedures.

k_get_domain_id

Obtiene un Identificador de dominio a partir de su nombre. La búsqueda será sensible a mayúsculas y minúsculas en función de cómo esté configurado el SGBDR.

NmDomain NVARCHAR	Nombre del Dominio buscado.
IdDomain INTEGER OUT	Id. del dominio o cero (0) si no se encontró.

k_get_workarea_id

Obtiene un GUID de un Área de Trabajo a partir de su nombre. La búsqueda será sensible a mayúsculas y minúsculas en función de cómo esté configurado el SGBDR.

NmWorkArea NVARCHAR	Nombre del Dominio buscado.
IdDomain INTEGER	Identificador del dominio al que pertenece el Área de Trabajo

IdWorkArea CHAR OUT	GUID del Área de Trabajo buscada o NULL si no se encontró ningún Área de Trabajo con dicho nombre.
---------------------	--

k_is_workarea_admin

Chequea si un Usuario pertenece al Grupo de Administradores de un Área de Trabajo.

IdWorkArea CHAR	GUID del Área de Trabajo.
-----------------	---------------------------

IdUser CHAR	GUID del Usuario.
-------------	-------------------

IsAdmin INTEGER OUT	1 si el Usuario pertenece al Grupo de Administradores, ó 0 en caso contrario.
---------------------	---

k_is_workarea_poweruser

Chequea si un Usuario pertenece al Grupo de Usuarios Avanzados de un Área de Trabajo.

IdWorkArea CHAR	GUID del Área de Trabajo.
-----------------	---------------------------

IdUser CHAR	GUID del Usuario.
-------------	-------------------

IsPowUser INTEGER OUT	1 si el Usuario pertenece al Grupo de Usuarios Avanzados, ó 0 en caso contrario.
-----------------------	--

k_is_workarea_user

Chequea si un Usuario pertenece al Grupo de Usuarios de un Área de Trabajo.

IdWorkArea CHAR	GUID del Área de Trabajo.
-----------------	---------------------------

IdUser CHAR	GUID del Usuario.
-------------	-------------------

IsUser INTEGER OUT	1 si el Usuario pertenece al Grupo de Usuarios, ó 0 en caso contrario.
--------------------	--

k_is_workarea_guest

Chequea si un Usuario pertenece al Grupo de Invitados de un Área de Trabajo.

IdWorkArea CHAR	GUID del Área de Trabajo.
-----------------	---------------------------

IdUser CHAR	GUID del Usuario.
-------------	-------------------

IsGuest	INTEGER	OUT	1 si el Usuario pertenece al Grupo de Invitados, ó 0 en caso contrario.
---------	---------	-----	---

k_get_user_from_email

Obtiene el GUID de un Usuario a partir de su dirección de e-mail principal.

TxMainEmail	VARCHAR		Dirección de e-mail tal y como aparece en el campo tx_main_email de la tabla k_users.
-------------	---------	--	---

Iduser	CHAR	OUT	GUID del Usuario o NULL si no se encontró ningún Usuario con dicho e-mail.
--------	------	-----	--

k_get_user_from_nick

Obtiene el GUID de un Usuario a partir de su nombre abreviado (nick name). A diferencia de los e-mails que son únicos en toda la tabla k_users, los nicknames pueden estar repetidos en distintos dominios.

IdDomain	INTEGER		Identificador del dominio al cual pertenece el usuario buscado.
----------	---------	--	---

TxNick	NVARCHAR		Nickname de usuario..
--------	----------	--	-----------------------

<u>IdUser</u>	CHAR	OUT	GUID del Usuario o NULL si no se encontró ningún Usuario con dicho nickname en el dominio especificado.
---------------	------	-----	---

k_get_group_id

Obtiene el GUID de un Grupo a partir de su nombre.

IdDomain	INTEGER		Identificador del dominio al cual pertenece el grupo buscado.
----------	---------	--	---

NmGroup	NVARCHAR		Nombre del Grupo.
---------	----------	--	-------------------

IdGroup	CHAR	OUT	GUID del Grupo o NULL si no se encontró ningún Grupo con dicho nickname en el dominio especificado.
---------	------	-----	---

k_sp_authenticate

Verifica un par usuario/clave para dilucidar si el usuario puede tener acceso al sistema. La clave es sensible a mayúsculas/minúsculas.

IdDomain	CHAR		GUID del Usuario a autenticar.
----------	------	--	--------------------------------

PwdText NVARCHAR	Clave de Acceso.
CoStatus SMALLINT OUT	Resultado de la autenticación:
1	El usuario y clave son válidos.
-1	No se encuentra ningún usuario con dicho GUID.
-2	La clave para el usuario no es válida.
-3	El usuario está desactivado (el campo <code>bo_active</code> es cero).
-8	La cuenta de usuario a expirado (la fecha actual es posterior al campo <code>dt_cancel</code>).
-9	La clave a expirado.

k_sp_del_group

Borra un Grupo de Usuarios.

IdGroup CHAR	GUID del Grupo a borrar.
--------------	--------------------------

k_sp_del_user

Borra un Usuario de la tabla `k_users` y de los grupos de permisos a los que perteneciese.

IdUser CHAR	GUID del Usuario a borrar.
-------------	----------------------------

☞ No debe usarse este método para borrar directamente un Usuario en un entorno de producción. Muchos registros de otros submódulos llevan asociado el GUID del Usuario que los creó y, si no se borran primero todos los objetos en la base de datos propiedad del Usuario, se producirá un error de violación de clave foránea cuando se intente borrar el Usuario.

El método Java `com.knowgate.acl.ACLUser.delete()` puede borrar objetos adicionales en el modelo de datos, pero, aún así este método también será incapaz de detectar referencias adicionales no estándar del modelo antes de borrar un Usuario.

Lo es preferible no borrar nunca los usuarios, sino simplemente desactivarlos poniendo el campo `k_users.bo_active` a cero y `k_users.dt_cancel` a la fecha actual.

En el caso de que realmente sea necesario eliminar físicamente los registros de usuario de la base de datos, se recomienda escribir un procedimiento previo que borre los objetos propiedad del usuario.

Procedimientos de Gestión de Categorías

k_sp_get_cat_id

Obtiene el GUID de una Categoría a partir de su nombre.

NmCategory NVARCHAR	Nombre de la Categoría.
IdCategory CHAR OUT	GUID de la Categoría o NULL si no se encontró ninguna Categoría con dicho nombre.

k_sp_cat_descendant

Verifica si una Categoría es descendiente de otra.

IdCategory CHAR	GUID de Categoría descendente.
IdAncestor CHAR	GUID de Categoría ascendente.
BoChild SMALLINT OUT	GUID 1 la Categoría IdCategory es descendente de IdAncestor ó 0 si no lo es.



Esta función no existe en la versión de PostgreSQL.

k_sp_cat_level

Obtiene el nivel de profundidad de una categoría en el árbol.

IdCategory CHAR	GUID de Categoría.
CatLevel INTEGER OUT	Nivel de profundidad de la Categoría (1 para categorías raíz).

k_sp_del_category

Borra una Categoría.

IdCategory CHAR	GUID de Categoría a borrar.
-----------------	-----------------------------



Dado que una Categoría puede tener un conjunto arbitrario de objetos asociados a través de la tabla k_x_cat_objs, se deben borrar siempre las categorías desde la clase Java com.knowgate.hipergate.Category y no directamente desde la base de datos. Si se borra directamente una Categoría de la base de datos es posible dejar zombies los objetos asociados que tenga, por ejemplo, productos, documentos o foros contenidos dentro de la misma.

k_sp_del_category_r

Borra una Categoría y todas sus descendientes.

IdCategory CHAR GUID de Categoría a borrar.

k_sp_get_cat_path

Compone una ruta simbólica a la Categoría concatenando los campos nm_category de todas sus antecesoras separadas por la barra de dividir '/'. Este procedimiento se utiliza para crear rutas que sirvan para almacenar archivos asociados a cada categoría dentro de un directorio en el disco, sin que los nombres de archivo de una categoría colisionen con los de otra al estar ubicados en directorios diferentes.

IdCategory CHAR GUID de Categoría.

CatPath NVARCHAR OUT Ruta a la Categoría.

k_sp_cat_obj_position

IdCategory CHAR GUID de Categoría.

OdPosition INTEGER OUT Posición del objeto dentro de la Categoría o NULL si la categoría no existe o el objeto no se encuentra dentro de la categoría.

k_sp_cat_expand

Expande todas las descendientes de una categoría en la tabla k_cat_expand.

En algunos casos es conveniente tener pre-expandidas todas las categorías descendientes de una determinada para acelerar los procesos de recorrido de todos los hijos y nietos.

Este procedimiento borra todos los descendientes que hubiese anteriormente en la tabla k_cat_expand y los sustituye por aquellos encontrados en la tabla k_cat_tree en un proceso recursivo de búsqueda descendente.

IdCategory CHAR GUID de Categoría a expandir.

☞ La implementación utilizada para la expansión varía de una base de datos a otra. En Oracle se expanden los descendientes mediante una sentencia SELECT ... FROM k_cat_tree START WITH gu_parent_cat = ? CONNECT BY gu_parent_cat = PRIOR gu_child_cat. En SQL Server se utiliza una tabla

temporal a modo de pila. En PostgreSQL se utilizan llamadas recursivas a la función `k_sp_cat_expand_node`.

k_sp_cat_usr_perm

Devuelve los permisos que tiene un usuario para una categoría, teniendo en cuenta los grupos a los que pertenece y los permisos otorgados a las categorías padre si es que no hay asignación explícita de permisos al usuario para la categoría especificada.

<code>IdUser</code>	<code>CHAR</code>	GUID del Usuario.
<code>IdCategory</code>	<code>CHAR</code>	GUID de la Categoría.
<code>AclMask</code>	<code>INTEGER OUT</code>	Máscara de permisos del usuario para la Categoría. Es una combinación OR bit a bit de los valores posibles del campo <code>k_lu_permissions.bit_mask</code> .

k_sp_cat_del_grp

Borra los permisos de un Grupo sobre una Categoría.

<code>IdCategory</code>	<code>CHAR</code>	GUID de la Categoría.
<code>IdGroup</code>	<code>CHAR</code>	GUID del Grupo.
<code>Recurse</code>	<code>SMALLINT</code>	Indica si hay que aplicar recursivamente el borrado de permisos a las categorías descendientes.
<code>Objects</code>	<code>SMALLINT</code>	Este campo no se usa actualmente. Debe ser 0.

k_sp_cat_del_usr

Borra los permisos de un Usuario sobre una Categoría.

<code>IdCategory</code>	<code>CHAR</code>	GUID de la Categoría.
<code>IdUser</code>	<code>CHAR</code>	GUID del Usuario.
<code>Recurse</code>	<code>SMALLINT</code>	Indica si hay que aplicar recursivamente el borrado de permisos a las categorías descendientes.
<code>Objects</code>	<code>SMALLINT</code>	Este campo no se usa actualmente. Debe ser 0.

k_sp_cat_set_grp

Asigna permisos de a Grupo sobre una Categoría. Si el grupo ya tenía permisos sobre la Categoría se producirá un error de clave primaria duplicada en la tabla `k_x_cat_group_acl`.

<code>IdCategory CHAR</code>	GUID de la Categoría.
<code>IdGroup CHAR</code>	GUID del Grupo.
<code>Recurse SMALLINT</code>	Indica si hay que aplicar recursivamente la asignación de permisos a las categorías descendientes.
<code>Objects SMALLINT</code>	Este campo no se usa actualmente. Debe ser 0.

k_sp_cat_set_usr

Asigna permisos de a Usuario sobre una Categoría. Si el usuario ya tenía permisos sobre la Categoría se producirá un error de clave primaria duplicada en la tabla `k_x_cat_user_acl`.

<code>IdCategory CHAR</code>	GUID de la Categoría.
<code>IdUser CHAR</code>	GUID del Usuario.
<code>Recurse SMALLINT</code>	Indica si hay que aplicar recursivamente la asignación de permisos a las categorías descendientes.
<code>Objects SMALLINT</code>	Este campo no se usa actualmente. Debe ser 0.

k_sp_get_user_mailroot

Devuelve la categoría raíz de correo de un usuario. La categoría raíz de correo está siempre inmediatamente bajo la categoría de inicio del usuario referenciada en la columna `gu_category` de la tabla `k_users`. La categoría raíz de correo se identifica siguiente un convenio de nomenclatura para la columna `nm_category` que debe ser: *DOMINIO_nickname_mail* donde *DOMINIO* es `k_domains.nm_domain` y *nickname* es `of k_users.tx_nickname`.

☞ Dado que el campo `tx_nickname` se utiliza para componer el nombre de la categoría raíz de correo, si se cambia después de crear el usuario se perderá el enlace del usuario a su carpeta raíz de correo anterior .

<code>GuUser CHAR</code>	GUID del Usuario.
<code>GuCategory CHAR OUT</code>	GUID de la categoría raíz de correo o NULL si no se encontró.

k_sp_get_user_mailfolder

Devuelve una carpeta de correo de un usuario. Las carpetas de correo deben ser categorías hijas de primer nivel de la categoría raíz de correo del usuario.

GuUser CHAR	GUID del Usuario.
NmFolder VARCHAR	Nombre de la Carpeta. Puede ser igual que nm_category de k_categories. O un nombre abreviado como : inbox, outbox, drafts, deleted, sent, spam, received o templates.
GuCategory CHAR OUT	GUID de la carpeta o NULL si no se encontró.

Procedimientos de Trabajo en Grupo

Pueden encontrarse en el archivo addrbook.ddl de la carpeta de cada SGBDR bajo com/knowgate/hipergate/datamodel/procedures.

k_sp_del_meeting

Elimina una Actividad.

MeetingId CHAR	GUID de la Actividad a eliminar.
----------------	----------------------------------

k_sp_del_fellow

Elimina una Persona. La Persona se eliminará de todas las Actividades pasadas y futuras a las que estuviese asociada.

FellowId CHAR	GUID de la Person a eliminar.
---------------	-------------------------------

k_sp_del_room

Elimina un Recurso Compartido. Si el recurso está en uso en alguna Actividad se producirá un error de clave foránea violada.

RoomNm NVARCHAR	Nombre del Recurso.
WorkAreaId CHAR	Área de Trabajo a la que pertenece.

Procedimientos de Gestión de Productos

Pueden encontrarse en el archivo products.ddl de la carpeta de cada SGBDR bajo com/knowgate/hipergate/datamodel/procedures.

k_sp_del_product

Elimina un Producto.

ProductId CHAR GUID del Producto a eliminar.

☞ En general, hay que eliminar los productos utilizando el método `delete()` de la clase Java `com.knowgate.hipergate.Product` y no una llamada directa a `k_sp_del_product`. El los procedimientos almacenados en base de datos no pueden borrar los archivos físicos ni las imágenes asociadas al Producto. Por consiguiente, si se llama directamente al procedimiento es posible dejar archivos zombie en el disco duro.

Procedimientos de Gestión de Ventas y Relaciones con Clientes

Pueden encontrarse en el archivo crm.ddl de la carpeta de cada SGBDR bajo com/knowgate/hipergate/datamodel/procedures.

k_sp_del_sales_man

Elimina un Vendedor.

SalesManId CHAR GUID del Vendedor a eliminar.

☞ Los Vendedores son subregistros de los usuarios del Dominio. Cuando se elimina un Vendedor, las Compañías que tenía asignadas quedan libres de cualquier asignación. Eliminar un vendedor no borra su Usuario correspondiente en el Dominio. Los pedidos asignados al vendedor tampoco se borran ni se alteran, el GUID del vendedor en el Pedido permanece, aunque el vendedor en si mismo deje de existir.

k_sp_del_contact

Elimina un Contacto.

ContactId CHAR GUID del Contacto a eliminar.

☞ En general, hay que eliminar los contactos utilizando el método `delete()` de la clase Java `com.knowgate.crm.Contact` y no una llamada directa a `k_sp_del_contact`. El los procedimientos almacenados en base de datos no pueden borrar los archivos físicos adjuntos al contacto. Por consiguiente, si se llama directamente al procedimiento es posible dejar archivos zombie en el disco duro.

k_sp_del_company

Elimina una Compañía.

CompanyId CHAR GUID de la Compañía a eliminar.

☞ Para eliminar una Compañía es necesario eliminar primero manualmente sus Contactos internos o se producirá un error de clave foránea.

k_sp_del_opportunity

Elimina una Oportunidad.

OpportunityId CHAR GUID de la Oportunidad a eliminar.

Procedimientos de Gestión de Listas

Pueden encontrarse en el archivo lists.ddl de la carpeta de cada SGBDR bajo com/knowgate/hipergate/datamodel/procedures.

k_sp_del_list

Elimina una Lista y todos sus Miembros.

ListId CHAR GUID de la Lista a eliminar.

k_sp_email_blocked

Verifica si una dirección de correo está en la [lista negra](#) asociada a una lista de distribución.

GuList CHAR(32) GUID de la lista base de distribución.

TxEmail VARCHAR(100) e-mail a verificar.

BoBlocked SMALLINT OUT 1 si TxEmail se encuentra en la [lista negra](#) asociada a la lista GuList, 0 en caso contrario.

k_sp_contact_blocked

Verifica si un Contacto está en la [lista negra](#) asociada a una lista de distribución.

GuList CHAR(32) GUID de la lista base de distribución.

GuContact CHAR(32) Contacto a verificar.

BoBlocked SMALLINT OUT 1 si GuContact se encuentra en la [lista negra](#) asociada a la lista GuList, 0 en caso contrario.

k_sp_company_blocked

Verifica si una Compañía está en la [lista negra](#) asociada a una lista de distribución.

GuList CHAR(32) GUID de la lista base de distribución.
GuCompany CHAR(32) Compañía a verificar.
BoBlocked SMALLINT OUT 1 si GuCompany se encuentra en la [lista negra](#) asociada a la lista GuList, 0 en caso contrario.

k_sp_del_duplicates

Borra los emails duplicados de una lista estática o directa.

ORACLE Este procedimiento almacenado no está disponible en Oracle, utilizar en su lugar

```
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)
```

ListId CHAR GUID de la lista base a desduplicar.
Deleted INTEGER OUT Cuenta de emails repetidos eliminados.

Disparadores de Miembros de Listas

La tabla [k_member_address](#) contiene una vista materializada todas las direcciones de Contactos y Compañías. k_member_address contiene información replicada de k_contacts, k_companies y k_addresses pero en una única tabla lo que elimina la necesidad de hacer JOINS para recuperar la información y facilita la creación de índices.

La tabla k_member_address se mantiene actualizada automáticamente mediante disparadores de acuerdo a las siguientes reglas:

1. Cuando se añade una dirección en la tabla k_addresses con el campo bo_active=1 se añade un registro a la tabla k_member_address.
2. Cuando se actualiza una dirección en la tabla k_addresses con el campo bo_active=1 se actualiza o añade su registro correspondiente en

la tabla `k_member_address`. Cuando se actualiza una dirección `bo_active=0` se borra el registro correspondiente en `k_member_address`.

3. Cuando se asocia una dirección a una compañía escribiendo en la tabla `k_x_company_addr`, se actualiza la información de la compañía en las direcciones correspondientes en `k_member_address`.

4. Cuando se asocia una dirección a un contacto escribiendo en la tabla `k_x_contact_addr`, se actualiza la información del contacto en las direcciones correspondientes en `k_member_address`.

5. Cuando se borra una dirección de `k_addresses` se borra también de `k_member_address`.

6. Cuando se borra una compañía de `k_companies` se actualiza el campo `gu_company` de `k_member_address` y se pone a `NULL`.

7. Cuando se borra un contacto de `k_contacts` se actualiza el campo `gu_contact` de `k_member_address` y se pone a `NULL`.

Cómo reconstruir la vista materializada `k_member_address`



En PostgreSQL está disponible la función PL/pgSQL `k_sp_rebuild_member_address` que reconstruye todos los registros de la vista materializada `k_member_address` a partir de los registros de `k_addresses`, `k_companies` y `k_contacts`.

Procedimientos de Foros

Pueden encontrarse en el archivo `forums.ddl` de la carpeta de cada SGBDR bajo `com/knowgate/hipergate/datamodel/procedures`.

`k_sp_del_newsgroup`

Elimina un Grupo de Mensajes.

`IdNewsGroup` CHAR

GUID del Grupo de Mensajes a eliminar.

`k_sp_del_newsmg`

Elimina un Mensajes.

IdNewsMsg CHAR

GUID del Mensaje a eliminar.

☞ Los mensajes que contengan archivos binarios adjuntos hay que eliminarlos productos utilizando el método `delete()` de la clase `Java com.knowgate.forums.NewsMessage`. Una llamada directa a `k_sp_del_newsmg` en mensajes con binarios, provocará una violación de clave foránea en contra la tabla `k_products`.

Procedimientos de Gestión de Proyectos e Incidencias

Pueden encontrarse en el archivo `projtrack.ddl` de la carpeta de cada SGBDR bajo `com/knowgate/hipergate/datamodel/procedures`.

k_sp_prj_expand

Expande todos los descendientes de un Proyecto y los añade en la tabla `k_project_expand`.

StartWith CHAR

GUID del Proyecto a expandir.

k_sp_del_project

Borra un Proyecto, incluyendo todos sus descendientes, Tareas e Incidencias Asociadas.

ProjectId CHAR

GUID del Proyecto a Eliminar.

k_sp_del_duty

Borra una Tarea.

DutyId CHAR

GUID de la Tarea a Eliminar.

k_sp_del_bug

Borra una Incidencia.

BugId CHAR

GUID de la Incidencia a Eliminar.

k_sp_prj_cost

Devuelve el coste total de un proyecto sumando los costes de todos sus proyectos hijos y nietos. Se incluye el coste de las tareas y los costes explícitamente listados en la tabla `k_project_costs`.

`ProjectId` CHAR GUID del Proyecto.

Procedimientos de Gestión de Correo

Pueden encontrarse en el archivo hipermail.ddl de la carpeta de cada SGBDR bajo com/knowgate/hipergate/datamodel/procedures. También existen un par de procedimientos para obtener las carpetas de correo de un usuario en el archivo categories.ddl

k_sp_del_mime_msg

Borra un mensaje.

MimeMsgId CHAR GUID del mensaje a eliminar.

☞ Los mensajes deben eliminarse mediante el método Java `DBMimeMessage.delete()` o haciendo `DBMimeMessage.setFlag(Flags.Flag.DELETED, true)` y luego `DBFolder.expunge()`. Llamar directamente al procedimiento almacenado `k_sp_del_mime_msg` no borra los mensajes en archivos MBOX. Incluso si se almacenan los mensajes en campos LONGVARBINARY de la base de datos y no en MBOX puede haber referencias externas a archivos desde los borradores de mensaje.

k_sp_get_mime_msg

Obtiene el GUID de un mensaje a partir del identificador asignado por el proveedor de correo que lo originó.

MsgId VARCHAR Id. del mensaje

Guid CHAR OUT GUID del mensaje.

k_sp_write_inet_addr

Inserta un remitente o destinatario para un correo y lo asocia (si procede) con la entrada en las tablas de Usuarios, Contactos o Compañías correspondiente.

DomainId INTEGER Identificador numérico de Dominio.

WorkAreaId CHAR Identificador del Área de Trabajo.

MsgGuid CHAR GUID del Mensaje.

MimeMsgId VARCHAR Id. del mensaje generado por el proveedor de correo.

RecipientTp VARCHAR	Tipo de destinatario { from to cc bcc }.
EMailTx VARCHAR	Dirección de correo del destinatario.
PersonalTx NVARCHAR	Nombre completo del destinatario.

5

Clases Java

hipergate proporciona un completo conjunto de librerías Java que contienen la funcionalidad y los servicios base del producto para su uso desde cualquier programa.

Esta sección está dedicada a los conceptos generales de uso de las librerías Java de soporte de hipergate.

Para una descripción detallada de cada método consultar el API JavaDoc.

Propósito de las librerías Java

Cuando se empieza a escribir una aplicación basada en web, es tentador escribir la funcionalidad dentro de páginas de servidor JSP, PHP u otro lenguaje de scripting.

Las páginas de servidor son sencillas de escribir y no requieren de un farragoso ciclo de compilación para funcionar. Además, la propia división en páginas independientes hace mucho más sencillo el control de versiones que en la librerías compiladas monolíticas.

Por desgracia, todo el código que se escribe sobre las páginas es virtualmente imposible de re-utilizar o de re-programar para ser llamadas desde otros programas.

En hipergate, las funcionalidades base están encapsuladas en clases Java y son independientes de las páginas JSP del producto. Esto proporciona mayor modularidad, mantenibilidad, flexibilidad y eficiencia al producto.

Paquetes genéricos y paquetes dependientes del modelo

Las clases del archivo principal hipergate.jar están agrupadas en paquetes que pueden dividirse en 2 grandes grupos: los paquetes genéricos, que sirven para cualquier aplicación, y los paquetes dependientes del modelo, que requieren de una estructura específica de tablas en la base de datos para funcionar.

Los paquetes genéricos proporcionan servicios de trazas de depuración, acceso a base datos, transformaciones XSLT, manipulación de cadenas y manipulación de ficheros.

Los paquetes genéricos son:

- **com.knowgate.debug** : Trazas de depuración.
- **com.knowgate.jdbc** : Pool de conexiones a base de datos.
- **com.knowgate.dataobjs** : Objetos de acceso a datos.
- **com.knowgate.datacopy** : Copia de estructuras de datos complejas.
- **com.knowgate.dataxslt** : Transformaciones XSLT.
- **com.knowgate.dfs** : Acceso a ficheros a través de NFS y FTP.
- **com.knowgate.cache** : Caché de datos en memoria.
- **com.knowgate.misc** : Subrutinas varias.
- **com.knowgate.ole** : Wrapper POI para leer documentos OLE2.

Los paquetes dependientes del modelo encapsulan la funcional de alto nivel de hipergate.

Los paquetes dependientes del modelo son:

- **com.knowgate.acl** : Autenticación de Usuarios.
- **com.knowgate.addrbook** : Herramientas colaborativas.
- **com.knowgate.crm** : Gestión de Ventas.
- **com.knowgate.dataxslt.db** : Plantillas y Datos XML en BB.DD.
- **com.knowgate.forums** : Foros.
- **com.knowgate.hipergate** : Categorización, Productos y Tienda.
- **com.knowgate.hipergate.datamodel** : Lanzador del Modelo de Datos
- **com.knowgate.http** : Servlets para servir contenido binario.
- **com.knowgate.ldap** : Autenticar usuarios usando LDAP.
- **com.knowgate.projtrack** : Seguimiento de Proyectos e Incidencias.
- **com.knowgate.scheduler** : Planificador de Tareas.
- **com.knowgate.workareas** : Áreas de Trabajo.

Paquetes adicionales compilados dentro de 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.

Paquetes adicionales compilados fuera de hipergate.jar

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

Modalidad de depuración y modalidad de explotación

Cada versión completa del archivo hipergate.jar se distribuye en 2 modalidades, la de depuración y la de explotación.

La diferencia entre ambas estriba en que la versión de depuración deja una traza detallada en el archivo /tmp/javatrc.txt en sistemas UNIX y C:\javatrc.txt en sistemas Windows.

La variable estática `trace` de la clase `com.knowgate.debug.DebugFile` controla si deben compilarse los fuentes en modalidad de depuración o en modalidad de explotación.

No es posible cambiar de modalidad de depuración a modalidad de explotación dinámicamente en tiempo de ejecución sin cambiar el archivo hipergate.jar porque la variable `trace` está declarada como `final` para permitir al compilador eliminar el código de depuración al generar los bytecodes Java.

Como averiguar el modo de traza desde la línea de comandos

Es posible saber desde la línea de comandos si se está utilizando una versión de depuración o de explotación. Teclear:

```
java com.knowgate.debug.DebugFile
```

Aparecerá en la consola bien "Debug mode enabled" bien "Debug mode disabled".

Propiedades de inicializacion

Las propiedades de inicializacion se especifican en el archivo `hipergate.cnf`.

Para acceder a estas propiedades se utiliza la clase `com.knowgate.misc.Environment`.

Para encontrar el archivo de propiedades hay que establecer la variable de entorno del sistema operativo `KNOWGATE_PROFILES` apuntando al directorio que contenga el archivo `hipergate.cnf`.

Por ejemplo:

```
KNOWGATE_PROFILES=/opt/knowgate/  
en UNIX  
o  
SET KNOWGATE_PROFILES=C:\\knowgate\\  
en Windows
```

Una vez leídas por primera vez las propiedades de inicializacion se almacenan en memoria en la clase `Environment` hasta que se llama al método `Environment.refresh()`.

Lanzador de tablas y datos iniciales

El paquete `com.knowgate.hipergate.datamodel` contiene los elementos necesarios para crear de cero una base de datos con el modelo completo de hipergate.

Típicamente la carga inicial de datos se hace mediante un volcado directo del SGBDR utilizado.

La clase `ModelManager` puede utilizarse como método alternativo de creación de la base de datos cuando se desean crear sólo partes del modelo o cuando se está trabajando en un porting de hipergate a un nuevo SGBDR.

Todas las sentencias SQL y DDL necesarias para crear el modelo están almacenadas en ficheros contenido en subcarpetas del paquete `com.knowgate.hipergate.datamodel` dentro del archivo `hipergate.jar`, divididas según el módulo funcional y SGBDR al que pertenezcan.

Las funcionalidades de `ModelManager` son accesibles desde la línea de comandos mediante:

```
java com.knowgate.hipergate.datamodel.ModelManager ruta
comando modulo [verbose]
```

O

```
java com.knowgate.hipergate.datamodel.ModelManager ruta
comando [domain|workarea] [nombre_dominio|nombre_dominio.
nombre_workarea] [verbose]
```

O

```
java com.knowgate.hipergate.datamodel.ModelManager ruta
clone workarea dominio_origen.workarea_origen
dominio_destino.workarea_destino [verbose]
```

☞ El comando `com.knowgate.hipergate.datamodel.ModelManager ruta create all` puede tardar varios minutos en ejecutarse.

Donde:

ruta : Es la ruta completa al archivo `hipergate.cnf` (u otro equivalente) que contenga las propiedades `driver`, `dburl`, `dbuser` y `dbpassword` para conectarse a la base de datos.

comando : Debe ser `create` o `drop` según de vaya a crear o a eliminar el módulo o dominio especificado a continuación. O `clone` para clonar un Área de Trabajo.

modulo : Nombre del módulo a crear o eliminar. Debe ser un valor de los siguientes: `all`, `kernel`, `lookups`, `security`, `jobs`, `thesauri`, `categories`, `products`, `teamwork`, `webbuilder`, `crm`, `lists`, `shop`, `projtrack`, `billing`.

Si se especifica `all` se crearán o eliminarán todos los módulos.

Los módulos tienen dependencias entre ellos, si se crean o se eliminan de forma individual, se debe respetar el orden de la lista anterior.

dominio : Nombre del Dominio a crear o eliminar.

dominio.workarea : Nombre de Dominio y nombre del Área de Trabajo.

verbose : Vuelca en la salida estándar el código del SQL ejecutado.

Ejemplo 1, crear el modelo completo:

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

Ejemplo 2, eliminar el módulo de gestión de proyectos:

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

Ejemplo 3, Crear un Dominio:

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

Ejemplo 4, clonar un Área de Trabajo desde el Dominio MODEL al Dominio TEST1:

```
java com.knowgate.hipergate.datamodel.ModelManager  
/opt/knowgate/hipergate.cnf clone workarea  
MODEL.model_default TEST1.test1_workingarea
```

☞ Para que el lanzador de tablas funcione es necesario tener instalado el driver JDBC que esté referenciado en el archivo hipergate.cnf y tener bien configurada la conectividad con la base de datos

Pool de Conexiones a base de datos

El paquete `com.knowgate.jdbc` proporciona un pool de conexiones a base de datos y un wrapper para que la clase `java.sql.Connection` pueda manejarse desde el pool.

Es recomendable utilizar siempre el pool, no sólo porque mejora el rendimiento y el uso de recursos, sino porque permite mantener de forma centralizada la información sobre las conexiones abiertas contra la base de

datos y detectar con mayor facilidad aquellas que se quedan bloqueadas o abiertas indefinidamente.

Mapeo de objetos Java a registros de base de datos

Existen muchas implementaciones diferentes en el mercado de sistemas de persistencia para objetos en base de datos relacional.

hipergate utiliza un paquete propietario (`com.knowgate.dataobjs`) para esta misión.

`dataobjs` es un paquete de persistencia diseñado para ser sencillo, rápido y fácil de usar a cambio de algunas restricciones de diseño:

- Un objeto persistente sólo puede estar compuesto de propiedades simples mapeables directamente a campos de la base de datos.
- Cada instancia de objeto debe corresponderse con un único registro de una tabla de la base de datos.
- El objeto persistente no puede contener referencias a otros objetos.
- Las propiedades del objeto persistente se llaman igual que los campos de la base de datos que las contienen.
- Cada objeto tiene una clave de uno o varios campos que se corresponde con la clave primaria de la tabla de almacenamiento en la base de datos.

Con este enfoque resulta muy sencillo mapear objetos Java a registros de base de datos con un mínimo de programación.

Por ejemplo:

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");
}
}

```

Ahora las propiedades que se asignen a las instancias de MyObject se grabarán en la base de datos al llamar al método store().

Basta con hacer:

```

com.knowgate.dataobjs.DBBind oDBB;
com.knowgate.jdc.JDCCConnection oCon;
MyObject oObj;

// Cargar los metadatos tomando la conexión de hipergate.cnf

oDBB = new DBBind();

try {
    // Obtener conexión del pool, asignarle nombre "my_object_test"

    oCon = oDBB.getConnection("my_object_test");

    // Instanciar el objeto heredado de DBPersist

    oObj = new MyObject();

    // Rellenar propiedades que se llamen igual que los campos de la
    base de datos

    oObj.put ("id_object", "A12345678");
    oObj.put ("tx_object", "mi primer objeto hipergate");

    // Grabar el registro en la tabla my_object_table
    // No es necesario preocuparse de si el registro ya existía o
    // es nuevo, la clase DBPersist maneja las inserciones y
    // actualizaciones automáticamente.

    oObj.store(oCon);

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

oObj = new MyObject();

// Cargar el objeto desde la BB.DD.
boolean bLoaded = oObj.load (oCon, new Object[]{ "A12345678" });

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

// Cerrar la conexión con el mismo nombre con el que se abrió para
que el recolector de estadísticas pueda monitorizarla.

oCon.close("my_object_test");

oCon = null;

```


☞ Para que los métodos `load()` y `store()` funcionen es necesario que la tabla base tenga una clave primaria definida.

Nombres de Tablas y Campos

Los nombres de tablas y campos están centralizados en constantes estáticas en la clase `com.knowgate.dataobjs.DB`.

Es posible cambiar los nombres de tablas y campos del modelo a nivel de las librerías simplemente cambiándolos en la clase `DB`.

No obstante, un proceso genérico de cambio de nombres requiere también revisar manualmente todos los formularios web que editan y graban datos.

Cache en RAM de metadatos del SGBDR

Es posible conectar el pool `com.knowgate.jdc.JDCCConnectionPool` directamente a la base de datos. No obstante, el acceso suele obtenerse a través del objeto singleton `com.knowgate.dataobjs.DBBind`. `DBBind` obtiene la información de conexión a la base de datos del archivo de propiedades `hipergate.cnf`. Este archivo debe encontrarse en la ruta apuntada por la variable de entorno `KNOWGATE_PROFILES` (ver Manual de Instalación).

Las conexiones típicamente se obtienen a través del método `DBBind.getConnection()` que encapsula a `JDCCConnection.getConnection()`.

`DBBind` añade una capa adicional que mantiene información cacheada en memoria acerca de la estructura de las tablas en la base de datos (metadatos).

Los metadatos son utilizados por los objetos `DBPersist`, y sus herederos, para decidir automáticamente como persistir información en la base de datos sin necesidad de parametrización adicional entre Java y el SGBDR.

Transacciones

Es cometido de la aplicación cliente el manejo de las transacciones.

Cada método de DBPersist recibe una conexión abierta contra la base de datos y realiza las operaciones encomendadas sin acometer ninguna transacción.

Es necesario prestar atención a cómo acometer o descartar transacciones en cada página. Dado que una conexión no se cierra realmente cuando se llama al método `JDBCConnection.close()`, una página puede dejar operaciones pendiente y bloquear la siguiente página. Para evitar este problema se recomienda:

- a) Usar `Connection.setAutoCommit(true)` antes de realizar ninguna operación en una página que escriba en la base de datos o
- b) Usar `Connection.setAutoCommit(false)` antes de realizar ninguna operación en una página que escriba en la base de datos y posteriormente realizar siempre `Connection.commit()` o `Connection.rollback()` antes de finalizar la página.

Auditoría

La auditoría de operaciones es responsabilidad de la aplicación cliente. hipergate proporciona la clase `DBAudit` para operaciones básicas de grabación en la tabla `k_auditing`.

No obstante, no se recomienda emplear misma base de datos de explotación para guardar registros de auditoría.

Lo mejor es volcar los registros de auditoría directamente a un fichero de texto.

Manejo de campos largos

Los campos largos pueden escribirse directamente como Strings (para los `LONGVARCHAR` y `CLOB`) o asociarse a archivos de disco.

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.*;
```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```

public class MyLongObject extends DBPersist {

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

```

```

// Ejemplo de grabación de un campo largo desde un archivo

com.knowgate.dataobjs.DBBind oDBB;
com.knowgate.jdc.JDCCConnection 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;

```

```

// Ejemplo de grabación de un campo largo desde un array

com.knowgate.dataobjs.DBBind oDBB;
com.knowgate.jdc.JDCCConnection 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;

```

Carga de Datos XML

La clase DBPersist proporciona el método `parseXML()` para cargar campos desde un archivo en formato XML.

Archivo 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 & international 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"));
```

Sólo se cargan en el DBPersist los valores del XML que correspondan a columnas definidas en la base de datos.

Durante el proceso de conversión el parser intenta convertir cada elemento XML de tipo cadena en el tipo interno de la columna correspondiente. Por consiguiente, si las fechas o los números están mal formateados el parser levantará una excepción.

Conjuntos de filas leídos a ráfagas

La filosofía general de las páginas JSP de hipergate es minimizar el tiempo que una conexión a base de datos permanece ocupada en cada petición de servicio.

La clase principal que se utiliza para este fin es

```
com.knowgate.dataobjs.DBSubset.
```

DBSubset es un array bidimensional en memoria que contiene un subconjunto de filas de una tabla leídas de una sola vez de la base de datos.

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

```
DBBind oDBB = new DBBind();
```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```

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

// Ejemplo DBSubset sin parámetros

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;

// Ejemplo DBSubset con parámetros

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;

// Ejemplo con un JOIN de 2 tablas y limitando el número máximo de
resultados a 100

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

// El último parámetro del constructor no limita el número de
registros a leer, sino que es sólo un indicador del tamaño
preferido de ráfaga de lectura. Para limitar el número de filas
leídas hay que establecer la propiedad MaxRows.

oCur.setMaxRows(100);

int iUserCount = oCur.load (oCon);

oCur = null;

oCon.close("dbsubset_test ");

oCon = null;

```

☞ Si no se limita el número máximo de filas a leer, el DBSubset cargará en memoria todas las filas de la base de datos que cumplan el criterio de la cláusula WHERE.

Uso de la clase ImportExport para cargar texto delimitado

La clase `com.knowgate.hipergate.datamodel.ImportExport` toma como entrada un archivo de texto delimitado y un descriptor de formato de entrada, y carga los datos del archivo de texto delimitado en la tabla o tablas convenientes.

☞ También posible usar la clase Java `com.knowgate.hipergate.datamodel.ModelManager` si lo que se desea es cargar un archivo de texto que tiene el mismo número de columnas que una tabla dada.

`ImportExport` trabaja creando instancias de clases que implementen el interfaz `com.knowgate.hipergate.datamodel.ImportLoader`. La clase concreta que se instancia depende de las instrucciones recibidas en el descriptor de formato de entrada.

`ImportExport` tiene un único método público llamado `perform()` que recibe como parámetro el descriptor. La sintaxis general para el descriptor es:

```
[APPEND|UPDATE|APPENDUPDATE]
[CONTACTS|COMPANIES|USERS|PRODUCTS|FELLOWS] CONNECT usuario
TO "cadena de conexión" IDENTIFIED BY contraseña SCHEMA
esquema WORKAREA "área de trabajo" [CATEGORY "nombre
categoría"] 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]
(definición de columna, definición de columna, ...)
```

```
definición de columna:= nombre [CHAR | VARCHAR | DATE
"formato fecha"| SMALLINT | INTEGER | FLOAT | DOUBLE |
NUMERIC]
```

El formato de fecha debe ser uno de los aceptados por [SimpleDateFormat](#).

Explicación de cada palabra clave:

APPEND, UPDATE o APPENDUPDATE: Modo de inserción de datos. Si es APPEND se insertarán los registros que no existan en la base de datos y para los que existan se producirá un error de clave primaria duplicada.

CONTACTS, COMPANIES, USERS, FELLOWS o PRODUCTS: Nombre de la entidad de alto nivel que se está intentado cargar. Actualmente sólo se soporta CONTACTS, COMPANIES, USERS, FELLOWS y PRODUCTS que instanciarán:

- `com.knowgate.crm.ContactLoader` para cargar las tablas `k_companies`, `k_contacts` y `k_addresses`.
- `com.knowgate.crm.CompanyLoader` para cargar las tablas `k_companies`, y `k_addresses`
- `com.knowgate.acl.UserLoader` para cargar la tabla `k_users`
- `com.knowgate.hipergate.FellowLoader` para cargar `k_users` y `k_fellows`
- `com.knowgate.hipergate.ProductLoader` para cargar `k_products`.

Las compañías se casan por el valor del campo `nm_legal`, es decir, si se intenta insertar una compañía con el mismo valor para el campo `k_companies.nm_legal` que otra anterior, se asumirá que es la misma.

Los individuos se casan por el campo `sn_passport`, es decir, si se intenta insertar un individuo con el mismo valor para el campo `k_contacts.sn_passport` que otro anterior, se asumirá que es el mismo.

Las direcciones se casan por el campo `ix_address`.

Los usuarios y los empleados se casan por e-mail o por la combinación `dominio+nickname`

Los productos se casan por el campo `gu_product`.

CONNECT *usuario* TO *cadena de conexión* IDENTIFIED BY *contraseña* SCHEMA *esquema*: Especificación completa de la base de datos a la que hay que conectarse y con qué usuario y contraseña. El parámetro *cadena de conexión*, es una cadena de conexión estándar JDBC, la que corresponda para cada SGBDR.

WORKAREA “*área de trabajo*”: Nombre o GUID del área de trabajo donde se insertarán los datos de entrada.

CATEGORY "*nombre categoría*": GUID o Nombre (k_categories.nm_category) de la categoría donde se insertarán los datos de entrada. Este parámetro sólo es válido si se están cargando productos.

INPUTFILE "*ruta archivo entrada*": Ruta completa del archivo con los datos de entrada.

BADFILE "*ruta archivo salida*": Ruta completa del archivo donde se escribirá el detalle de los errores que se produzcan.

DISCARDFILE "*ruta archivo salida*": Ruta completa del archivo donde se escribirán las líneas que no hayan podido cargarse. En este archivo no se escribe ningún texto de error sólo las líneas fallidas literalmente.

CHARSET *juego de caracteres*: Juego de caracteres del archivo de entrada. El juego de caracteres debe ser uno de los soportados por Java. Para la lista completa ver <http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html>.

ROWDELIM *delimitador*: Delimitador de filas, puede ser un carácter entre comillas o una de las palabras clave CR, LF o CRLF. Para retorno de carro, salto de línea (Unix) o retorno de carro+salto de línea (DOS).

COLDELIM *delimitador*: Delimitador de columnas. Puede ser un carácter entre comillas o una de las palabras clave TAB para el tabulador.

RECOVERABLE: Indica que el proceso de carga debe ejecutarse dentro de una única transacción. Si tan sólo una de las líneas falla en cargar todo el proceso se descarta y la base de datos queda intacta. Esta opción puede desbordar los segmentos de rollback de la base de datos si la carga es demasiado grande.

UNRECOVERABLE: El proceso de carga hará commit por fila. Este es el valor por defecto.

PRESERVE SPACE: Indica que se deben respetar los espacios en blanco a la derecha de cada campo.

INSERTLOOKUPS: Indica que se deben dar de alta los valores nuevos en las tablas de remonte.

WITHOUT DUPLICATED [NAMES|EMAILS]: Esta cláusula sólo está permitida en el caso de que lo que se esté cargando sean contactos. Evita que se carguen nombre y apellidos duplicados o e-mails duplicados.

Ejemplo de carga de contactos:

Descriptor de archivo

```
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)
```

Archivo de entrada

```
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
```

Código Java:

```
ImportExport oImpExp = new ImportExport();
oImpExp.perform("APPENDUPDATE CONTACTS CONNECT knowgate TO
\"jdbc:postgresql://192.168.1.10:10801/hgoltp8t\" IDENTIFIED
BY password WORKAREA test_default INPUTFILE
```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```

"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)");

```

Ejemplo de carga de usuarios:

Código Java:

```

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)");

```

Archivo de entrada

```

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;piibox;ignore;pi.boxaderas@hg.com;PEDRO I.;BOXADERAS;Delegado
2050;TEST / Users;205715;jfuset;ignore;Jordi.Fuset@hg.com;JORDI FUSESET;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

```

Ejemplo de carga de productos:

Archivo de entrada:

```

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

```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

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

Descriptor de Archivo :

```
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 es el nombre único de la categoría donde deben insertarse los productos tal y como figura en la columna nm_category de la tabla k_categories. El GUID de la categoría destino puede ser también usado en vez del nombre único como valor para el parámetro **CATEGORY**.

Código Java:

```
ImportExport oImpExp = new ImportExport();
oImp.perform("APPENDUPDATE PRODUCTS CONNECT knowgate TO
\"jdbc:postgresql://192.168.1.10:5432/hgo1tp2d\" 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)");
```

Cómo funciona el cargador de usuarios y empleados

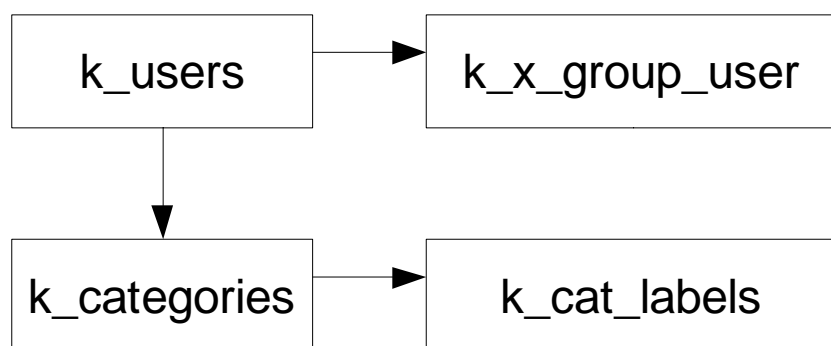
Los usuarios y los empleados de hipergate son dos entidades estrechamente vinculadas en el modelo de datos.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

Los usuarios se cargan en la tabla `k_users` y los grupos a los que pertenecen se cargan en `k_x_group_user`. Asimismo, la clase `com.knowgate.acl.UserLoader` (que es la que realiza la carga de usuarios) llama también al script `com/knowgate/hipergate/datamodel/scripts/user_categories_create.js` a través de la clase `com.knowgate.hipergate.datamodel.ModelManager` para la creación de las categorías por defecto del usuario en la biblioteca virtual y en el WebMail.

Cuando está activado el modo de inserción o actualización, dos usuarios se consideran el mismo si coinciden en dominio y nickname.

Esquema de datos cargados durante la importación de un usuario desde un fichero de texto simple.



Archivo de entrada:

```

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
  
```

Código Java:

```

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

El primer parámetro en el archivo de entrada es el nombre del grupo en la tabla `k_acl_groups`, aunque también es posible especificar directamente el GUID del grupo al que pertenezca el usuario. En la carga

de usuarios desde ficheros sólo es posible especificar un grupo de permisos.

El área de trabajo a la cual deba conectarse el usuario por defecto puede especificarse globalmente con la palabra reservada `WORKAREA` o una por una diferente para cada usuario. Al igual que el grupo, el área de trabajo puede especificarse por nombre (`nm_workarea`) o GUID (`gu_workarea`).

El identificador numérico del dominio (`id_domain`) puede ser inferido del área de trabajo, especificarse explícitamente en cuyo caso debe ser consistente con el área de trabajo, es decir, el área de trabajo debe pertenecer al dominio dado.

Los empleados se cargan en la tabla `k_fellows` y en sus tablas de remonte `k_fellows_lookup` y `k_lu_fellow_titles`.

Aunque no es imprescindible para preservar la integridad referencial del modelo de datos, la carga de un empleado precarga previamente un usuario. Esto lo realiza automáticamente la clase `com.knowgate.addrbook.FellowLoader`.

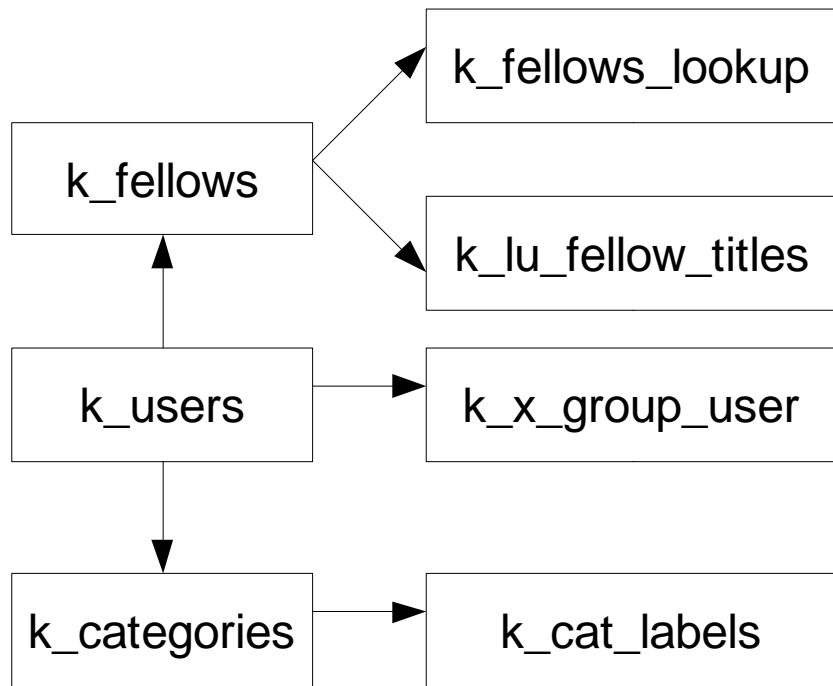
Algunos campos de la tabla `k_fellows` son redundantes con `k_users` y, aunque pueden cargarse con diferentes valores, se recomienda cargar los mismos. En particular:

<code>k_fellows</code>	<code>k_users</code>
<code>tx_name</code>	<code>nm_user</code>
<code>tx_surname</code>	<code>tx_surname1</code> + " " + <code>tx_surname2</code>
<code>tx_company</code>	<code>nm_company</code>
<code>tx_email</code>	<code>tx_main_email</code>

Si se especifica sólo uno de estos dos pares, entonces el que es nulo toma por defecto el valor del otro.

En las tablas `k_fellows_lookup` y `k_lu_fellow_titles` solo se escribirá si se especifica el modificador `INSERTLOOKUPS`.

Esquema de datos cargados durante la importación de un empleado desde un fichero de texto.



Archivo de entrada:

```

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
  
```

Código Java:

```

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

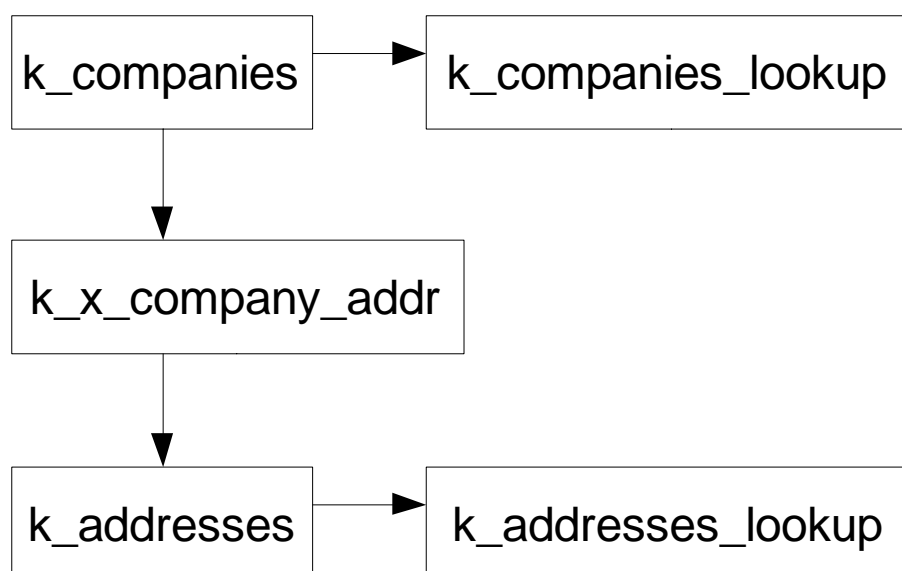
Cómo funciona el cargador de compañías

El cargador de compañías se encuentra en la clase `com.knowgate.crm.CompanyLoader`. Dicha clase escribe en la tabla `k_companies` y, dependiendo de los valores de los indicadores `WRITE_LOOKUPS` y `WRITE_ADDRESSES` para el método `store()`,

también en las tablas { `k_companies_lookup`, `k_addresses_lookup` } y { `k_addresses`, `k_x_company_addr` }

Si la llamada se realiza desde el método `perform()` de la clase `ImportExport`, entonces el comportamiento por defecto es no cargar los lookups (a menos que se especifique el modificador `INSERTLOOKUPS`) y si cargar las direcciones.

Esquema de datos cargados durante la importación de una compañía desde un fichero de texto.



☞ En las compañías no está permitido que la razón social (campo `nm_legal` de la tabla `k_companies`) esté duplicada en una misma área de trabajo.

Cómo funciona el cargador de contactos

El cargador de contactos se encuentra en la clase `com.knowgate.crm.ContactLoader`. Dicha clase escribe en la tabla `k_contacts`. Si en el método `store()` se especifica el indicador `WRITE_COMPANIES` entonces se escribe también en la tabla `k_companies`. Y si se especifica `WRITE_ADDRESSES` entonces se escribe también en la tabla `k_addresses`. El comportamiento por defecto desde el método `perform()` de la clase `ImportExport` es cargar tanto las compañías como las direcciones al cargar un contacto, pero no los lookups a menos que se especifique explícitamente la palabra reservada `INSERTLOOKUPS` que activa el indicador `WRITE_LOOKUPS` en el método `store()` de `ContactLoader`.

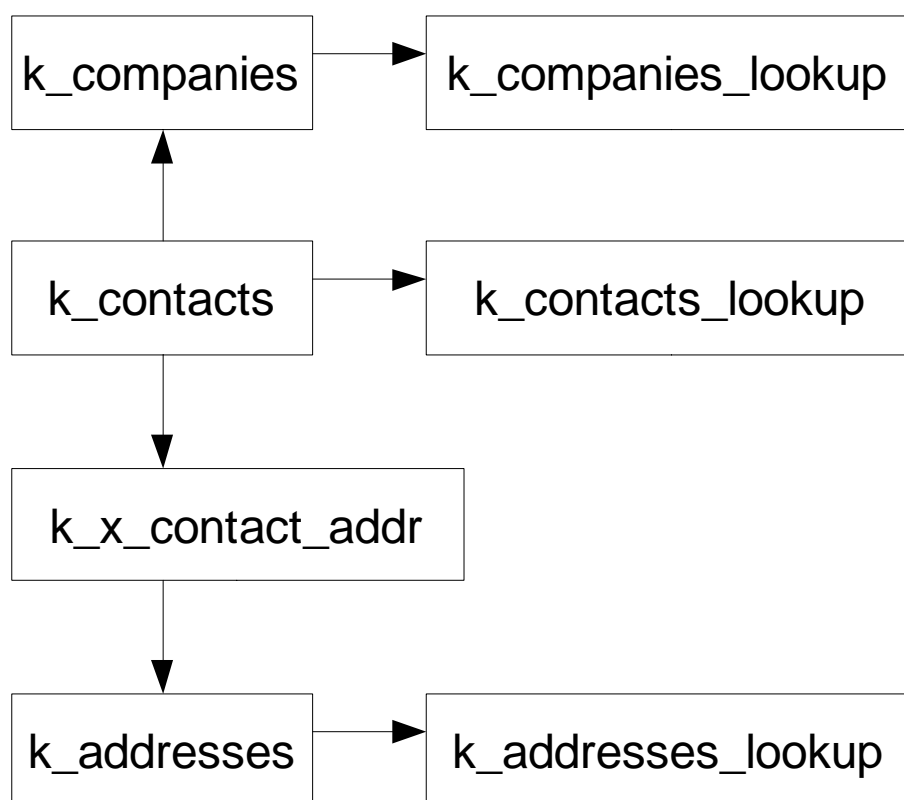
Cuando está activado el modo de inserción o actualización, dos contacts se consideran el mismo si su columna `sn_passport` es la misma.

Para evitar que se dupliquen los contactos, existen dos indicadores que se pueden pasar como parámetro al método `store()`:

- `NO_DUPLICATED_NAMES` : Evita que haya nombres y apellidos duplicados (`tx_name+tx_surname` en `k_contacts`).
- `NO_DUPLICATED_EMAILS` : Evita que haya e-mails duplicados.

Desde el método `perform()` de la clase `ImportExport` la cláusula que hay que emplear para evitar duplicados es `WITHOUT DUPLICATED NAMES` o `WITHOUT DUPLICATED EMAILS`.

Esquema de datos cargados durante la importación de un contacto desde un fichero de texto.



Cómo funciona el cargador de productos

Cargar producto en el modelo de datos de hipergate es una tarea relativamente compleja. Los datos sobre los productos se almacenan en siete tablas.

1. **k_products**: Contiene la información básica sobre el producto: nombre, precio estándar, etc.
2. **k_addresses**: Es un subregistro de k_products. Cada producto tiene una dirección asociada opcional.
3. **k_prod_fares**: Desde la versión 3.0. Cada producto puede tener varias tarifas.
4. **k_prod_locats**: Un producto puede estar presente en varias ubicaciones. Si es un ítem físico puede encontrarse en varios almacenes. Si es un archiv es posible que se pueda descargar de diferentes sitios.
5. **k_prod_attr**: Esta tabla contiene algunos atributos de uso común en los productos: tamaño, peso, etc.
6. **k_prod_keywords**: Palabras clave (para búsquedas). Las palabras clave se almacenan todas en un campo longvarchar, por consiguiente, sólo hay un registro de claves por cada producto.
7. **k_x_cat_objs**: Esta tabla mantiene asociados los productos a su categoría correspondiente.

Potencialmente, cuando se inserta un producto hay que escribir en todas las tablas descritas anteriormente. Esto es lo que hace la clase ProductLoader class del paquete com.knowgate.hipergate.

Restricciones a la carga de productos

Dado que la clase ProductLoader carga los productos desde un archivo de texto delimitado, existen fuertes restricciones a lo que se puede cargar. En particular, sólo es posible cargar una tarifa y una ubicación para cada producto.

Carga de datos desde archivos de texto delimitado (CSV)

La clase `com.knowgate.dataobjs.DBSubset` permite cargar archivos de texto en una tabla mediante el uso combinado de los métodos `parseCSV()` y `store()`.

En el siguiente ejemplo se usan dos páginas web: un HTML estático que hace POST de un archivo de texto y una JSP que recoge el archivo y lo inserta en la tabla `k_companies`.

Archivo de ejemplo 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 es el GUID del Área de Trabajo donde vayan a insertarse los datos.

☞ Los valores de remonte “ACTIVE” y “CLIENTS” deben haberse precargado en la tabla k_companies_lookup antes de cargar el archivo.

Página 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>
```

Página csvload.jsp

```
<%@ page
import="java.util.Enumeration,java.sql.SQLException,com.oreilly.servlet.Mul
tipartRequest,com.knowgate.jdbc.JDBCConnection,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");

    JDBCConnection 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>

```

Ejemplo de carga de texto compleja.

El siguiente código muestra una carga combinada de contactos con su dirección correspondiente desde un archivo de texto delimitado.

El descriptor del archivo se pasa como parámetro en un campo de la página csvpost.html –ver clase `com.knowgate.misc.CSVParser` en el API JavaDoc–.

Las variables `sGuWorkArea` y `sIdUser` de la página csvload.jsp hay que cambiarlas para que coincidan con el Área de Trabajo y Usuario deseados para realizar la carga.

Antes de cargar el archivo `contacts.csv` hay que cargar los remotes de las tablas `k_contacts_lookup` y `k_addresses_lookup` para los campos `de_title` y `tp_street`, `id_state` respectivamente.

En la tabla `k_contacts_lookup` hay que cargar el campo `gu_owner` con el Área de trabajo, el campo `id_section` con el valor 'de_title' y cada campo `vl_lookup` con los valores {'BUSINESS DEVELOPMENT MANGER',

'CIO', 'PRODUCT MANAGER'} que son los valores para los cargos que aparecen en el archivo `contacts.csv`.

En la tabla `k_addresses_lookup` hay que cargar igualmente `gu_owner=sGuWorkArea, id_section='tp_street'` y `vl_lookup = {'CALLE','AVDA.'}` y, por otra parte, `id_section='es'`, `vl_lookup='MAD'`.

Las compañías con razón social: ABC, ACS, AD PEPPER y ADQUIRA deben existir en la tabla `k_companies` previamente. EL método `com.knowgate.crm.Contact.store()` asigna automáticamente si no existe un GUID `gu_company` a cada contacto insertado buscando una compañía cuya razón social coincida con la del contacto insertado.

Archivo de ejemplo `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;;
```

Página `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>
```

Página `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" %>
```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```

<%@ 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;
DBCColumn 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

JDBCConnection 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 = (DBCColumn) 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 = (DBCColumn) 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>

```

Volcado de base de datos a archivos de texto

La clase `com.knowgate.dataobjs.DBSubset` permite volcar directamente registros de una tabla en formato de texto delimitado o XML.

El método `print()` se utiliza para generar volcados a archivos de texto delimitado.

```

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();

```

El método `toXML()` se utiliza para generar cadenas de texto XML.

```

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();
```

Métodos Abreviados de Acceso a Tablas de Remonte

Una parte considerable de los accesos a base de datos se produce contra [tablas de remonte](#) que casi nunca cambian.

Los valores de remonte suelen ser unas pocas decenas o hasta varios centenares de filas que se muestran típicamente en combos o en un pop-up de selección.

La clase `com.knowgate.hipergate.DBLanguages` proporciona algunos métodos útiles para recuperar rápidamente valores de remonte y reducir el volumen de accesos a base de datos.

Seguridad y Roles de Usuarios

Cómo crear un nuevo Dominio

Los dominios se crean desde el método `createDomain()` de la clase `com.knowgate.hipergate.datamodel.ModelManager`.

El proceso de creación de un dominio es bastante largo y delicado. Para simplificarlo, los nuevos dominios se crean haciendo copias clónicas del dominio MODEL que viene precargado en la instalación estándar.

Durante el proceso de clonado se genera automáticamente un nuevo identificador numérico para el dominio recién creado. Asimismo, se crean nuevos [GUIDs](#) para las [Áreas de Trabajo](#), [Usuarios](#) y [Grupos](#).

Los datos a copiar para cada dominio están definidos en los archivos `domain_clon.xml` (uno por cada base de datos) dentro de la carpeta `com/knowgate/hipergate/datamodel/scripts/dbms/`. Estos archivos XML se manejan desde la clase `com/knowgate/datacopy/DataStruct` que es la que los interpreta y ejecuta.

El código de clonado de dominio está ubicado en el script BeanShell `domain_create.js` en `com/knowgate/hipergate/datamodel/scripts/`. Al estar externalizado, es posible modificar el procedimiento de creación de nuevos dominios sin tener que recompilar todos los fuentes.

Los dominios se pueden crear **desde código Java** mediante:

```
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 */ }
```

O desde la **línea de comandos** :

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

Ver: [Scripts Java Bean Shell](#)

Cómo eliminar un Dominio

La eliminación de un Dominio es un proceso en cascada que borra todos los datos asociados a dicho dominio.

Los dominios se pueden eliminar **desde código Java** mediante:

```
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 */ }
```

O desde la **línea de comandos** :

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

Cómo crear una nueva Área de Trabajo Vacía

Las Áreas de Trabajo se crean desde la clase
`com.knowgate.workareas.WorkArea`.

Para crear un Área de Trabajo es necesario :

1. Insertar un registro en `k_workareas` asociado a un Dominio.
2. Crear los directorios del Área de Trabajo.
3. Asignar los permisos por aplicación y grupo de usuarios.

☞ Este procedimiento de creación no inserta ningún dato en las tablas de remonte del Área de Trabajo.

```
import java.sql.Statement;  
  
import com.knowgate.workareas.*;  
  
import com.knowgate.acl.ACLDomain;  
import com.knowgate.jdc.JDCCConnection;  
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  
JDCCConnection 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");
```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```

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 workareasput 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");

```

 Cuando se Crea un Área de Trabajo hay que crear sus directorios de trabajo .

Cómo duplicar un Área de Trabajo

En muchos casos, más que crear un Área de Trabajo Vacía, interesa crear un Área de Trabajo que esté cargada desde el primer momento con archivos de ejemplo, valores en las tablas de remonte y datos de prueba.

Desde la **línea de comandos** :

```
java com.knowgate.hipergate.datamodel.ModelManager
hipergate.cnf clone workarea dominio_origen.workarea_origen
dominio_destino.workarea_destino [verbose]
```

Cómo eliminar un Área de Trabajo

Desde la **código Java** :

```
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
JDBCConnection oCon = oDBB.getConnection("delete_workarea");

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

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

O desde la **línea de comandos** :

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

Cómo crear un Usuario

Para crear un usuario es necesario:

1. Insertar un registro en la tabla `k_users` con los campos `id_domain` y `gu_workarea`.
2. Adscribir el usuario a los grupos de permisos pertinentes.
3. Crear la Categoría *Home* para el Usuario.
4. Asociar la Categoría Home con el Usuario.
5. Asignar permisos para el usuario sobre la su Categoría Home.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

6. Asignar permisos a los administradores del dominio sobre la Categoría Home del nuevo usuario.
7. (Opcional) Añadir el GUID del usuario a las tablas k_bugs_lookup y k_duties_lookup para que pueda ser asignado a incidencias y tareas. Existe un ejemplo de cómo hacer esto en la página /vdisk/usernew_store.jsp.

```
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 oUstr = new ACLUser();

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

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

oUstr.put(DB.id_domain, 1027);
oUstr.put(DB.gu_workarea, "012345678901234567890123456789AB");
oUstr.put(DB.tx_nickname, "usernicks");
oUstr.put(DB.tx_pwd, "donttell");
oUstr.put(DB.tx_main_email, "usernicks");

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

String sUstrGUID = oUstr.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, sUstrGUID);

// *****

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

```

// 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");

```

Script Alternativo de Creación de Usuario con Categorías por defecto :

```

import bsh.*;

import java.sql.*;

import com.knowgate.acl.*;
import com.knowgate.dataobjs.DB;
import com.knowgate.dataobjs.DBBind;
import com.knowgate.jdc.JDCCConnection;
import com.knowgate.hipergate.datamodel.ModelManager;

final int iDomainId = 1027;
final String sWorkAreaId = "012345678901234567890123456789AB";

```

```

Integer iOne = new Integer (1);

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

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

// Get Connection from Pool
JDBCConnection oCon = oDBB.getConnection("create_user2");

// Create User
String sUsrGUID = ACLUser.create (oCon, new Object[] { new
Integer(iDomainId), "username", "userpwd", "username", 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");

```

Consultas por Formulario (QBF)

Las consultas por formulario (o *Query By Form*) son una forma de guiar al usuario para componer consultas sencillas contra una única tabla o vista.

En un QBF se pide al usuario que proporcione los valores para algunos campos de la tabla o vista y luego se buscan los registros cuyos campos cumplen con las condiciones especificadas.

En el modelo estándar los QBFs admiten 3 campos de búsqueda simultáneos conectados por operadores lógicos (OR, AND).

La definición del QBF se guarda en archivos XML en el directorio /storage/qbf.

Cada Consulta generada (una instancia para unos parámetros concretos de una definición de QBF) se guarda en la tabla k_queries.

Ejemplo de archivo de definición de QBF:

```
<?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&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>
    </field>
  </fields>
</qbf>
```



```

        <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>

```

Cómo crear una definición de consulta paso a paso.

- 1º) Crear un nuevo documento XML vacío con el tag <qbf>. Establecer el juego de caracteres para que el parser de XML los reconozca.
- 2º) Elegir un título en los idiomas que se vayan a soportar en el cliente: es, en, fr, de, etc. Añadir los títulos en los tags <title_xx>.
- 3º) Elegir un método de paso de parámetros a la página de visualización HTML: "post" o "get".
- 4º) Establecer la URL de la página de visualización de resultado en HTML <action>. Los caracteres ampersand (&) de la URL deben escaparse como entidades &. La página <action> es sólo para la salida de resultados online en HTML. Dentro de la suite, la página /common/qbf.jsp puede invocar también al servlet com.knowgate.http.HttpQueryServlet para enviar al cliente resultados en formato Excel o texto delimitado.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

5º) Establecer un objeto base y su alias. Debe ser una única tabla o vista del modelo de datos.

6º) Establecer un filtro base. Los filtros base sirven para separar los datos de un Área de Trabajo del resto. Cuando el usuario lanza una instancia parametrizada de una especificación de consulta, el filtro base se añade automáticamente al SQL generado antes de ejecutarlo. El filtro base contiene dos tipos de sentencias comodín `${cookie.nombre}` y `${param.nombre}` que se sustituyen en tiempo de ejecución por los valores de las cookies y los parámetros de la petición HTTP (`HttpServletRequest`).

7º) Definir los campos por los que se podrá filtrar. Para cada campo especificar:

7.1) Nombre real del campo en la base de datos.

7.2) Etiqueta traducida en cada idioma soportado.

7.3) Tipo { `varchar` | `smallint` | `integer` | `flota` | `date` | `lookup` }

7.4) Sólo para los tipos `lookup`

7.4.1) URL de llamada al formulario de selección de valores de remonte.

8º) Definir las columnas visualizables como salida y si estarán seleccionadas por defecto o no. La elección de columnas de salida sólo afecta a la generación de ficheros de texto delimitado y hojas Excel, no a las consultas con salida HTML cuya estructura de columnas viene fijada por la página JSP de visualización.

9º) Especificar qué columnas admiten ordenación. La consulta puede ordenar por una de ellas.

10º) Grabar el archivo producido en `/storage/qbf`.

11º) Las instancias parametrizadas de la consulta pueden generarse ahora con la página `/common/qbf.jsp`.

12º) La consulta puede ejecutarse directamente con la clase

`com.knowgate.hipergate.QueryByForm` la clase

`com.knowgate.http.HttpQueryServlet` es un ejemplo de cómo hacerlo.

Acceso a archivos

La clase `com.knowgate.dfs.FileSystem` proporciona un mecanismo unificado para la copia y borrado de archivos y directorios locales y remotos.

Como una de las diferencias principales entre la rama `/storage` y la rama `/web` de archivos (ver Manual de Instalación) se asume que los archivos

de /web deben ser locales al servidor /web, mientras que los archivos de /storage pueden accederse también a través de FTP.

☞ El acceso a archivos via FTP a través de la clase `com.knowgate.dfs.FileSystem` está en fase alfa de desarrollo en la versión 1.0 de hipergate y no debe ser utilizada en entornos de producción.

Leer archivos de texto en un solo paso

A veces es útil leer un archivo de texto plano directamente a un array de caracteres.

Es posible usar `com.knowgate.dfs.FileSystem.readfile()` para este propósito.

Convertir archivos de texto ASCII a Unicode

La clase `com.knowgate.dfs.FileSystem` contiene el método `convert()` para pasar archivos ASC-II a Unicode o viceversa.

Transformaciones XSLT

El sistema de producción de contenidos funciona aplicando una hoja de estilo XSL a un archivo de datos en formato XML.

El modelo está basado en 3 elementos clave:

1. La plantilla (template) u hoja de estilo XSL.
2. La definición de la información que admite la plantilla (metadatos)
3. Los datos en si mismos.

Tanto los metadatos como los datos son ambos documentos XML.

Introduciremos ahora algunos conceptos relativos al modelo.

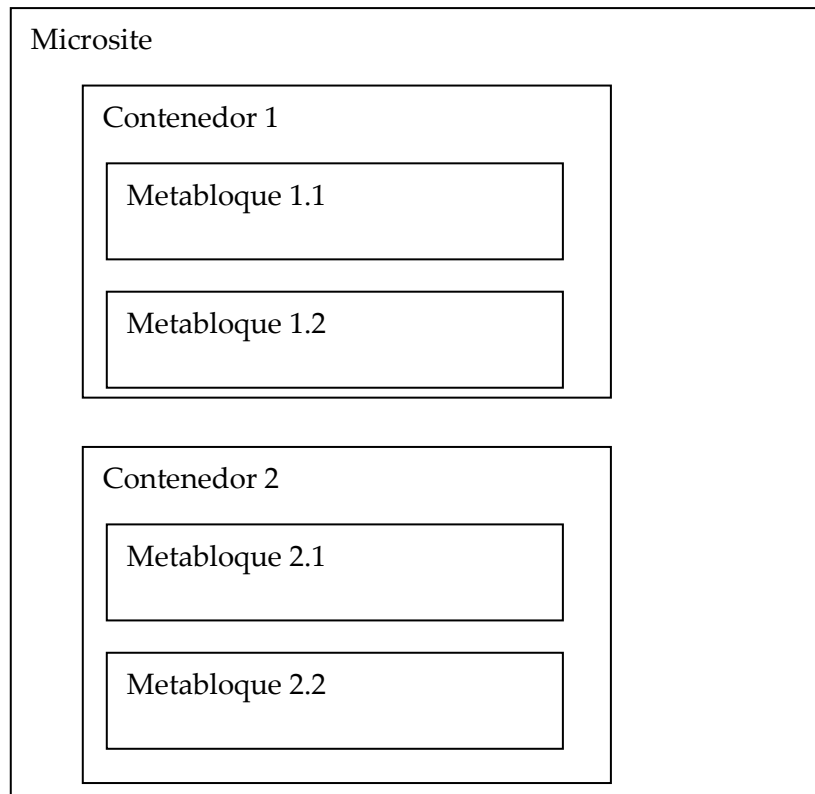
Metadatos

Microsite: Metadatos que definen la estructura interna de un conjunto de páginas. Los Microsites tienen *Contenedores*.

Contenedor: Se corresponde (típicamente) con la definición estructural de una página HTML.

Metabloque: Cada uno de los tipos de bloque que es lícito colocar dentro de un contenedor.

☞ El microsite se define y almacena en un fichero XML que debe cumplir con el schema microsite.xsd



Datos

PageSet: Es una instancia concreta de un Microsite.

Page: Es una instancia de un objeto de tipo contenedor.

Block: Es una instancia de un objeto de tipo metabloque.

☞ Los Pagesets se definen y almacenan en un ficheros XML que debe cumplir con el schema pageset.xsd

☞ No es necesario definir metadatos para los párrafos ni las imágenes ya que al ser todos iguales su estructura interna van cableados dentro del código.

Ejecutor de tareas sencillo

El ejecutor de tareas (jobs) sencillo es una clase Java única (`com.knowgate.scheduler.SingleThreadExecutor`) pensada para procesar de forma secuencial los átomos de una tarea.

El ejecutor es mono-thread y utiliza la base de datos como soporte de progreso de operaciones. Se trata de un código pequeño, sencillo, robusto y fácil de depurar, aunque no es muy escalable.

`SingleThreadExecutor` es ideal para procesar pequeños volúmenes de información, por ejemplo, para enviar entre 100 y 1000 correos o publicar unas pocas decenas de archivos por FTP.

Ejecución mono-thread por línea de comandos

Es posible arrancar un proceso mono-thread para procesar un Job particular por línea de comandos.

Pueden darse dos circunstancias diferentes :

- a) Que el Job esté creado previamente en la tabla `k_jobs`.
- b) Que haya que crear el Job a partir de un archivo de definición XML.

Planificador de Tareas

El planificador de tareas se encuentra en el paquete `com.knowgate.scheduler`. El planificador tiene una estructura mucho más compleja que el ejecutor simple. En el ejecutor simple se utiliza la base de datos como soporte para llevar el progreso de la tarea y, además, sólo existe un thread ejecutor. Por el contrario, el planificador utiliza una cola en RAM de la que obtiene átomos un pool de threads ejecutores. La base de datos se actualiza por lotes de trabajo y hay varios procesos ejecutores con lo cual se gana enormemente en eficiencia, aunque también en complejidad.

El planificador de se compone de los siguientes elementos:

1º) Tareas : (Jobs) Se crean y almacenan en la tabla [k_jobs](#).

2º) Comandos : (Commands) Cada tarea tiene asignado un comando (listados en la tabla [k_lu_job_commands](#)). El comando le dice a la tarea qué es lo que debe hacer con los Átomos que la componen. La implementación para los comandos estándar está en el paquete `com.knowgate.scheduler.jobs`. Cada comando se implementa como una subclase de la clase abstracta `com.knowgate.scheduler.job`.

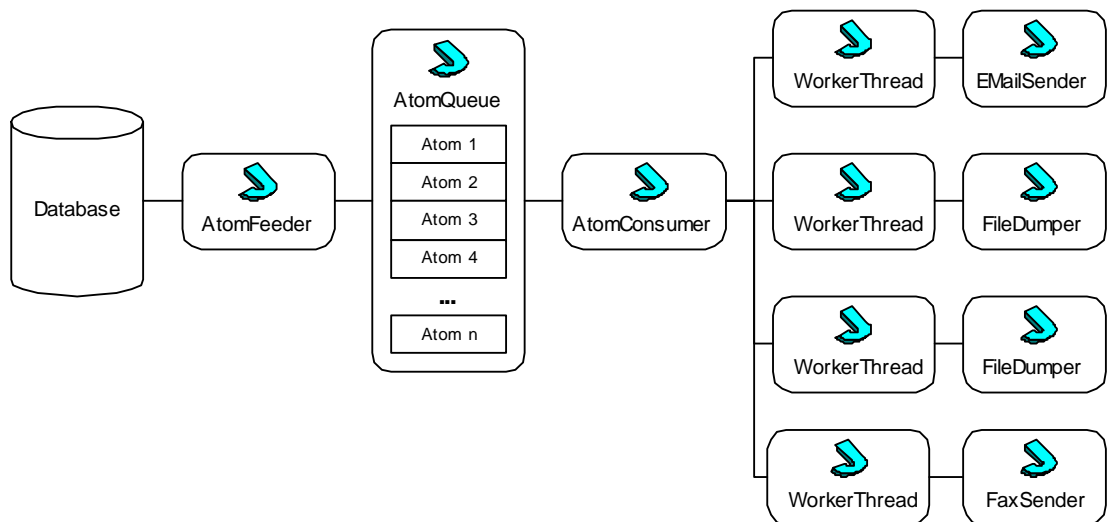
3º) Átomos : (Atoms) Cada tarea se divide en unidades atómicas de proceso llamadas Átomos a los que se les aplica el comando asignado a la tarea. Los átomos pendientes de ejecutar se almacenan en la tabla [k_job_atoms](#). Los átomos ya ejecutados se almacenan en la tabla `k_job_atoms_archived`. Los átomos son la unidad transaccional de la tarea. Cada Átomo se materializa en memoria RAM como un objeto Java obtenido de la tabla `k_job_atoms` antes de ejecutarse y luego vuelve a la tabla `k_job_atoms_archived` una vez que ha sido ejecutado.

4º) Cola de Ejecución : (AtomQueue) La cola de ejecución es una clase Java que mantiene en memoria un número limitado de átomos pendientes de ejecutar. La cola existe por cuestiones de eficiencia computacional: es más rápido recoger los Átomos por lotes que lanzar una consulta SQL por cada uno de ellos, y, por otra parte, no es posible cargar todos los Átomos pendientes de ejecutar en RAM, pues su número puede ser arbitrariamente grande.

5º) Alimentador de la Cola : (AtomFeeder) Un objeto Java específico para alimentar la cola de átomos pendientes de ejecución.

6º) Consumidor de la Cola : (AtomConsumer) Un objeto Java específico para sacar los átomos de la cola de uno en uno en un entorno multi-thread.

7º) Threads Ejecutores : (WorkerThread) Un pool de threads especialmente diseñados para ejecutar comandos de las tareas. Cada thread ejecutar saca el primer átomo disponible en la cola a través del consumidor, inspecciona el comando a ejecutar para dicho átomo según la tarea a la que pertenezca e instancia la subclase de comando pertinente.



Ejecución por línea de comandos

El planificador de tareas puede arrancarse por línea de comando mediante: `com.knowgate.scheduler.SchedulerDaemon ruta_cnf [verbose]`

Donde *ruta_cnf* es una ruta absoluta al archivo de propiedades de inicialización, por ejemplo `/etc/hipergate.cnf`

Si se especifica el parámetro *verbose*, el progreso de cada thread en ejecución se mostrará por la consola del sistema.

Al arrancar, el demonio llevará a cabo las siguientes acciones.

- 1ª) Instanciar una Cola de Ejecución en RAM junto con su correspondiente Alimentador y Consumidor.
- 2ª) Conectarse con la base de datos especificada en la propiedad `dburl` del archivo de propiedades de inicialización.
- 3ª) Crear un pool de thread ejecutores.
- 4ª) Buscar en la tabla `k_jobs` tareas pendientes de empezar a ejecutarse.
- 5ª) Cargar por lotes los átomos de las tareas pendientes en la cola.
- 6ª) Arrancar todos los threads ejecutores del pool para que consuman átomos de la cola.

Callbacks

Los thread ejecutores pueden proporcionar información de progreso mediante callbacks. Para ello hay que escribir una subclase de la clase `WorkerThreadCallback` e implementar el método abstracto `call()`.

Las operaciones que actualmente son comunicadas por los threads ejecutores mediante callbacks son:

Código de Operación	Descripción
WT_EXCEPTION	Se produjo una excepción dentro del thread ejecutor.
WT_JOB_INSTANTIATE	Se creó una instancia de una subclase de Job.
WT_JOB_FINISH	Se finalizó la ejecución de un Job completo.
WT_ATOMCONSUMER_NOMORE	No existen más átomos para consumir en la cola.
WT_ATOM_GET	Se obtuvo un Átomo de la cola.
WT_ATOM_CONSUME	Se consumió un Átomo previamente obtenido de la cola.

Ejemplo de una subclase de callback tomado del proyecto JobController:

```
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");
    }
}
```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```

        iWritten += sMessage.length() + 1;
    } // call
} // ThreadNotify

```

Creación y registro de ThreadNotify como 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();

```

Configuración del número de threads ejecutores

El número máximo de threads ejecutores se configura mediante la propiedad `maxschedulerthreads` del archivo `hipergate.cnf`.

Archivo de Log

Para cada Tarea se crea un archivo de log en
`/storage/jobs/guid_workarea/guid_job.txt`

Subclases de Job

La versión estándar trae 4 subclases de Job :

DummyJob : Una versión hueca de Job que no hace nada, sólo para propósitos de testeo.

FileDumper : Es la versión más sencilla de Job. Pensada sobre todo como ejemplo. Toma los archivos de entrada, reemplaza los tags de personalización para cada miembro de la lista de distribución y graba los archivos resultantes en el directorio de log del Job
`(/storage/jobs/guid_workarea/guid_job/)`.

MimeSender : Envía páginas HTML personalizadas por e-mail.

FTPPublisher : Envía archivos por FTP a un servidor remoto.

Para cada subclase de Job existe un comando asociado en la tabla `k_lu_job_commands` según la siguiente correspondencia :

class name	id_command
<code>com.knowgate.scheduler.jobs.DummyJob</code>	DUMY
<code>com.knowgate.scheduler.jobs.FileDumper</code>	SAVE
<code>com.knowgate.scheduler.jobs.MimeSender</code>	SEND
<code>com.knowgate.scheduler.jobs.FTPPublisher</code>	FTP

Propiedades de entorno para las subclases de Job

Los Jobs pueden leer y utilizar propiedades de entorno del archivo `hipergate.cnf`, o de cualquier otro colección de propiedades especificada cuando se llama al método `Job.instantiate()`.

Propiedades de `hipergate.cnf` de uso típico en los Jobs

<code>driver</code>	Driver JDBC
<code>dburl</code>	URL a la base de datos
<code>dbuser</code>	Usuario para conectarse a la bbdd
<code>dbpwd</code>	Clave del usuario
<code>workareasput</code>	Ruta de directorios para recuperar archivos generados previamente desde el webbuilder.
<code>maxschedulerthreads</code>	Número máximo de threads ejecutores
<code>mail.transport.protocol</code>	Protocolo de transporte de correo
<code>mail.host</code>	Host para el envío de correos
<code>mail.user</code>	Usuario para conectarse al host de correo

Parámetros para cada subclase de Job

Cada subclase de Job puede tener sus propios parámetros adicionales en el campo `k_jobs.tx_parameters`. Los parámetros se almacenan separados por comas en formato "nombre:valor,nombre:valor,nombre:valor".

Los parámetros utilizados son:

bo_attachimages	Puede ser "1" o "0". Si es "1" indica que las imágenes deben ir adjuntas con el mensaje enviado, es decir, hay que convertir los en .
bo_path	Puede ser "1" o "0". Si es "1" indica que hay que crear las rutas de directorios que no existan en destino al copiar archivos por FTP.
gu_list	GUID de la lista de distribución donde enviar el mensaje.
gu_pageset	GUID del PageSet a enviar por e-mail o fax.
gu_workarea	GUID del Área de Trabajo a la que pertenece el PageSet a enviar por e-mail o fax.
nm_file	Nombre del archivo a copiar al host remoto.
nm_pageset	Nombre del PageSet a enviar por e-mail o fax.
nm_server	Nombre del host remoto.
path	Ruta de directorios en el host remoto donde copiar los archivos.
tx_from	e-mail del remitente del mensaje.
tx_nickname	Nombre de usuario utilizado para conectarse por FTP al host remoto.
tx_pwd	Clave de usuario utilizado para conectarse por FTP al host remoto.
tx_sender	Nombre completo del remitente del mensaje.
tx_subject	Asunto del mensaje.

Envío de e-mails desde la línea de comandos

Hipergate proporciona dos clases para el envío de e-mails desde la línea de comandos:

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```
com.knowgate.hipergate.SendMail  
com.knowgate.scheduler.jobs.MimeSender
```

SendMail es una clase apropiada para enviar unos pocos mensajes a direcciones de correo almacenados en un fichero de texto plano externo a la base de datos de hipergate. La clase SendMail no permite personalizar los mensajes enviados ni insertar Web Beacons.

MimeSender es una clase más compleja que SendMail utilizada para enviar mensajes por lotes a miembros de una lista de distribución almacenada en la base de datos de hipergate. La clase MimeSender permite personalizar los mensajes enviados a cada destinatario e insertar Web Beacons en ellos para obtener información acerca de los correos leídos.

A continuación se describe el funcionamiento de ambas clases.

Envío de e-mails a direcciones externas mediante SendMail

La clase `com.knowgate.hipermail.SendMail` puede ser utilizada para enviar e-mailings masivos desde la línea de comandos, con o sin la creación de un lote de proceso.

Esta clase coge como parámetro la ruta absoluta a un fichero de propiedades (.cnf) que contiene toda la información necesaria para realizar el e-mailing.

La sintaxis de la línea de comandos es pues :

```
#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
```

o en 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
```

Si los e-mails se envían mediante un Job hay que especificar un Segundo parámetro que apunte a archivo .cnf de conexión a la base de datos:

```
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
```

El archivo sendmail.cnf debe contener las siguientes propiedades:

Propiedad	Descripción
mail.transport.protocol	smtp
mail.user	Usuario para el servicio SMTP
mail.password	Contraseña para el servicio SMTP
mail.smtp.host	Host de envío de correo
mail.smtp.socketFactory.class	Sólo para envíos a través de SSL. Establecer esta propiedad en javax.net.ssl.SSLSocketFactory
mail.smtp.socketFactory.port	Sólo para envíos a través de SSL.
recipients	Ruta absoluta completa al archivo que contiene las direcciones de correo de los destinatarios a razón de una dirección por línea.
encoding	Codificación de caracteres para los correos. Ej. ISO-8859-1 o UTF-8
userdir	Ruta completa al directorio que contiene los archivos que compondrán el mensaje
textplain	(Opcional) Nombre del archivo que contiene la parte de texto simple del mensaje. Este archivo debe estar en el directorio userdir
texthtml	(Opcional) Nombre del archivo que contiene la parte de texto HTML del mensaje. Este archivo debe estar en el directorio userdir
subject	Asunto
from	Dirección de e-mail del remitente
displayname	Nombre mostrado del remitente
replyto	(Opcional) Dirección de respuesta
recipienttype	Tipo de destinatario: to, cc o bcc
attachments	(Opcional) Nombres de los archivos adjuntos separados por punto y coma.
job	(Opcional) Nombre del Job

	utilizado para enviar los correos. Si se utiliza esta propiedad, en la línea de comandos es obligatorio indicar como segundo parámetro la ruta al fichero .cnf de conexión a la base de datos.
messageid	(Opcional) Identificador único mensaje

Jobs en la clase SendMail

Si se especifica un nombre de job en el primer fichero de propiedades pasado como parámetro a la clase SendMail, entonces se creará un nuevo lote de trabajo en la tabla k_jobs con sus correspondientes átomos en la tabla k_job_atoms a razón de un átomo por e-mail a ser enviado.

Los procesos de envío masivo de correos por lotes se pueden re-lanzar en caso de que fallen o se corten en mitad de su ejecución. En tal caso el proceso continuará ejecutándose a partir del último e-mail que se envió con éxito. El lote se identifica a través del nombre que se le da en la propiedad job del fichero .cnf y que es almacenado en la columna tl_job de la tabla k_jobs.

Envío de e-mails mediante MimeSender

hipergate permite enviar e-mails personalizados desde la línea de comandos mediante la clase

`com.knowgate.scheduler.jobs.MimeSender` class.

La utilidad de línea de comandos crea un lote de proceso sobre la marcha y empieza a enviar con él los correos inmediatamente.

Para enviar correos con MimeSender hay que tener previamente :

1. Una cuenta de correo creada en el modulo de hypermail.
2. Una lista de destinatarios en el gestor de contactos.
3. Una plantilla para el cuerpo del mensaje (bien texto plano o HTML) con sus archivos adjuntos.
4. Un archivo de propiedades .cnf que contenga la información necesaria para conectarse a la base de datos y recuperar la lista de destinatarios.

Ejemplo de archivo de propiedades para MimeSender

```
# Archivo de configuración de hipergate MimeSender

# Base de datos
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

# Sistema de Archivos
fileserv=localhost
fileprotocol=file\://
fileuser=
temp=C:\\Windows\\Temp

# Las plantillas del e-mail deben estar en el subdirectorio
# \mailing\nombre.de.la.lista del directorio storage
storage=C:\\ARCHIV~1\\Tomcat\\storage

# Host SMTP

# mail.account debe ser el mismo valor que tenga la columna
# k_user_mail.tl_account
mail.account=account_name

# mail.list debe ser el mismo valor que tenga la columna
# k_lists.de_list
mail.list=Descripcion_Lista

mail.job.title=Test eMailing 2009-01-01
```

Este archivo es similar al hipergate.cnf estándar pero tiene menos propiedades y la sección adicional # Mail System.

El fichero de propiedades puede tener cualquier nombre, pero debe estar ubicado en el mismo directorio donde esté hipergate.cnf. Por defecto, este directorio es /etc para Linux y C:\\Windows para Windows.

La propiedad mail.account debe ser el nombre de una cuenta en el webmail (columna tl_account en la tabla k_user_mail). Para crear una nueva cuenta en la aplicación ir a Herramientas Colaborativas – WebMail y hacer click en el enlace de crear nueva cuenta.

La propiedad mail.list debe ser la descripción de una lista de destinatarios (columna de_list en la tabla k_lists).

La propiedad mail.job.title debe ser un nombre único cualquiera que se asigne al lote de trabajo (columna t1_job en la tabla k_jobs).

Plantilla de contenidos para los e-mails

La plantilla para el cuerpo del e-mail se debe poner en el directorio storage/mailling/*descripción_de_la_lista*. Para el ejemplo anterior, el directorio sería concretamente:

C:\ARCHIV~1\Tomcat\storage\mailling\Descripcion_Lista

El e-mail puede ser texto simple o HTML. Si es HTML se agregará una parte adicional de texto simple al mensaje extraído automáticamente eliminando los tags de formato del HTML.

Para enviar e-mails de texto simple colocar un archive llamado body.txt en el directorio storage/mailling/ *descripción_de_la_lista* y para enviar HTML colocar un archive llamado body.htm en el mismo directorio.

Este es un HTML de ejemplo para un 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\mailling\Descripcion_Lista\hipergate190x50.gif"
WIDTH="190" HEIGHT="50" BORDER="0" ALT="hipergate logo" />
    <BR/>
    Hello World!
    <BR/>
    Este mensaje es para {#Datos.Nombre} {#Datos.Apellidos}

    <!--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>
```

Todas las imágenes referenciadas en el HTML serán adjuntadas al mensaje como adjuntos.

Personalización de los e-mails

Los mails se pueden personalizar añadiendo algunas [marcas predefinidas](#) que se reemplazan por los valores de determinadas columnas de la base

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

de datos. En el ejemplo {#Datos.Nombre} hace referencia a la columna tx_name de la tabla k_contacts y {#Datos.Apellidos} hace referencia a la columna tx_surname de la misma tabla.

Seguimiento de mails leídos con Web Beacons

Hipergate procesa un comentario especial <!--WEBBEACON SRC="..."--> que es reemplazado por una imagen oculta de 1x1 pixel con el mismo SRC. Este comentario debe insertarse justo antes del tag </BODY> de HTML.

hipergate dispone tambien de un manejador de Web Beacons por defecto en la página /hipermail/web_beacon.jsp Esta página devuelve una imagen en formato GIF transparente de 1x1 pixel y escribe un registro en la tabla k_job_atoms_tracking.

El Web beacon debe recibir como parámetros el GUID del lote que envié el e-mail, el número de átomo, el GUID de la Compañía y/o Contacto y la dirección e-mail del destinatario. Esto se consigue insertando los siguiente parámetros en el query string de la URL del Web Beacon.

gu_job={#Job.Guid}	GUID del Lote
pg_atom={#Job.Atom}	Número de Átomo
gu_company={#Data.Company_Guid}	GUID de la Compañía
gu_contact={#Data.Contact_Guid}	GUID del Contacto
tx_email={#Address.EMail}	e-Mail del destinatario

En tiempo de ejecución cada marca {#...} es reemplazada por el valor adecuado justo antes de enviar cada e-mail.

Línea de comandos MimeSender

Las librerías requeridas para ejecutar MimeSender son:

- The JDBC driver for your database
- Sun JavaMail
- Apache Commons HttpClient
- Jakarta ORO
- HtmlParser

Por tanto una línea de comandos tiene el siguiente aspecto:

```
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
```

Cómo funciona el proceso de mailing

Los pasos seguidos por el método MimeSender.main() son:

1. Buscar en la tabla `k_jobs` un lote cuyo título sea igual que la propiedad `mail.job.title` del archivo `.cnf` especificado en la línea de comandos.
 - 1.1. Si no se encuentra ningún lote con el título especificado, se crea uno nuevo en estado `running` (3) dicho estado inicial impide que otros hilos de ejecución del Job Scheduler arranquen el mismo lote.
 - 1.2. Si se trata de un lote nuevo, se resuelven todas las direcciones de e-mail incluídas en la lista cuya descripción coincida con la propiedad `mail.list property` del archivo `.cnf` y se crea un átomo en la tabla `k_job_atoms` para cada dirección de correo de la lista copiando los valores de las columnas de `k_member_address` en la tabla `k_job_atoms`. Cada átomo es creado en estado `running` (3).
2. Se abre una session SMTP según los parámetros de la tabla `k_user_mail` que correspondan a la cuenta `mail.account` del archive `.cnf` de propiedades.
3. Se itera por los átomos.
 - 3.1. Para cada átomo (destinatario de e-mail) en el cuerpo del mensaje se reemplazan las marcas `{#...} tags` por los valores de Nombre, Apellidos, etc. presentes en la tabla `k_job_atoms`.
 - 3.2. Se resuelven y se adjuntan al mensaje todas las referencias a imágenes en el HTML.
 - 3.3. Si el mensaje es HTML, se crea una parte alternative de texto plano extraído automáticamente del HTML.
 - 3.4. Se reemplazan los comentarios `<!--WEBBEACON SRC="..."-->` por imágenes transparentes de 1x1 pixel.
 - 3.5. Se envían los mensajes personalizados mediante JavaMail.
 - 3.5.1. Si el e-mail se envió con éxito, mover el átomo de la tabla `k_job_atoms` a `k_job_atoms_archived`.
 - 3.5.2. Si se produjo algún error enviando el e-mail, poner el estado del átomo en `interrupted` (4) y escribir una

descripción breve del error en la columna `tx_log` de la tabla `k_job_atoms`.

3.6. Escribir información de progreso en `system stdout`.

4. Cuando no haya más átomos pendientes de enviar, poner el estado del lote en `finished (0)`.

Logs de los lotes

Los logs de ejecución de los lotes se escriben en el archivo `storage/jobs/guid_del_job.txt`.

Cómo escribir rutinas propias de envío de mails

En ocasiones puede resultar útil escribir rutinas propietarias para enviar documentos basadas en código hipergate.

Para enviar mensajes personalizados es necesario:

1º) Disponer de la lista de destinatarios en algún formato procesable: una tabla de base de datos o un fichero de texto.

2º) Decidir el formato del mensaje: HTML, texto, fax, etc.

3º) Decidir qué partes del mensaje irán personalizadas: asunto, texto interno, etc. Para cada parte personalizable definir los tags que se utilizarán para ser reemplazados por los datos de cada destinatario.

4º) Si hay imágenes o archivos adjuntos, decidir cómo se incluirán en el mensaje.

5º) Decidir qué programa de envío masivo se utilizará.

Obtención de listas de destinatarios

La forma más sencilla de obtener una lista de destinatarios extraída de hipergate es utilizar el interfaz de consultas basado en web y exportar el resultado a Excel o a texto delimitado por comas.

Si la extracción de datos se hace por código Java, lo más práctico es utilizar el método `print()` de la clase `com.knowgate.dataobjs.DBSubset`. `print()` utiliza los delimitadores establecidos en los métodos

`setColumnDelimiter()` y `setRowDelimiter()` y vuelca todas las filas de un `DBSubset` a un `Stream`.

Si la lista de destinatarios está en un archivo de texto en vez de en la base de datos, es posible utilizar la clase `com.knowgate.misc.CSVParser` para leer la lista en un array bidimensional. `CSVParser` admite archivos de texto delimitado con una fila por línea. Los campos pueden ir entrecomillados o no. El delimitador de columna puede ser coma, punto y coma o tabulador.

Personalización de los mensajes

El proceso de personalización debe ser lo bastante rápido y eficiente como para permitir enviar un gran volumen de mensajes con el menor consumo de memoria y CPU.

La clase utilizada para esto en `hipergate` es `com.knowgate.dataxslt.FastStreamReplacer`. `FastStreamReplacer` coge como entrada un archivo de texto (como `InputStream`) y un mapeo de valores (como `HashMap`). En un proceso de una sola pasada se buscan en el archivo de entrada todas las claves del mapeo de la forma “`{#nombre}`” y se reemplazan por su valor correspondiente. No se admiten expresiones regulares ni caracteres comodín. La salida se vuelca a un `String` en memoria.

`hipergate` utiliza un sistema de marcas arbitrario para la personalización de los mensajes. Cada marca se corresponde con un campo en la base de datos que es puesto en el lugar de la marca justo antes de enviar el mensaje. Las marcas pueden estar en inglés o en español indistintamente. Es la clase `com.knowgate.scheduler.Atom` quien tiene cableada dentro del código la tabla de marcas. Durante el proceso de personalización y envío, una subclase de `Job` obtiene el mapeo de `Atom` y lo pasa como parámetro a `FastStreamReplacer` para reemplazar las marcas por valores antes de enviar cada mensaje.

Marcas estándar cableadas dentro de la clase Atom.

Campo Base de Datos	Marca Inglés	Marca Español
no es un campo de bbdd, se reemplaza por la fecha actual 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.Lineal
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

Ejemplo de uso de 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");
```

Imágenes adjuntas a los mensajes

Los e-mails enviados con imágenes incluídas requieren que dichas imágenes se adjunten con el cuerpo del mensaje. En HTML se utiliza el tag para indicar una referencia a una imagen adjunta. Esto implica que antes de enviar un mensaje HTML hay que cambiar todas sus referencias http:// o file:// por cid:

La clase `com.knowgate.scheduler.jobs.EmailSender` usa el parser `org.acmsl.htmlparser.html.HTMLProcessor` para encontrar todos los tags y reemplazar el SRC por una referencia cid: que luego se convierte en un archivo adjunto al mensaje.

Ejemplo de una clase basada en SendMail para enviar correos

```
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(0,1)}, null, null, null);

        System.out.println(oMails.getField(3,1)+" OK");

    } catch (Exception e) {
        System.out.println("ERROR "+oMails.getField(0,1)+"
"+e.getClass().getName()+" "+e.getMessage());
    }
    } // fi
} // next
}
} // SMail

```

Cómo usar el carro de compra

El carro de compra es la clase java
com.knowgate.hipergate.ShoppingBasket

El carro de la compra no se usa desde el back-end de hipergate, está pensado para utilizarse en un site de compra genérico.

El objeto carro de la compra tiene 3 elementos básicos:

1. Identificador del comprador
2. Propiedades globales
3. Líneas de pedido

Las propiedades globales son un conjunto de objetos contenidos en el carro y referenciados por nombre.

Cada línea de pedido tiene, a su vez, su propio conjunto de objetos, también referenciados por nombre.

Lo habitual es que los objetos para cada línea sean los mismos con el mismo nombre: número de línea, cantidad, precio unitario, subtotal, etc. No existe ningún convenio fijo sobre cómo deben llamarse los atributos de las líneas.

Cómo cargar el carro desde JavaScript

La forma más fácil de cargar el carro desde JavaScript es componer una cadena en un campo oculto donde figuren todos los atributos de cada línea separados por comas (u otro delimitador) y luego usar el método:
ShoppingBasket.addLine(String, String)
para cargar cada línea.

Es decir, una página del lado cliente con:

```
<FORM>
<INPUT TYPE="hidden" NAME = "id_comprador" VALUE="12345">
<INPUT TYPE="hidden" NAME = "tipo_comprador" VALUE="SOHO">
<INPUT TYPE="hidden" NAME = "linea1" VALUE="precio=10,cantidad=1,producto=xxx">
<INPUT TYPE="hidden" NAME = "linea2" VALUE="precio=10,cantidad=1,producto=yyy">
</FORM>
```

Y luego en el lado servidor:

```
ShoppingBasket oBasket = new ShoppingBasket();
oBasket.setCustomer(request.getParameter("id_comprador"));
oBasket.setProperty("tipo", request.getParameter("tipo_comprador"));
oBasket.addLine(request.getParameter("linea1"),",",");
oBasket.addLine(request.getParameter("linea2"),",",");
```

El carro de la compra puede almacenarse entonces en el [objeto Session](#) del servidor web para ser mantenido durante todo el proceso de compra.

Funciones de búsqueda y suma

Los carros permiten buscar una línea cuyo atributo tenga un determinado valor o sumar el valor de todos los atributos del mismo nombre en todas las líneas. Para que los atributos sean sumables su tipo debe ser `java.math.BigDecimal`.

Soporte multimonedas

El soporte multimonedas lo implementan las clases

```
com.knowgate.hipergate.DBCurrency y
com.knowgate.math.Money
```

La clase `Money` es una extensión de `java.math.BigDecimal` que añade la descripción de la moneda utilizada (`CurrencyCode`) al tipo base `BigDecimal`.

Para realizar conversiones entre monedas, se puede especificar manualmente un ratio, o dejar que la librería busque la conversión en tiempo real mediante una llamada al servicio web `ConversionRate` de `webserviceX.NET`. La llamada al servicio web puede tardar varios segundos por lo cual, en la medida de lo posible, es preferible almacenar los ratios de conversión cacheados localmente.

La clase `DBCurrency` se utiliza para acceder a las tablas `k_lu_currencies` y `k_lu_currencies_history`. La columna `nu_conversion` está pensada para almacenar un ratio de conversión fijo de una moneda base a otras monedas extranjeras. Dicha moneda base no está especificada en la base de datos y depende de la aplicación cliente.

6

Scripts Java BeanShell

Java BeanShell se proporciona como el interfaz estándar para ampliar la funcionalidad de hipergate mediante un lenguaje de scripting sin necesidad de recompilar la clases base.

La tecnología de scripting integrada en hipergate está basada en 3 componentes Open Source:

[Jakarta Bean Scripting Framework](#)

[BeanShell](#)

[Mozilla Rhino Shell](#)

Convenios en el paso de parámetros entre Java y BeanShell

BeanShell puede tomar y devolver objetos Java como parámetros de entrada y salida.

Por convenio se utilizan los siguientes nombres estándar para los parámetros:

DefaultConnection: Entrada. `JDBCConnection`. Conexión a base de datos principal. En las operaciones que involucran origen y destino se utiliza para leer datos (origen).

AlternativeConnection: Entrada. `JDBCConnection`. Conexión a base de datos alternativa. En las operaciones que involucran origen y destino se utiliza para escribir datos (destino).

ErrorMessage: Salida. `String`. Texto del mensaje de error o "" si no se produjo ningún error.

ReturnValue: Salida. Object. Valor de retorno del script.

Ejemplo tomado de `com.knowgate.hipergate.datamodel.ModelManager`

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

// Crea un nuevo dominio clonado del dominio MODEL (1025)
public int createDomain(JDCCConnection oConn, String sDomainNm)
throws EvalError, IOException, FileNotFoundException, SQLException
{
    Interpreter oInterpreter = new Interpreter();

    // Recuperar el código fuente del script contenido en el .JAR
    String sCode = ModelManager.getResourceAsString
        ("scripts/domain_create.js");

    // Asignar los parámetros de entrada del script
    oInterpreter.set("DomainNm", sDomainNm);
    oInterpreter.set("DefaultConnection", oConn);
    oInterpreter.set("AlternativeConnection", oConn);

    // Interpretar el 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;
}
```

hipergate no es sólo un conjunto de librería base, sino que incluye un completo interfaz de usuario final 100% basado en web y listo para utilizar.

Estructura general de las páginas

Las páginas se agrupan por subdirectorios según el módulo funcional al que pertenecen.

<code>/addrbook</code>	Herramientas Colaborativas
<code>/common</code>	Páginas Comunes usadas por varios módulos.
<code>/custom</code>	Páginas de instalaciones a medida.
<code>/crm</code>	Gestión de Relaciones con Clientes y Listas de Distribución.
<code>/dynapi</code>	DynAPI Cross-Browser DHTML Library
<code>/examples</code>	Ejemplos.
<code>/forums</code>	Foros.
<code>/includes</code>	Fragmentos de código HTML para componer otras páginas.
<code>/javascript</code>	Librerías JavaScript Cliente.
<code>/jobs</code>	Planificador de Tareas.
<code>/mailwire</code>	Envío de Newsletters.
<code>/methods</code>	Métodos estáticos Java para usar en páginas JSP
<code>/projtrack</code>	Gestión de Proyectos.
<code>/register</code>	Registro de Usuarios.
<code>/shop</code>	Tienda Virtual.
<code>/skins</code>	Decorados personalizables para la aplicación.
<code>/vdisk</code>	Disco Virtual y Categorías de Documentos.
<code>/wab</code>	Importación de Windows Address Book.
<code>/webbuilder</code>	Producción de Contenidos basados en plantillas

Juego de Caracteres del contenedor de servlets

Las versiones 1.x de hipergate utilizan el juego de caracteres ISO-8859-1 (Latin1) para todas las páginas.

A partir de la versión 2.0 dicho juego se cambió a UTF-8.

Para forzar al contenedor de servlets a utilizar Unicode se ha incluido la siguiente sentencia en el archivo /methods/dbbind.jsp :

```
<% request.setCharacterEncoding("UTF-8"); %>
```

En el caso de que por algún motivo el contenedor de servlets no soporte correctamente el metodo `setCharacterEncoding()` lo mejor es recodificar los parámetros de entrada en el método `loadrequest()` de los archivos /methods/reqload.jspf y /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"));
```

Convenios de programación

Cabeceras de página

- Se importarán sólo las clases utilizadas en cada página.
- Se usarán páginas sin beans de sesión.

```
<%@ page
import="java.io.IOException,java.net.URLDecoder,java.sql.SQLException,com.knowgate.jdc.JDCConnection,com.knowgate.acl.*"
language="java" session="false"
contentType="text/html; charset=UTF-8" %>
```

Manejo de Conexiones

- Las conexiones a base de datos deben obtenerse siempre del pool y con un nombre único por cada página que permita al recolector de estadísticas identificar desde dónde se solicitó la conexión.

- Las conexiones deben cerrarse explícitamente SIEMPRE incluso en el caso de que se produzca una excepción.
- Se deben acometer o descartar explícitamente TODAS las transacciones abiertas incluso en el caso de que se produzca una excepción.
- Las páginas obtener una conexión al inicio y devolverla al pull lo más rápido posible. Todas las operaciones de escritura en el cliente se efectuarán, con preferencia después de haber liberado la conexión.
- Ningún acceso a base de datos hecho desde una página puede tardar más de 20 segundos.

Tipos de Páginas

A grandes rasgos, existen 4 tipos de páginas en la suite:

- Páginas de Menú
- Páginas de Listado
- Formularios de Edición
- Páginas de Grabación y Borrado

El flujo de control comienza en las páginas de menú, pasa a las páginas de listado, luego a las de edición y finalmente a las de grabación y/o Borrado.

Beans de Aplicación

Acceso a Datos GlobalDBBind

El objeto `GlobalDBBind` es una instancia de la clase `com.knowgate.dataobjs.DBBind` que actúa como singleton de aplicación. `GlobalDBBind` mantiene la estructura del modelo de datos cacheada en memoria que se utiliza para las operaciones de persistencia automática de las clases heredadas de `DBPersist`. `GlobalDBBind` también mantiene una referencia interna al pool global de conexiones contra la base de datos. La primera vez que se instancia un `DBBind` los valores de conexión a la base de datos se leen del archivo `hipergate.cnf` (ver manual de instalación).



Un `DBBind` mantiene cacheada en RAM la estructura del modelo de datos aunque se hagan cambios. Por tal motivo es preciso reiniciar el servidor de

aplicaciones cada vez que se altere la estructura del modelo de datos para que los DBBind recarguen los cambios.

Conexiones a múltiples bases de datos

El bean `GlobalDBBind` lee sus parámetros de conexión a la base de datos del archivo `hipergate.cnf`. Dado que cada instancia de `DBBind` mantiene una copia en memoria de la estructura del modelo de datos y un pool de conexiones, cada instancia del bean sólo puede estar conectado a una base de datos simultáneamente.

La forma más sencilla de conectarse a otras bases de datos es usar una clase derivada de `DBBind` que lea sus parámetros de inicialización de otro archivo.

La instalación estándar de `hipergate` proporciona 4 clases precompiladas para este propósito: `DBPortal`, `DBTest`, `DBDemo` y `DBReal`. Cada una de las cuales lee por defecto de los archivos `/etc/portal.cnf`, `/etc/test.cnf`, `/etc/demo.cnf` y `/etc/real.cnf`, (o `C:\WINNT\portal.cnf` en Windows). Estas 4 clases son herencias muy simples de `DBBind` que no implementan ninguna funcionalidad adicional, excepto que leen de archivos de configuración distintos y mantienen pools de conexiones independientes.

Cache Distribuido GlobalCacheClient

`GlobalCacheClient` Es una instancia de la clase `com.knowgate.cache.DistributedCachePeer`. A nivel local el cache es una tabla hash (`java.util.HashMap`) que asocia cadenas de texto (tokens) con objetos. El cache tiene un número limitado de entradas (por defecto 400). El tamaño de los objetos almacenados no se tiene en cuenta, de modo que no se deben almacenar objetos grandes si el consumo de memoria es un factor a tener en cuenta.

El cache local de cada servidor web puede funcionar de forma coordinada con los caches locales de otros servidores web para mantener la consistencia de los datos cacheados en un entorno donde múltiples máquinas pueden actualizar concurrentemente los mismos datos. Para funcionar en granjas de servidores se requiere un coordinador de caches, es un servicio común que mantiene información sobre la fechas de última actualización de cada token a través de un conjunto de servidores. En la

instalación inicial, hipergate viene configurado para trabajar sólo con el cache local, de modo que no hay que preocuparse del coordinador de cache a menos que se instale una configuración con sesiones distribuidas en múltiples servidores.

El cache se utiliza sobre todo para mantener datos de valores de remonte por áreas de trabajo y para mantener en el servidor la clave de acceso y el rol de los usuarios conectados.

Sesiones de Usuario

Cookies

hipergate no mantiene sesiones en el servidor, la información del usuario conectado se almacena en cookies de sesión en el equipo cliente. Toda la información se pasa de un formulario a otro mediante métodos GET y POST de http, sin que exista ningún controlador centralizado de flujo.

Por estar basado en cookies, no es posible tener dos sesiones abiertas concurrentemente desde la misma máquina cliente con usuarios distintos.

Las cookies de sesión se graban en la página `/common/login_chk.jsp` y son las siguientes:

domainid	Identificador del Dominio al que se está conectado.
domainnm	Nombre del Dominio.
skin	Nombre del subdirectorío del decorado (skin) activo para la aplicación. Este directorío contiene la hoja de estilo CSS y las imágenes que determinan el aspecto de la aplicación.
userid	GUID del Usuario conectado.
authstr	Clave de acceso del Usuario, puede estar encriptado o no en función de cómo se haya configurado la página <code>login_chk.jsp</code> . Por defecto las claves van en texto simple sin encriptar.

appmask	Máscara de bits de derechos de acceso a las aplicaciones de la suite. Cada aplicación se identifica con un número entre 1 y 31. Dicho número representa un bit de un número de 32 bits. El código del archivo <code>/common/tabmenu.jsp</code> mostrará la pestaña de la aplicación sólo si su bit correspondiente está a 1.
idaccount	Nº de cuenta de acceso (tabla <code>k_accounts</code>) para los usuarios en modalidad ASP.
workarea	GUID del Área de Trabajo actual.
path_workarea	Ruta de archivos del Área de Trabajo actual.

Información cacheada

Por motivos de rendimiento, el rol de los usuarios conectados se almacena en el caché. En cierto sentido, esto es equivalente a mantener una sesión en el servidor para cada usuario conectado, con la diferencia de que existe un límite absoluto para el consumo de recursos en el servidor y que las sesiones pueden ser descartadas en caso de sobrecarga del sistema y luego recuperadas nuevamente de forma transparente para el usuario.

Objetos cacheados por usuario (si el objeto no existe en cache se entiende que es lo mismo que si el objeto asociado tuviese el valor **false**).

Token	Tipo	Descripción
<code>[UserId,trial]</code>	Boolean	<code>true</code> para las cuentas de prueba
<code>[UserId,owner]</code>	Boolean	<code>true</code> si es el administrador del dominio
<code>[UserId,admin]</code>	Boolean	<code>true</code> si el usuario pertenece al grupo de administradores del dominio.
<code>[UserId,powuser]</code>	Boolean	<code>true</code> si pertenece al grupo de usuarios avanzados
<code>[UserId,user]</code>	Boolean	<code>true</code> si pertenece al grupo de usuarios
<code>[UserId,guest]</code>	Boolean	<code>true</code> si pertenece al grupo de invitados
<code>[UserId,options]</code>	String	Texto HTML de las opciones del menu.
<code>[UserId,suboptions]</code>	String	Textos HTML de las subopciones del menu.
<code>[UserId,authstr]</code>	String	Clave de acceso del usuario
<code>[UserId,mailbox]</code>	String	Nombre del buzón de correo

Proceso de creación de cuentas

Tipos de cuentas

El objeto 'cuenta' se encuentra representado en base de datos por la tabla k_billing.

En modalidad ASP podemos distinguir tres tipos de cuentas : Corporativa, profesional y de sistema (k_billing.tp_account = 'C', k_billing.tp_account = 'P' y k_billing.tp_account = 'S' respectivamente) .

La cuenta corporativa está adscrita a un dominio, pudiendo entonces varios usuarios, los pertenecientes al dominio, pertenecer a la misma cuenta. Se toma como convenio que el usuario responsable administrativo de la cuenta (k_billing.gu_user) es el usuario 'administrador' del dominio. Si suponemos que una empresa contrata utiliza un dominio para realizar sus tareas corporativas mediante los distintos módulos de hipergate esta cuenta debería ser el punto de relación con un futuro sistema de facturación.

La cuenta profesional está adscrita a un usuario final determinado, un usuario final que estará circunscrito a un dominio determinado. Se toma como convenio el que todos los usuarios con cuenta profesional pertenezcan a un mismo dominio.

Por último, la cuenta de sistema es utilizada para identificar a usuarios superadministradores que pueden de esta forma controlar el sistema de forma general.

El proceso de registro

Se da la oportunidad de que el usuario se registre por un periodo de prueba en principio establecido en 60 días, momento en el cual la cuenta pasa a estar en estado 'en pruebas' (k_billing.bo_trial = 1, k_billing.bo_active=1, k_billing.dt_cancel= now() + 60días).

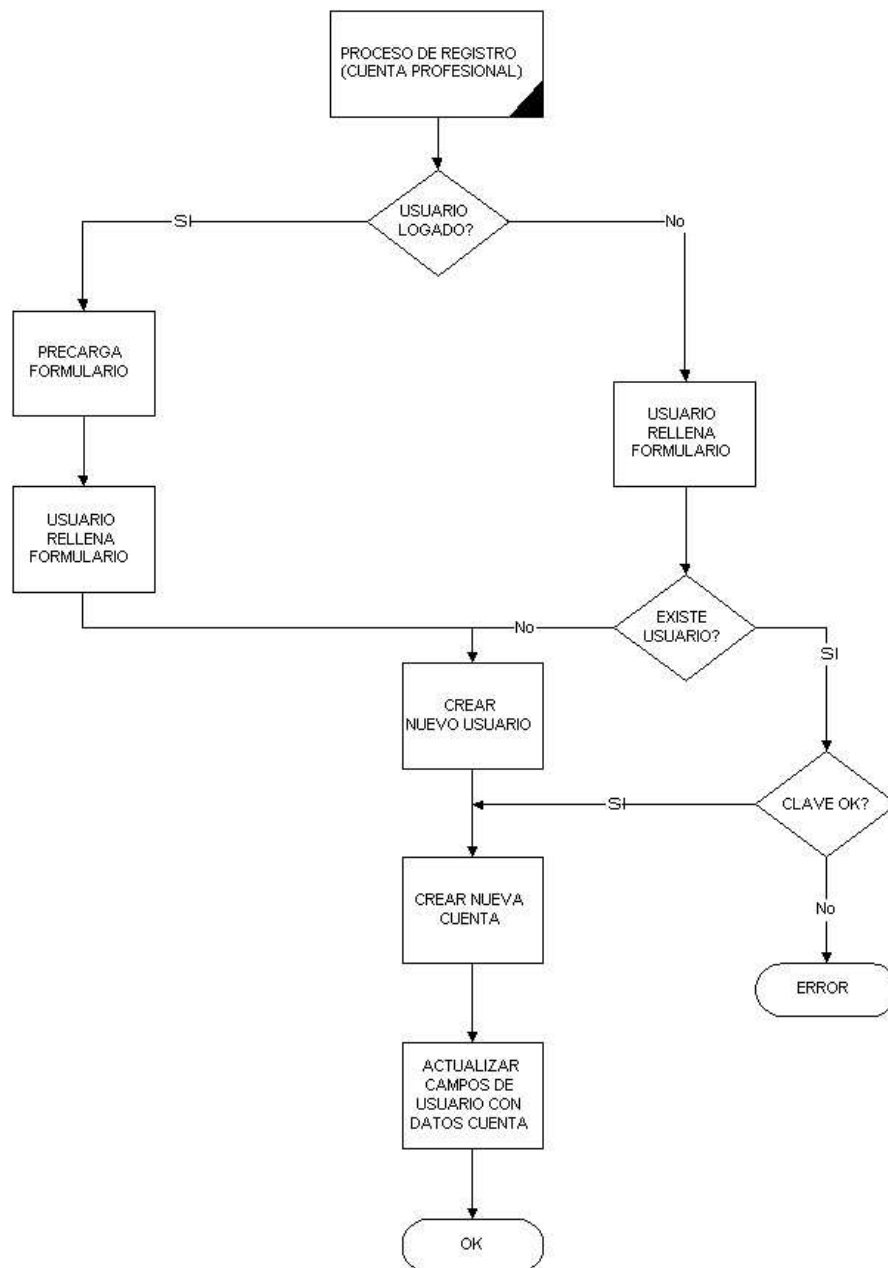
Durante dicho periodo la cuenta permanece en estado 'en pruebas' (k_billing.bo_trial = 1, k_billing.bo_active=1).

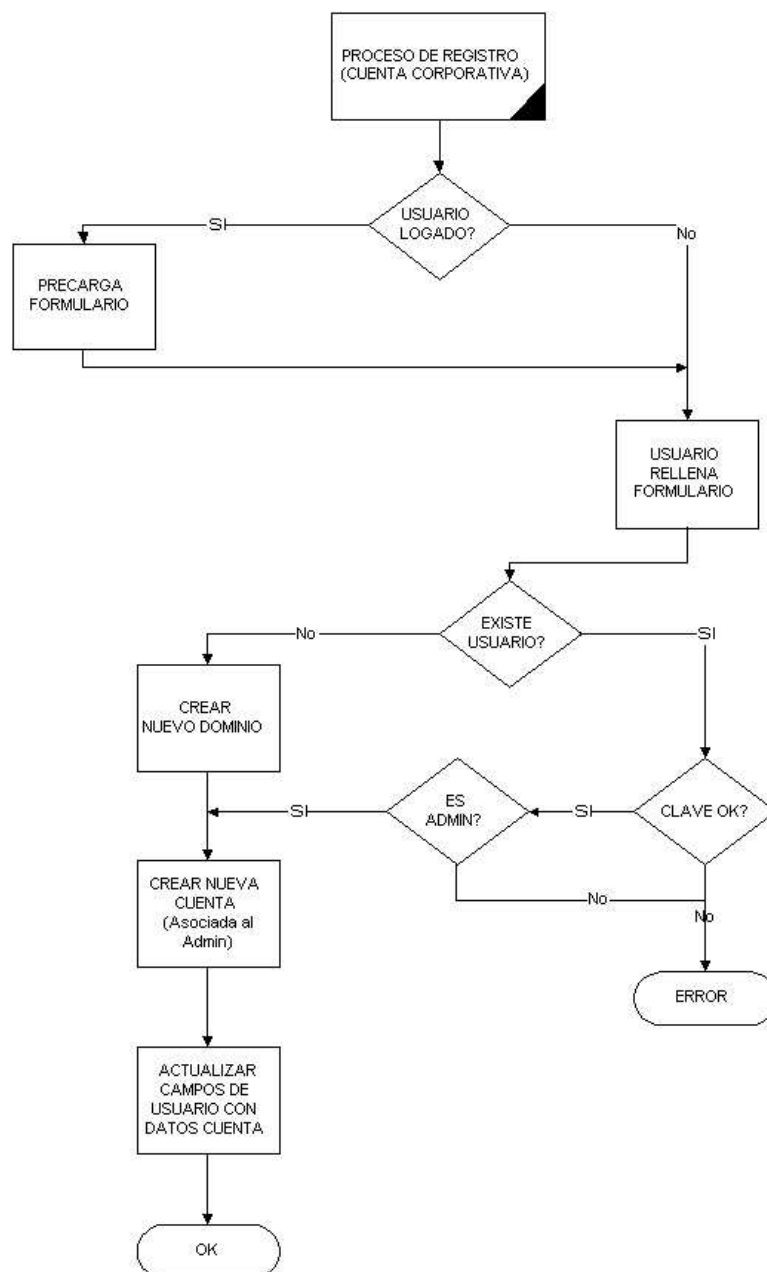
Finalizado el periodo de prueba ($\text{now}() < \text{k_billing.dt_cancel}$) si no se realiza el pago correspondiente la cuenta pasa a estar inactiva ($\text{k_billing.bo_active} = 0$).

Si en el transcurso de este tiempo el usuario accede al pago del importe correspondiente la cuenta pasa a ser 'cuenta de pago' ($\text{k_billing.bo_trial} = 0$, $\text{k_billing.bo_active} = 1$, $\text{k_billing.dt_cancel} = \text{k_billing.dt_cancel} + 365\text{días}$), suponiendo que la cuenta se renueva anualmente.

La comprobación de la caducidad de la cuenta se realiza de forma pasiva, es decir, no existe ningún proceso de sistema corriendo en segundo plano. Es el usuario cuando se registra el que lanza la comprobación de si su cuenta esta operativa (debe cumplirse que $\text{now}() > \text{k_billing.dt_cancel}$) y en caso contrario esta debe desactivarse ($\text{k_billing.bo_active} = 0$). Ver procedimiento almacenado ' k_check_account '.

Mostramos a continuación los diagramas de flujo del proceso de registro para cuentas profesional y corporativa respectivamente. Las cuentas de sistema se crean por defecto cuando se instala el sistema por primera vez.






Proceso de autenticación de usuarios

Autenticación Inicial

El proceso de autenticación inicial se realiza en la página `/common/login_chk.jsp` desde allí se efectúan los siguientes pasos:

- 1º) Si la autenticación es a través de un par {e-mail , clave} hay que buscar previamente a qué usuario pertenece el e-mail y cual es el área de trabajo asignada por defecto a dicho usuario. Esta información se extrae de los campos `tx_main_email` y `gu_workarea` de la tabla `k_users` llamando al método `com.KnowGate.acl.ACLUser.getIdFromEmail()`.
- 2º) Si la autenticación es a través del conjunto parámetros {nick , clave , dominio, área de trabajo} hay que extraer el GUID del usuario correspondiente al nick y dominio dados.
- 3º) Una vez que se ha recuperado el GUID del usuario (si existe) llamar al método `com.knowgate.acl.ACL.authenticate()` para verificar que el usuario está activado, que la clave de acceso coincide, que la clave no está caducada.
- 4º) En modalidad ASP, para los usuarios adscritos a cuentas facturables (tabla `k_billing`) verificar que la cuenta existe, está activada y no ha caducado.
- 5º) A partir de la versión 2.2, todos los intentos de conexión se graban en la tabla [`k_login_audit`](#).
- 6º) Una vez se ha verificado todo lo anterior, escribir la cookies de sesión, guardar en el cache el rol del usuario en el área de trabajo y redireccionar a la página `/common/desktop.jsp`.

 Los logins de usuario pueden activarse y desactivarse sin alterar ningún otro parámetro. Si el campo `k_users.bo_active` está a cero, el proceso de autenticación detectará la cuenta como inactiva y rechazará el intento de inicio de sesión.

Re-autenticación por página

Como el servidor no mantiene estados, es necesario autenticar cada petición HTTP cuando se produce. Esto se hace llamando al método estático `JSP.authenticateSession(GlobalDBBind, request, response)` incluido en el archivo `/includes/authusr.jsp`.

Cómo conectar un usuario a diferentes Áreas de Trabajo

El proceso de autenticación utiliza normalmente el e-mail y la clave de un usuario para conectarlo a la aplicación.

☞ Recordemos que los usuarios pertenecen a los dominios, no a las áreas de trabajo, por consiguiente, cuando se crea un usuario se le asigna normalmente un área de trabajo por defecto (campo `k_users.gu_workarea`).

Autenticación estándar

La autenticación estándar de la página `/common/login_chk.jsp` sólo toma como parámetros de entrada el e-mail y la clave del usuario.

La tabla `k_users` tiene un índice único por e-mail que permite recuperar el GUID de un usuario a partir de su e-mail.

Asimismo, el campo `gu_workarea` de la tabla `k_users` determina el área de trabajo a la que el usuario será conectado.

Autenticación extendida

En algunos casos, puede ser conveniente que el mismo usuario pueda conectarse a varias áreas de trabajo. Esto se consigue cambiando el par {email, clave} por {nickname, nombre dominio, area de trabajo, clave}. La página `login_chk.jsp` puede procesar indistintamente ambos juegos de parámetros de conexión.

En términos prácticos, para conectar a un usuario a un área de trabajo distinta de la de por defecto reemplazar la página `login.html` con un formulario que contenga los campos:

```
<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>
```

Cómo reemplazar el modelo de seguridad nativo

Toda la lógica de autenticación está contenida en la clase `com.knowgate.acl.ACL`, el incluye `authusrs.jsp`, y la página `login_chk.jsp`. Reemplazando estas partes de código es posible que la aplicación utilice otro sistema de autenticación, por ejemplo basado en LDAP.

No obstante, hay que tener en cuenta que los permisos de acceso en las Categorías están vinculados al modelo de Usuarios y Grupos. Por consiguiente si se descarta el modelo de seguridad nativo habrá que modificar el código de las categorías para que reconozcan el nuevo modelo.

Existen 3 métodos de login de usuario incluidos dentro del producto estándar: **nativo**, **LDAP** y **NTLM**.

Autenticación Nativa:

Al inicio de sesión la página `/common/logi_chk.jsp` recibe por POST el e-mail del usuario y su clave de acceso. A partir del e-mail se halla el GUID del usuario y su área de trabajo por defecto. Se verifica la clave de acceso y, si es válida y la cuenta no está desactivada ni expirada se permite el acceso. El GUID del usuario y su clave encriptada se guardan en cookies de sesión que se reciben en cada petición HTTP.

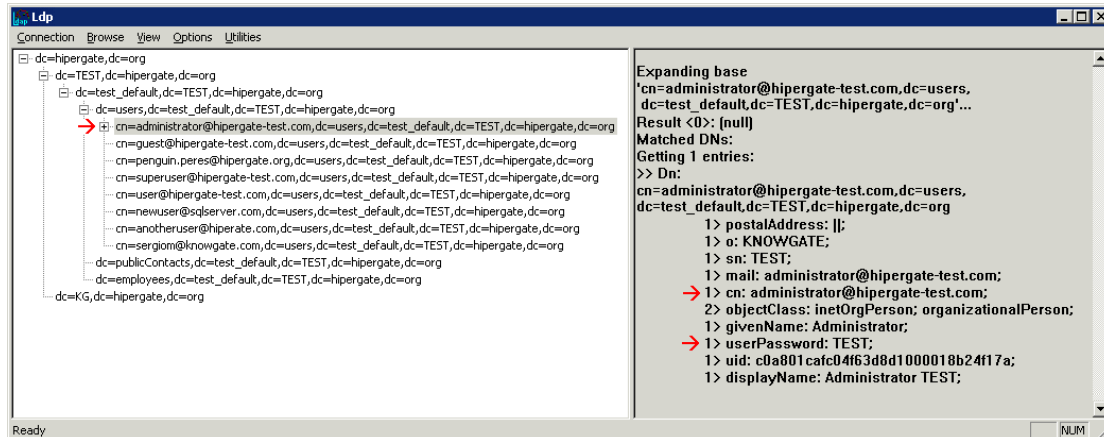
La clave del usuario se guarda desencriptada en el cache para evitar accesos repetidos a la base de datos cada vez que se solicita una página y hay que verificar la clave. hipergate es una aplicación sin estados, y por consiguiente, no usa sesiones en el servidor. Por ello es preciso volver a comprobar la clave del usuario en cada petición HTTP.

Autenticación LDAP:

La clave para un usuario puede guardarse en un directorio LDAP. Para que funcione la implementación por defecto este directorio debe tener la estructura que se describe en el capítulo de integración con directorios LDAP.

La clave para cada usuario se guarda desencriptada en el campo `userPassword` del siguiente objeto LDAP:

cn=usuario@mail.com,dc=users,dc=nombre_area_de
trabajo,dc=NOMBRE_DOMINIO,dc=hipergate,dc=org



Cómo simular accesos de usuarios anónimos

La forma más sencilla de permitir accesos anónimos a hipergate es conectar automáticamente a todos los usuarios anónimos con una cuenta de invitado del área de trabajo.

Por ejemplo esta página HTML conecta automáticamente al usuario y lo redirige a la página principal de hipergate.

```
<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>
```

El skin debe ser siempre xp.

Los valores para las cookies `domainid` y `domainnm` deben cogerse de la tabla `k_domains`.

Los valores para las cookies `userid`, `authusr` y `workarea` deben cogerse de la tabla `k_users`.

Una vez dentro de la aplicación los usuarios anónimos se detectan por su rol de invitados mediante la función `isDomainGuest()` contenida en el archivo `/methods/authusrs.jspf`

La mayoría de las páginas de hipergate se ponen en modo de sólo lectura cuando se entra con rol de invitado pero es conveniente verificar este punto para evitar fallos de seguridad.

Si se desea dar la opción de que los usuarios anónimos se autentifiquen en cualquier momento lo mejor añadir una caja de login /common/tabmenu.jspf, existen una cuantas líneas de código de ejemplo comentado en dicho archivo.

[illegible]

Librerías de Métodos Estáticos para páginas JSP

Estos métodos se encuentran en el subdirectorio `/methods` y se incluyen dentro de las páginas con directivas tipo

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

authenticateCookie

```
short authenticateCookie (
    DBBind dbb, HttpServletRequest req, HttpServletResponse res)
throws ClassNotFoundException, InstantiationException,
ServletException
```

Comprueba que las cookies `userid` y `authstr` corresponden a una combinación de usuario y clave válidas. La validación de claves se hace preferentemente desde el cache local. En caso de que la clave no esté en el cache o haya sido modificada, se llama al método `com.knowgate.acl.ACL.authenticate()`

Retorno: 0 si la combinación es válida ó uno de los siguientes valores si la combinación no es válida.

```
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
```

authenticateSession

```
short authenticateSession (
    DBBind dbb, HttpServletRequest req, HttpServletResponse res)
throws ClassNotFoundException, InstantiationException,
ServletException, IOException
```

Comprueba que las cookies `userid` y `authstr` corresponden a una combinación de usuario. Si la combinación no es válida redirecciona a la página `/common/errmsg.jsp`.

Retorno: 0 si la combinación es válida ó uno de los siguientes valores si la combinación no es válida.

```
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)
```

Retorno: "es" si el navegador cliente está configurado con su primer idioma español. "en" para cualquier otro idioma.

getCookie

```
String getCookie (HttpServletRequest req, String sName,  
String sDefault)
```

Obtiene el valor de la cookie o devuelve un valor por defecto si la cookie no se encuentra.

nullif.jsp

nullif

```
String nullif (String sParam)
```

Devuelve el parámetro de entrada o " " si el parámetro de entrada es **null**

nullif

```
String nullif (String sParam, String sDefaultVal)
```

Devuelve el parámetro de entrada o el valor por defecto sDefaultVal si el parámetro de entrada es **null**

reqload.jsp

loadRequest

```
void loadRequest (ServletRequest r, DBPersist p)  
throws NumberFormatException, java.text.ParseException
```

Carga todos los valores de un DBPersist presentes en la colección de parámetros del ServletRequest. La subrutina recorre una a una las columnas del DBPersist y busca si existe un parámetro no nulo y no vacío (" ") en el ServletRequest con el nombre de la columna. Cada parámetro se convierte del tipo String de la colección de parámetros al tipo adecuado según la definición de columna del DBPersist.

hasXssSignature

```
boolean hasXssSignature (String s)
```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

Verifica mediante expresiones regulares si una cadena es candidata probable a ser un ataque por cross-site scripting (XSS). Devuelve true si la cadena parece un intento de ataque XSS, false en caso contrario.

hasSqlSignature

```
boolean hasSqlSignature (String s)
```

Verifica mediante expresiones regulares si una cadena es candidata probable a ser un ataque por inyección de SQL. Devuelve true si la cadena parece un intento de ataque, false en caso contrario.

safeXssGetParameter

```
String safeXssGetParameter (HttpServletRequest r, String n)  
throws SecurityException
```

Obtiene un parámetro del objeto request de JSP lanzando una excepción de seguridad si éste cumple con las expresiones regulares utilizadas por hasXssSignature para detectar ataques cross-site scripting.

safeSqlGetParameter

```
String safeSqlGetParameter (HttpServletRequest r, String n)  
throws SecurityException
```

Obtiene un parámetro del objeto request de JSP lanzando una excepción de seguridad si éste cumple con las expresiones regulares utilizadas por hasSqlSignature para detectar ataques por inyección de SQL.

El menú superior de pestañas

El menú se encuentra en el archivo `/common/tabmenu.jsp`

La aplicación no utiliza marcos; el menu debe incluir manualmente en cada una de las páginas en las que deba aparecer inmediatamente al principio del cuerpo de la página:

```
<%@ 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>
```

El código del menu requiere que los métodos del archivo `cookies.jsp` y los beans `GlobalDBBind` y `GlobalCacheClient` hayan sido incluidos previamente.

Cómo mostrar la opción seleccionada

La selección de opciones está dividida en 2 niveles. La opción y subopción seleccionadas deben pasarse en los parámetros `selected` y `subselected` de la URL con índices a partir de cero. Si el índice es negativo, no se mostrará ninguna opción seleccionada.

Máscara de Aplicaciones

Para cada Usuario y Área de Trabajo existe una máscara de aplicaciones que viene determinada por los Grupos a los que pertenezca el usuario que ejerzan algún rol sobre el área de trabajo.

La máscara de aplicaciones es un entero de 32 bits con un bit por aplicación.

Típicamente, cada aplicación se corresponde con una pestaña del nivel superior del menú.

Cache de opciones de menú

Para mejorar eficiencia en la composición del menú, las opciones de cada usuario se guardan en el cache local usando el bean de aplicación `GlobalCacheClient`. Como el menú cambia según los valores de los parámetros `selected` y `subselected` no es posible almacenar un único menú por sesión de usuario sino que el cache debe almacenar el menú en todas sus posiciones.

Acceso a Configuración para el administrador del dominio

Independientemente de lo que se configure en los roles y aplicaciones del Área de Trabajo, el administrador del dominio siempre tendrá acceso

desde el menú a las utilidades de Configuración. Es así para evitar que el rol de administración se elimine por causas fortuitas del módulo de Configuración y la aplicación quede en un estado imposible de administrar desde el interfaz web.

Personalización de la página de inicio

A partir de la versión 2.1 hipergate incluye la posibilidad de personalizar la página de inicio para cada usuario.

hipergate personaliza la página usando un tipo de pseudo-portlets (clases Java que implementan el interfaz `javax.portlet.GenericPortlet`). Las subclases de hipergate que implementan `GenericPortlet` no son auténticos portlets porque hipergate no requiere un contenedor de portlets para funcionar sino que emula el comportamiento de un contenedor de portlets auténtico sobre el objeto `HttpServletResponse`.

Para cada portlet los siguientes elementos están implicados en el proceso de composición :

- La página que contiene el portlet
- La clase Java que implementa el interfaz de portlet
- Una hoja de estilo que presenta la información como XHTML
- Una fila en `k_x_portlet_user` que indica la posición y estado
- Un conjunto de propiedades de entorno para el portlet
- Un archivo con el contenido del portlet cacheado para cada usuario y área de trabajo.

Como se realiza la composición de la página desktop.jsp

1. Para cada usuario y zona, se recupera un `DBSubset` del bean de aplicación `GlobalCacheClient` de la clase `DistributedCachePeer`. Si no hay ninguna entrada válida en el cache, se lee de la tabla `k_x_portlet_user` y se crea la entrada.
2. Para cada zona, se itera por la lista de portlets que deben mostrarse para el usuario y área de trabajo actuales.
3. Se recogen las propiedades de entorno requeridas por `GenericPortlet.render()` y se pasan a una instancia de `HipergateRenderRequest`.
4. Desde dentro del método `render()` determinar si la fecha en el campo `dt_modified` de `k_x_portlet_user` es anterior o posterior a la fecha de creación del fichero de cache del portlet almacenado

en el directorio `storage/domains/domain_id/workareas/workarea_guid/cache/user_id` si el cache no es válido, obtener la hoja de estilo XSL del `StylesheetCache`, fusionar la hoja de estilo con los datos XML del portlet, actualizar el cache y volcar los resultados por el objeto `RenderResponse`.

Cache para portlets

Dado que los portlets son llamados cada vez que se carga una página, y que cada página puede contener varios portlets, es importante que el mecanismo para pintarlos sea eficiente. Los portlets de hipergate disponen de 3 mecanismos de cache:

- 1º) La información de qué portlets deben pintarse en cada página y en qué estado, se almacena en el bean de aplicación `GlobalCacheClient` de clase `DistributedCachePeer`.
- 2º) La salida del portlet se cachea para cada página, usuario y área de trabajo en un fichero bajo la rama de directorios `/storage`. Mientras el cache sea válido, no realiza ningún acceso a base de datos para pintarlo.
- 3º) Cada portlet necesita una hoja de estilo XSL para formatear su salida. Cargar una hoja de estilo XSL puede ser un proceso computacionalmente costoso. Por ello los portlets mantienen un cache de hojas de estilo pre-cargadas en memoria.

Selección de valores de remonte

Si se sigue el convenio del modelo de datos para la creación de tablas de remonte, es posible reutilizar la misma página JSP para todos los valores de remonte de la aplicación.

El convenio de creación de remotes es crear una tabla base (por ejemplo `k_base`) y otra tabla `k_base_lookup` (mismo nombre que la tabla base pero terminado en `_lookup`). La tabla `k_base_lookup` contiene todos los valores de remonte para la tabla `k_base` por cada Área de Trabajo (cada Área de Trabajo puede tener sus propios valores de remonte privados).


Sólo se admiten valores de remonte de tipo texto.

Para cada remonte se especifica:

- 1º) El nombre de la sección –típicamente el nombre del campo en la tabla `k_base` aunque no siempre (`k_base_lookup.is_section`)–.
- 2º) El valor interno para el remonte (`k_base_lookup.vl_lookup`).
- 3º) la etiqueta traducida a cada idioma del valor del remonte.

Cómo llamar a la página lookup_f.jsp

La página lookup_f.jsp se abre en ventana nueva desde los formularios de mantenimiento de cada tabla.

 Es posible abrir todos los formularios de remonte en la misma venta o en ventanas separadas. Internet Explorer abre ventana mucho más rápido cuando no se especifica un nombre para ellas. Por este motivo se recomienda abrir cada remonte en su propia ventana.

Parámetros	<code>nm_table</code>	Nombre de la tabla de remonte.
	<code>id_language</code>	Idioma para mostrar las etiquetas.
	<code>id_section</code>	Sección, normalmente el nombre del campo a rellenar en la tabla base.
	<code>nm_control</code>	Nombre del control HTML en el formulario de mantenimiento. Cuando se seleccione un valor la página de lookup traspasará su etiqueta traducida a este control automáticamente y cerrará la ventana.
	<code>nm_coding</code>	Campo oculto en el formulario de mantenimiento donde se almacena el valor interno del lookup.
	<code>tp_control</code>	Tipo de control en la tabla base. 1 <INPUT TYPE="text"> 2 <SELECT>

Carga de remotes desde scripts SQL

Los valores de remonte pueden cargarse directamente por SQL en vez de hacerlo desde el front-end.

Para ello basta con identificar la tabla base e insertar los registros deseados en su tabla de lookup correspondiente. Por ejemplo:

```
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' );
```

Los campos a rellenar son:

gu_owner: GUID del Área de Trabajo a la que pertenecerán los registros.

id_section: Nombre del campo en la tabla base.

pg_lookup: Ordinal progresivo del valor. Único por cada valor en un área de trabajo pero puede repetirse en diferentes áreas de trabajo.

vl_lookup: Valor que se insertará en el campo especificado en **id_section** en la tabla base.

tr_es: Etiqueta traducida español.

tr_en: Etiqueta traducida inglés.

Atributos definidos por el usuario

hipergate permite añadir atributos definidos por el usuario a las entidades estándar. Estos atributos no se crean físicamente como campos en el modelo de datos sino que se almacenan verticalmente en tablas separadas. Ver [Modelo de Datos de Atributos Definibles por el Usuario](#).

En la capa de JSP, el soporte para atributos definibles por el usuario lo proporciona el módulo `/methods/customattrs.jsp` este módulo debe ser incluido en las páginas JSP mediante

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

Para pintar en un formulario los atributos definidos por el usuario, el método `paintAttributes()` hace lo siguiente:

1. Lee la definición de metadatos de la tabla `k_lu_meta_attrs` que corresponde a la Tabla y Área de Trabajo actuales.

2. La información de metadatos se almacena en el cache local con la clave `tabla#idioma[workarea]` -por ejemplo `k_contacts_attrs#en[012345678901234567890123456789AB]`. La próxima vez que se llame al método `paintAttributes()` los metadatos se leerán del cache de memoria y no de la tabla de bb.dd.
3. Si el usuario conectado pertenece al grupo de administradores del Área de Trabajo, se pintan los enlaces para añadir y eliminar atributos.
4. Los nombres de los atributos definidos se listan separados por comas en el objeto HTML `<INPUT TYPE="hidden" NAME="custom_attributes">`. Esta lista oculta es utilizada por el método `storeAttributes()` para saber qué atributos existen.
5. Finalmente se escribe cada atributo con su correspondiente valor.
6. El método `paintAttributesHidden()` es igual que `paintAttributes()` excepto que pinta los atributos en objetos HTML ocultos en vez de en objetos editables. Esto se utiliza para traspasar datos en páginas que tienen más de un formulario.

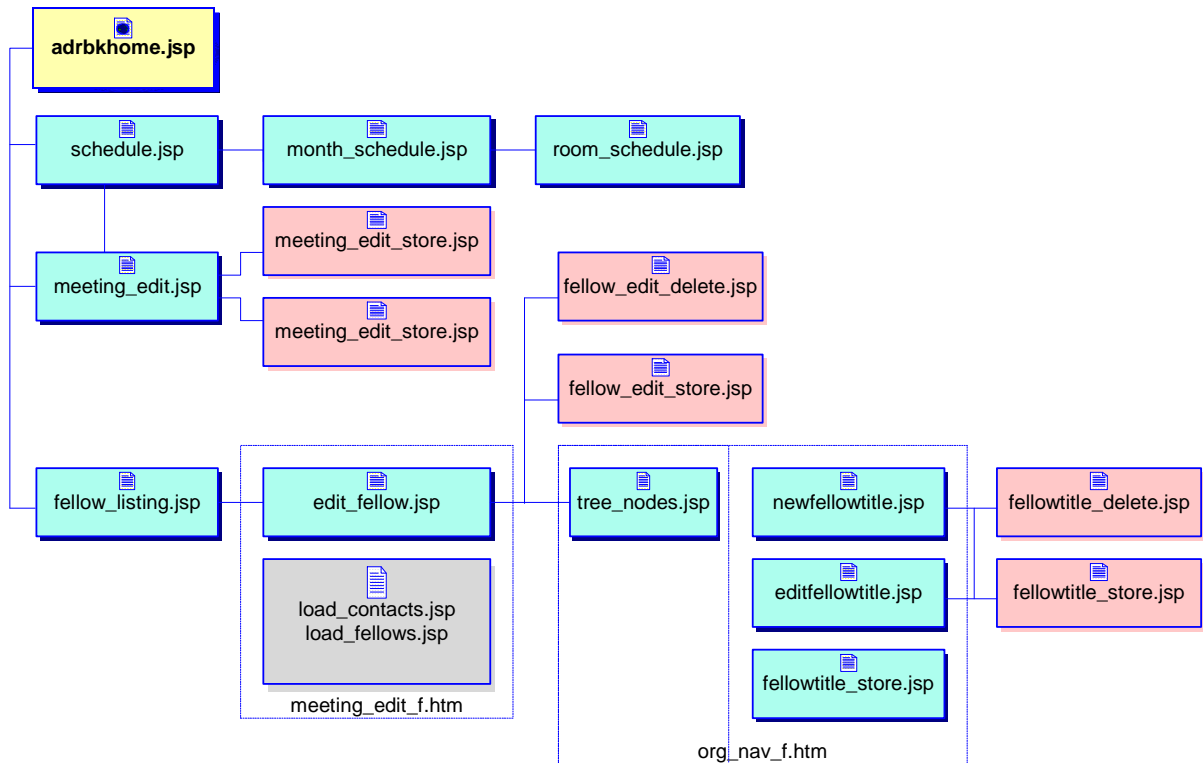
Envío de correo en respuesta a acciones

La clase `com.oreilly.servlet.MailMessage` proporciona un interfaz sencillo sobre JavaMail para el envío de mensajes de correo por SMTP.

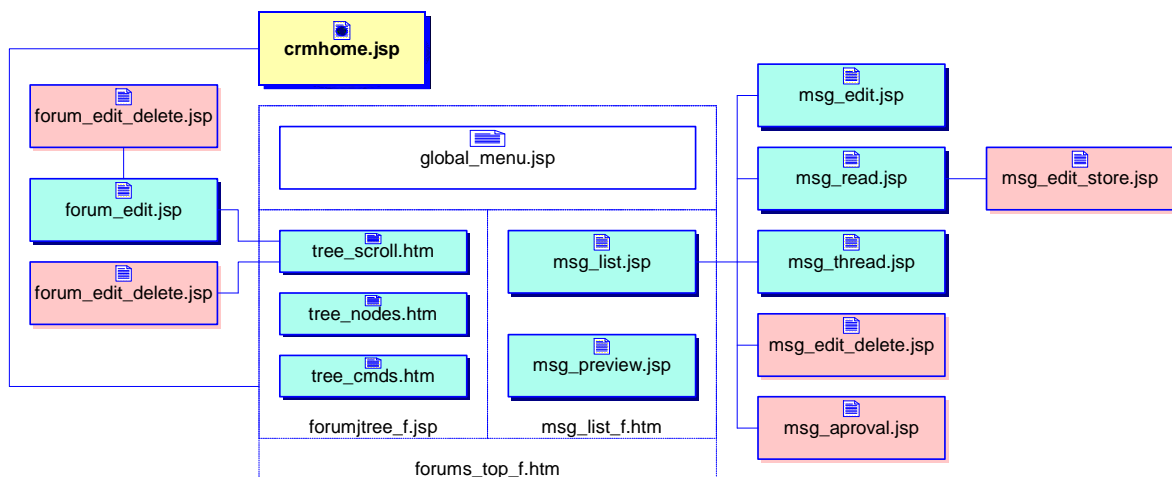
El siguiente código de ejemplo usando `MailMessage` puede ser agregado a cualquier página estándar de grabación de datos.

```
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();
```

Módulo de Trabajo en Grupo



Módulo de Foros



Modo de almacenamiento de los mensajes

El cuerpo de los mensajes se almacena en texto simple en el campo `tx_msg` de la tabla `k_newsmgsge`. El contenido de los mensajes puede ser texto plano o HTML pero es responsabilidad de la capa de presentación manejar los tags HTML.

El cuerpo de los mensajes puede ser tan grande como lo permita la longitud de los campos CLOB o LONGVARCHAR de la base de datos.

Los archivos adjuntos se almacenan fuera de la base de datos bajo un subdirectorio de la rama `/storage`. Los archivos se referencian mediante la tabla `k_prod_locats`. Si un mensaje contiene archivos adjuntos, entonces su columna `gu_product` apunta a un objeto en la tabla `k_products`. Cada una de las entradas en `k_prod_locats` para este objeto representa un archivo adjunto.

Los archivos adjuntos a los mensajes se descargan utilizando el servlet `com.knowgate.http.HttpBinaryServlet`, véase el API JavaDoc para más información sobre `HttpBinaryServlet`.

Generación de documentos RDF Site Summary (RSS)

A partir de la versión 2.0, hipergate incluye un par de páginas de ejemplo sobre cómo generar documentos RSS para publicar mensajes de los foros.

<code>/forums/msg_rss10.jsp</code>	Genera RSS 1.0
<code>/forums/msg_rss20.jsp</code>	Genera RSS 2.0

Estas páginas son sólo un fundamento para generar archivos XML a partir de los mensajes en los foros. Deben ser modificadas a la medida de cada aplicación cliente para ser plenamente operativas.

Parámetros de entrada

Cada página toma por GET o POST los siguientes parámetros de entrada:

`gu_newsgroup` GUID del grupo de mensajes (NewsGroup).

`nm_newsgroup` Nombre de grupo de mensajes.

`id_language` Identificador del idioma (código de 2 caracteres).

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

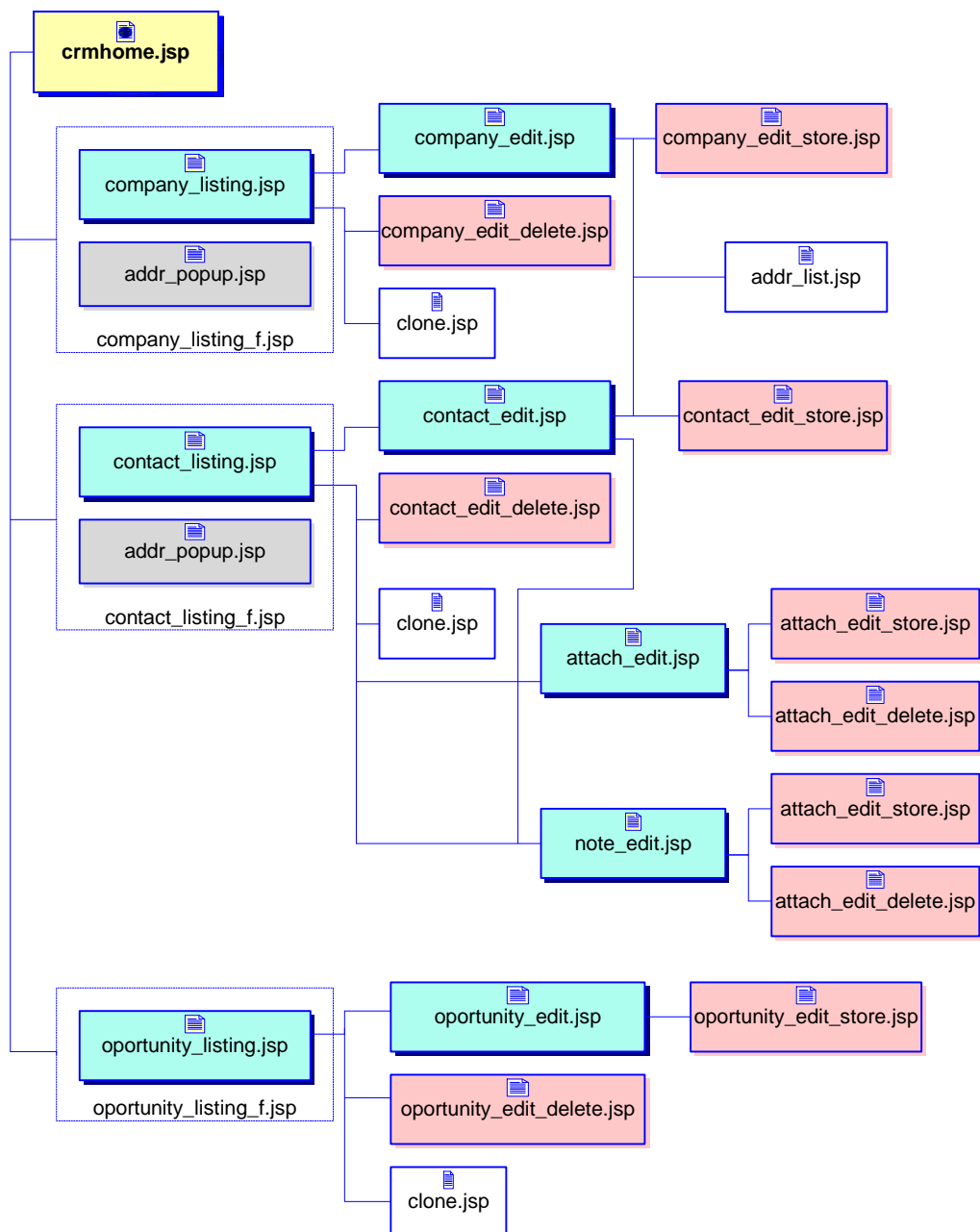
`nu_messages` Número máximo de mensajes a mostrar.

Parámetros de control

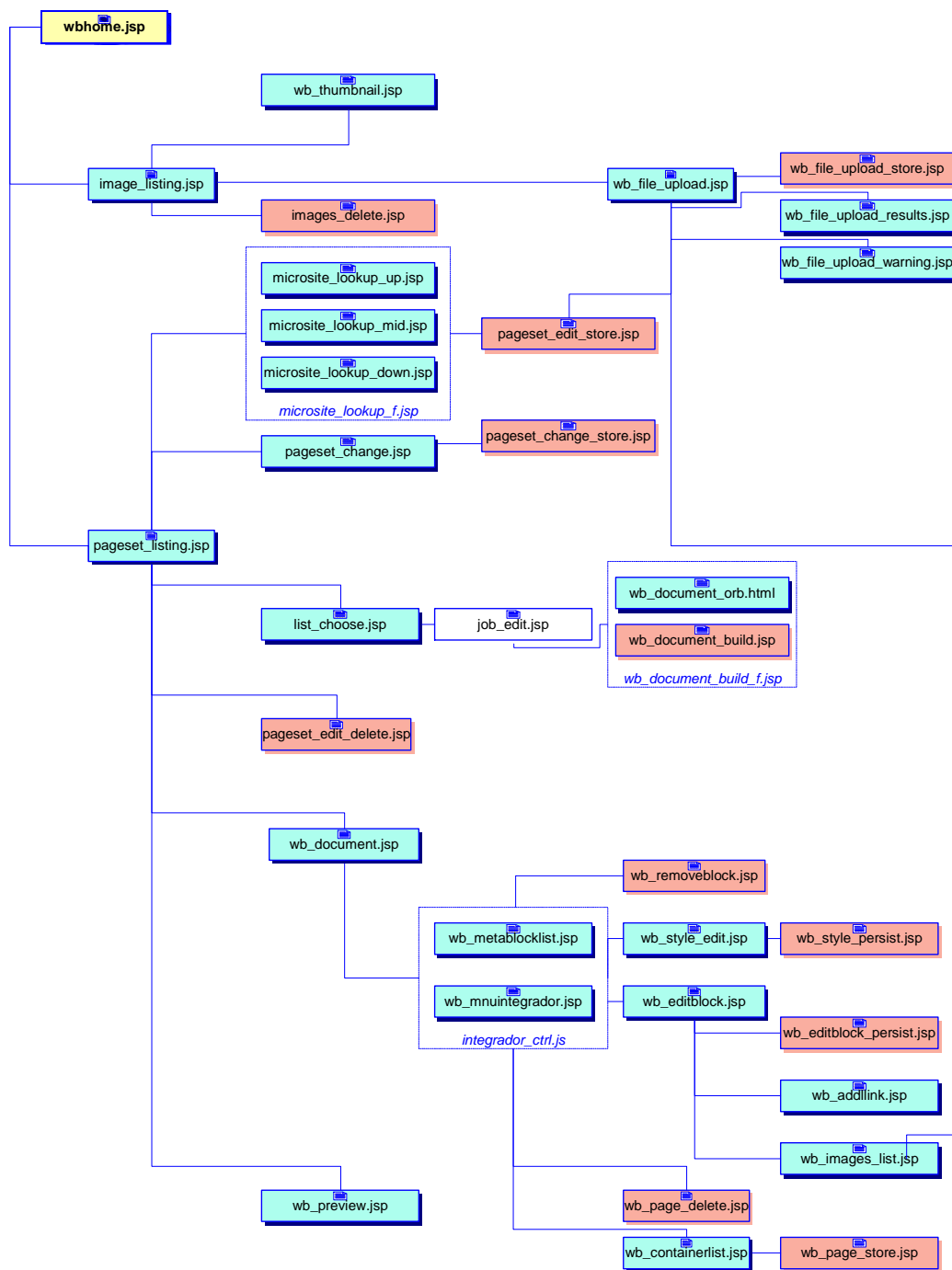
Dentro del código JSP existen dos variables que controlan aspectos del formato de salida:

ENCODING	Indica cómo se codificará el texto de los tags <code><description></code> . Puede ser:
ENCODE_NONE	El texto se vuelca en XML tal cual salga de la base de datos.
ENCODE_HTML	El texto se pasa por la función <code>Gadgets.HTMLEncode()</code> para convertir los caracteres no ASCII-7 en entidades HTML.
ENCODE_CDATA	El texto se encierra en tags <code><![CDATA[...]]></code>
	En valor por defecto es <code>ENCODE_HTML</code> .
MAX_MSG_DESC_LEN	Longitud máxima del texto dentro de los tags <code><description></code> . El valor por defecto es de 200 caracteres.

Módulo de Gestión de Relaciones con Clientes



Módulo de WebBuilding



Borrado diferido de archivos

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

Cuando se borra un PageSet no siempre es posible eliminar inmediatamente sus archivos asociados. En algunos casos, el servidor web puede dejar bloqueados los archivos hasta que se reinicia. Para solventar este inconveniente la página `pageset_edit_store.jsp` genera una lista de archivos pendientes de borrar en el fichero `shell/cleanup.txt`. Una posible opción es hacer un script que borra los archivos pendientes cada vez que se reinicia el servidor.

8

JavaScripts

Librerías de terceros incluídas en el producto

DynAPI	http://dynapi.sourceforge.net/dynapi/
CKEditor	http://www.ckeditor.net/
HTMLArea	http://www.interactivetools.com/

Convenciones

Decorados y Hoja de Estilo (CSS)

El decorado de la aplicación se configura con las imágenes y la hoja de estilo `styles.css` de los subdirectorios `/web/skins/...`

El decorado por defecto es el del subdirectorio `/xp`.

El decorado activo se mantiene en la cookie `skin` y se establece por primera vez en la página `/common/login_chk.jsp`.

Para cargar los decorados por JavaScript basta con incluir los tags:

```
<SCRIPT LANGUAGE="JavaScript" SRC="/javascript/cookies.js"></SCRIPT>  
<SCRIPT LANGUAGE="JavaScript" SRC="/javascript/setskin.js"></SCRIPT>
```

Fechas

Por convenio las fechas cortas de todos los formularios se escriben en formato AAAA-MM-DD independientemente del idioma.

Calendario

Existe un calendario común en la página `/common/calendar.jsp`.

La función JavaScript estándar para llamar al calendario es:

```
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.getFullYear()) +  
        "&m=" + dtnw.getMonth() + "&c=" + ctrl, "",  
        "toolbar=no,directories=no,menubar=no,resizable=no,width=171,  
        height=195"); }  
}
```

Donde:

ctrl: Objeto de tipo `<INPUT>` de HTML donde se colocará la fecha seleccionada.

Menús



Los menús superiores de los formularios de edición están creados con Likno AllWebMenus. <http://www.likno.com>

Librerías JavaScript

Leer y Escribir Cookies

Ubicación: Archivo `/javascripts/cookies.js`

```
function getCookie (name)
```

Valor Retorno: Contenido de la cookie sin caracteres de escape o `null` si no se encuentra ninguna cookie con el nombre especificado.

```
function setCookie (name, value, expire)
```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

Establecer el valor de una cookie.

name	Nombre de la cookie a establecer.
value	Valor a establecer. Internamente se aplicará la función JavaScript <code>escape()</code> a la cadena de entrada.
expire	Opcional. Variable de tipo Date con la fecha de expiración absoluta para la cookie.

```
function deleteCookie (name)
```

Elimina una cookie haciéndola expirar.

Manipulación de ComboBoxes

Ubicación: Archivo /javascripts/combobox.js

```
function setCombo (objCombo, idValue)
```

Mueve la selección de una ComboBox al valor especificado.

objCombo	Objeto HTML <SELECT>.
----------	-----------------------

idValue	Valor a buscar.
---------	-----------------

```
function comboIndexOf (objCombo, idValue)
```

Devuelve el índice de un valor en una ComboBox

objCombo	Objeto HTML <SELECT>.
----------	-----------------------

idValue	Valor a buscar.
---------	-----------------

Valor Retorno: Índice [0..objCombo.options.length-1] ó -1 si el valor buscado no se encontró en la ComboBox.

```
function comboPush (objCombo,txValue,idValue,defSel,curSel)
```

Añade una opción a una ComboBox.

objCombo	Objeto HTML <SELECT>.
txValue	Texto de la opción.
idValue	Valor de la opción.
defSel	true si la opción estará seleccionada por defecto.
curSel	true si la opción debe convertirse en la selección actual.

```
function getCombo (objCombo)
```

Devuelve el valor seleccionado en una ComboBox.

objCombo	Objeto HTML <SELECT>.
----------	-----------------------

Valor Retorno: Contenido del atributo VALUE de la opción seleccionada o **null** si no hay ninguna opción seleccionada.

```
function getComboText (objCombo)
```

Devuelve el texto seleccionado en una ComboBox.

objCombo	Objeto HTML <SELECT>.
----------	-----------------------

Valor Retorno: Contenido del texto de la opción seleccionada o **null** si no hay ninguna opción seleccionada.

```
function clearCombo (objCombo)
```

Elimina todas las opciones de una ComboBox.

objCombo Objeto HTML <SELECT>.

```
function sortCombo (objCombo)
```

Ordena de menor a mayor los textos de una ComboBox.

Validación de fechas

Ubicación: Archivo /javascripts/datefuncs.js

```
function getLastDay (month, year)
```

Devuelve el ultimo día del mes. Tiene en cuenta los años bisiestos.

month Mes [0..11].

year Año (4 dígitos).

```
function isDate (dtexpr, dtformat)
```

Verifica si una cadena tiene un formato de fecha predeterminado.

dtexpr Cadena a verificar.

dtformat Código del Formato. Actualmente sólo se implementa la verificación para el formato “d” que acepta fechas cortas con el patrón “YYYY-MM-DD”. Se comprueba tanto la sintaxis de la cadena como el mes [1..12] y el último día del mes [28..31] teniendo en cuenta años bisiestos.

Valor Retorno: **true** si la cadena de entrada representa una fecha válida. **false** en caso contrario.

```
function parseDate (dtexpr, dtformat)
```

Devuelve un objeto de tipo Date a partir de un String.

dtexpr Cadena con la fecha.

dtformat Código del Formato. Actualmente se aceptan los siguientes: formatos
"d" fechas cortas con el patrón "YYYY-MM-DD"
(el rango de mes es de 1 a 12).
"s" fechas cortas con el patrón "DD/MM/YYYY"
"ts" fecha y hora con patrón "YYYY-MM-DD hh24:mm:ss"

Valor Retorno: Objeto de tipo Date o **null** si dtexpr no es una fecha válida.

```
function dateToString (dtexpr, dtformat)
```

Convierte una fecha a formato cadena.

dtexpr Cadena con la fecha.

dtformat Código del Formato. Actualmente se aceptan los siguientes: formatos
"d" fechas cortas con el patrón "YYYY-MM-DD"
(el rango de mes es de 1 a 12).
"s" fechas cortas con el patrón "DD/MM/YYYY"
"ts" fecha y hora con patrón "YYYY-MM-DD hh24:mm:ss"

Valor Retorno: Objeto de tipo String con el patrón de fecha especificado.

```
function daysDiff (dt1, dt2)
```

Halla el número entero de días entre dos fechas.

Valor Retorno: Objeto de tipo entero con los días desde dt1 dt2.

```
function addHours (dt1, hrs)
```

Suma una determinada cantidad de horas a una fecha.

Valor Retorno: Objeto de tipo Date con la nueva fecha.

Validación de direcciones de e-mail

Ubicación: Archivo /javascripts/email.js

```
function check_email (email)
```

Verifica si una dirección de e-mail es sintácticamente correcta.

Búsqueda de subcadenas dentro de una página

Ubicación: Archivo /javascripts/findit.js

```
function findit (sValue)
```

Busca una subcadena dentro de la página actual.

Obtención de parámetros de la URL

Ubicación: Archivo /javascripts/getparam.js

```
function getURLParam (name, target)
```

Obtiene un parámetro de la URL.

name Nombre del parámetro.

target Opcional. Objeto de tipo window en cuya URL se buscará el parámetro.

Valor Retorno: Valor del parámetro en la URL o `null` si no se encontró ningún parámetro con el nombre especificado.

Manipulado de cadenas

Ubicación: Archivo /javascripts/trim.js

```
function ltrim (str)
```

Elimina los espacios en blanco por la izquierda.

```
function rtrim (str)
```

Elimina los espacios en blanco por la derecha.

Validación de Documentos de Identidad

Ubicación: Archivo /javascripts/simplevalidations.js

```
function validarDocumento (documento, tipodocumento)
```

Valida un DNI o NIF.

documento Número de documento a validar. En formato A12345678 para NIFs y 12345678^a para DNIs

tipodocumento N para NIFs ó D para DNIs.

Validaciones básicas

Ubicación: Archivo /javascripts/simplevalidations.js

```
function hasForbiddenChars (str)
```

Esta función se utiliza principalmente para validar aquellos campos que no deban contener algunos caracteres especiales, como comillas, asteriscos, etc.

Valor Retorno: **true** si la cadena de entrada contiene alguno de los siguientes caracteres { ' (comilla simple), " (comilla doble), | (barra vertical), * (asterisco), ¿ (abrir interrogación), ? (cerrar interrogación), & (ampersand), ; (punto y coma), ` (acento grave), / (barra del siete), \ (antibarra) }, **false** en caso contrario.

```
function isIntValue (expr)
```

Valor Retorno: **true** si la expresión de entrada es convertible a un número entero con o sin signo, **false** en caso contrario.

```
function isFloatValue (expr)
```

Valor Retorno: **true** si la expresión de entrada es convertible a un número en punto flotante con o sin signo, **false** en caso contrario.

Validación de Cuentas Bancarias

Ubicación: Archivo /javascripts/simplevalidations.js

```
function isBankAccount (entity,office,dc,cc)
```

Verifica los dígitos de control de una cuenta bancaria.

Valor Retorno: **true** si los dígitos calculados coinciden con el parámetro dc, **false** en caso contrario.

Validación de Tarjetas de Crédito

Ubicación: Archivo /javascripts/creditcards.js

Eric Krock

(c) 1997 Netscape Communications Corp.

Creación de la base de datos

9

La base de datos de hipergate puede crearse de dos formas: 1ª) es posible cargar directamente un archivo de exportación en el formato nativo del SGBDR generado previamente; y 2ª) es posible utilizar la clase `com.knowgate.hipergate.datamodel.ModelManager` que tiene rutinas para crear desde cero la base de datos utilizando exclusivamente script SQL transportables y código Java con conexión JDBC.

`ModelManager` está especialmente pensado para ser invocado por línea de comandos, aunque también posee un interfaz accesible desde Java.

Pasos en la creación de la base de datos

Crear una base de datos vacía supone básicamente 4 cosas :

- 1º) Crear todas las tablas, vistas y procedimientos almacenados.
- 2º) Cargar los datos de los dominios SYSTEM y MODEL, imprescindibles para arrancar la aplicación.
- 3º) Cargar los dominios adicionales (típicamente TEST, DEMO y REAL).

Scripts SQL transportables

hipergate dispone de un conjunto de scripts SQL para la creación de objetos en el SGBDR.

Estos scripts pueden encontrarse descomprimidos en el directorio `com/knowgate/hipergate/datamodel` del paquete de fuentes o en la misma ruta dentro de `hipergate.jar`.

Los scripts pueden tener extensión SQL o DDL. La única diferencia entre ambas extensiones es el delimitador de comando utilizado. Para los archivos SQL cada sentencia SQL está separada de la siguiente por un punto y coma. Para los archivos DDL se utiliza una antebarra (backslash). La clase `ModelManager` especialmente preparada para procesar estos delimitadores.

Los scripts están divididos según su tipo :

- tables
- indexes
- constraints
- views
- data
- procedures
- triggers
- drop

Los scripts de procedures, triggers, vistas y drop son dependientes del SGBDR, los scripts tables, indexes, constraints y data son independientes del SGBDR.

Para conseguir independencia del SGBDR en la definición de tipos, hipergate usa sólo un número restringido de [tipos](#) y un nombre especial para cada tipo que es traducido por `ModelManager` al nombre nativo del SGBDR en el momento de lanzar el script.

Símbolo hipergate	Oracle	MSSQL	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

División modular de los scripts SQL/DDL

Aparte de estar divididos según el tipo de sentencias que contengan y el SGBDR para el que estén escritos, los scripts están divididos por módulos funcionales. Esto facilita agregar nuevas partes a la base de datos o modificar algunas ya existentes sin alterar otras.

Comandos para crear y borrar la base de datos

Crear una base de datos por defecto

Desde la línea de comandos escribir :

```
java com.knowgate.hipergate.datamodel.ModelManager  
/etc/hipergate.cnf create database verbose
```

Esto creará todas las tablas para todos los módulos y creará los dominios SYSTEM, MODEL, TEST, DEMO y REAL.

Crear una base de datos mínima

Desde la línea de comandos escribir :

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

Esto creará todas las tablas para todos los módulos y creará los dominios SYSTEM, MODEL.

Borrar una base de datos

Desde la línea de comandos escribir :

```
java com.knowgate.hipergate.datamodel.ModelManager  
/etc/hipergate.cnf drop all verbose
```

Esto borrará todas las tablas, vistas, índices y procedimientos.

Cómo ejecutar un script SQL contra la base de datos

Desde la línea de comandos escribir :

```
java com.knowgate.hipergate.datamodel.ModelManager  
/etc/hipergate.cnf execute /tmp/script.sql verbose
```

Cómo generar un script SQL a partir de los datos de una tabla

hipergate dispone de una utilidad para generar un script de comandos SQL de inserción a para los datos de una tabla precargada.

Desde la línea de comandos escribir :

```
java com.knowgate.hipergate.datamodel.ModelManager  
/etc/hipergate.cnf script nombre_tabla /tmp/salida.sql
```

Puede hacerse también desde una página JSP del estilo:

```
<%@ 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",  
"nombre_tabla", "/tmp/salida.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>Volcado a script SQL</title>  
  </head>  
  <body>  
    Volcado finalizado con éxito.  
  </body>  
</html>
```

Cómo cargar un archivo en una tabla desde línea de comandos

Es posible usar la clase Java

`com.knowgate.hipergate.datamodel.TableLoader` para cargar un archivo de texto delimitado en una tabla.

TableLoader es rápido y sencillo, pero tiene algunas limitaciones:

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

1. Las columnas del archive deben estar delimitadas por tabuladores.
2. Cada registro debe estar delimitado por un fin de línea.
3. El número de columnas en el archivo de entrada debe coincidir exactamente con el número de columnas en la tabla.
4. Las fecha deben estar en formato yyyy-MM-dd HH:mm:ss
5. Los número decimales deben usar el punto como separador decimal.

Es posible invocar a TableLoader desde el método `main()` de la clase `com.knowgate.hipergate.datamodel.ModelManager` que toma los siguientes parámetros de entrada:

- **ruta al archivo de propiedades:** Usualmente `/etc/hipergate.cnf` o `C:\Windows\hipergate.cnf`
- **comando:** para cargar un archivo el commando debe ser `bulkload`
- **table de destino:** nombre de la tabla donde sera cargada la información.
- **juego de caracteres:** debe ser cualquiera de los recogidos en [Java Supported Character Encodings](#).
- **verbose:** Opcional. Si se especifica el parámetro `verbose` se mostrará información adicional de progreso en la salida estándar del sistema.

Ejemplo de uso (para 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
```

☞ Se debe añadir también la referencia al .jar del driver JDBC para el SGBDR que se esté utilizando.

☞ Ver también [carga de datos desde archivos de texto](#) para cargas de datos más complejas sobre varias tablas simultáneamente.

Integración con Jakarta Lucene



10

Lucene es el indexador open source escrito en Java para el proyecto Jakarta de Apache Software Foundation.

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

Lucene es un indexador multi-plataforma bastante rápido y de propósito general.

<http://jakarta.apache.org/lucene/docs/index.html>

Interfaz de indexación Lucene - hipergate

En la clase `com.knowgate.lucene.Indexer` permite cargar índices full-text en Lucene desde las tablas `k_bugs`, `k_newsgroups` y `k_mime_msgs`.

La clave de la técnica estriba en indexar una serie de campos de la base de datos y ser capaz de recuperar el GUID de cada registro cuyo texto cumple con la condición de búsqueda.

Los índices de Lucene se organizan por áreas de trabajo y por tabla base. Es decir, existes tres índices para cada área de trabajo, uno para los bugs, otro para los mensajes de los foros y otro para los e-mails.

La indexación con Lucene se activa automáticamente especificando dos propiedades en el archivo `hipergate.cnf`: `luceneindex` y `analyzer`.

luceneindex Esta propiedad es el directorio base que se utilizará para almacenar los archivos de Lucene (por ejemplo `/opt/knowgate/storage/lucene/`) Bajo este directorio se crearán otros dos, uno con el nombre de la tabla base y, por debajo de este, otro para cada Área de Trabajo.

analyzer Esta propiedad es opcional. Es el nombre de la clase que se utilizará para el analizador de Lucene. Por defecto es `org.apache.lucene.analysis.SimpleAnalyzer`.

Estructura de los documentos indexados

Los documentos de los índices contienen un conjunto de campos comunes

Campo	Tipo	Comentarios
workarea	Keyword	GUID del área de trabajo. Normalmente es el mismo valor para todos los documentos de un índice.
container	Keyword	Nombre del proyecto, nombre de la carpeta o nombre del foro según se trate de un Bug, un e-Mail o un mensaje.
guid	Keyword	GUID del registro en la base de datos correspondiente al documento.

number	Keyword	Número de bug u ordinal de e-mail en la carpeta. No es aplicable a mensajes de los foros.
created	Keyword/Fecha	Fecha de modificación o de envío si es un e-mail.
size	Keyword	Tamaño en bytes (sólo aplicable a los e-mails)
title	Text	Título o Asunto.
autor	Text	Nombre del autor o remitente.
abstract	Text	Resumen
recipients	Text	Nombres de los destinatarios (sólo e-mails)
comments	UnStored	Comentarios
text	UnStored	Texto del documento

Véase el API JavaDoc para mayor información.

11

Integración con Jakarta POI

Jakarta POI es el interfaz Java de acceso a archivos OLE2 de Apache Software Foundation. POI permite leer y escribir documentos compuestos OLE2 desde código 100% puro Java.



Un interfaz muy sencillo para leer y escribir propiedades de documentos OLE2 puede encontrarse en la clase `com.knowgate.ole.OLEDocument`. Esta clase se utiliza, en particular, desde la página `docedit_store.jsp` para rellenar automáticamente los campos de la tabla `k_prod_attr` para documentos introducidos en la Biblioteca Corporativa que contengan hojas de propiedades OLE2.

12

Integración con LDAP

hipergate puede almacenar las claves de acceso de los usuarios en un directorio LDAP. También es posible leer los contactos desde un cliente de correo cuya agenda permita explorar un directorio LDAP.

Cómo crear un directorio compatible con hipergate

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

La estructura LDAP que exporta hipergate trata de ser lo más sencilla posible, con el principal objetivo de permitir la conexión de clientes de correo como Outlook Express, Mozilla Mail o Ximian Evolution. Esta estructura sirve como base para adaptaciones a Servicios de Directorio Corporativos como Active Directory, OpenLDAP, Novell NDS, etc.

El sistema de comunicación con LDAP ha sido desarrollado con el Novell Directory Server SDK (<http://developer.novell.com/ndk/>), de distribución gratuita. Este API soporta todas las llamadas estándar de servicios de directorio LDAPv3.

Esta parte de hipergate ha sido desarrollada y testada con la distribución estándar de OpenLDAP 2.1. Una configuración por defecto será suficiente para un entorno controlado con reglas de seguridad poco estrictas.

Ejemplo rápido de configuración con OpenLDAP

- 1) Instalar el servidor OpenLDAP 2.1 o superior. Puede utilizarse el código fuente, paquetes RPM o DEB, PKG para Solaris...
- 2) Añadir o modificar los siguientes parámetros en el fichero slapd.conf (ubicado en el directorio etc/ ó /etc/openldap)

```
suffix          "dc=hipergate,dc=org"
rootdn          "cn=Manager,dc=hipergate,dc=org"

# Cleartext passwords, especially for the rootdn, should
# be avoided. See 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) Iniciar el servicio OpenLDAP. Comprobar que el equipo está escuchando por el puerto 389. Para ello se puede utilizar, por ejemplo, el comando netstat

```
$ netstat -na | fgrep ":389"
tcp      0      0  0.0.0.0:389          0.0.0.0:*           LISTEN
```

- 4) Crear el siguiente fichero de texto en un editor y guardar con el nombre de fichero 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

```

NOTA: En este fichero hay que tener cuidado con el final de las líneas (sin espacios en blanco al final) y las líneas vacías.

- 5) Cargar en LDAP el fichero que acabamos de crear:

```
ldapadd -x -D "cn=Manager,dc=hipergate,dc=org" -W -f init.ldif
```

Debemos introducir la contraseña del parámetro “rootpw” del fichero slapd.conf

## Qué información se almacena en LDAP

El modelo de datos relacional de hipergate almacena información sobre individuos, usuarios y empleados (Directorio de Personal). En la exportación a LDAP se ha reflejado la siguiente información:

- Usuarios (Users, obtenidos de la tabla k\_users)
- Empleados (Employees, obtenidos de la tabla k\_fellows)
- Contactos (Contactos, obtenidos de la tabla k\_member\_address)

Estos tres tipos de objeto contienen las mismas propiedades, y únicamente se diferencian por su posición en el árbol y si disponen o no de contraseña para identificarse.

La información se almacena en LDAP según la siguiente estructura:

```

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

```

Cada elemento **dc** es un contenedor. Algunos tienen un nombre fijo, impuesto desde la librería de acceso a LDAP de hipergate (org, hipergate,

users, privateContacts, publicContacts y employees). En el caso de los campos resaltados en negrita, reflejan los campos de la BB.DD. que se utilizan para rellenar dicha información.

La rama hipergate contiene una entrada por cada dominio de seguridad que existe en la tabla k\_domains. Cada dominio contiene a su vez todas las áreas de trabajo (*workareas*) configuradas.

Cada workarea dentro de LDAP alberga tres contenedores bien diferenciados:

- users: Usuarios de la aplicación, miembros de la tabla k\_users con contraseña pero sin información de Dirección Postal ni teléfono.
- publicContacts: Contactos públicos, es decir, aquellas entradas de k\_member\_address cuyo flag bo\_private sea cero (*false*). Disponen de información de Dirección Postal y teléfonos.
- employees: Miembros de la tabla k\_fellows. Su Dirección Postal es en realidad la conjunción de Departamento, División y Ubicación.

Cada entrada en la rama users puede albergar a su vez un subcontenedor privateContacts, en el que se almacenarán las entradas de la k\_member\_address cuyo flag bo\_active sea distinto de cero (*true*). Para saber a qué contacto de hipergate pertenece la entrada en LDAP se utiliza el email principal (tx\_main\_email).

Los objetos de tipo “Persona+Dirección” que crea hipergate son el formato más sencillo de LDAP, compatibles con Outlook Express, WAB (Windows Address Book), Mozilla y Ximian Evolution. Muchos de los campos del modelo de datos de hipergate podrían aparecer en esta lista de propiedades, pero el concepto de diseño es mantener al mínimo en número de extensiones LDAP que hay que utilizar por defecto. Añadir, por ejemplo, información referente a URLs o Cuentas Bancarias implica realizar extensiones al Modelo de Datos del Servicio LDAP (*schema*).

Estos objetos se basan en los Tipos de Objeto LDAP (*objectClass*) inetOrgPerson y organizationalPerson. Los campos cargados por defecto son:

|           |                                            |
|-----------|--------------------------------------------|
| <b>cn</b> | <b>Common Name</b>                         |
|           | Dirección principal de e-mail              |
|           | <b>Users:</b> k_users.tx_main_email        |
|           | <b>Employees:</b> k_fellows.tx_email       |
|           | <b>Contacts:</b> k_member_address.tx_email |

|            |                  |
|------------|------------------|
| <b>uid</b> | <b>Unique ID</b> |
|------------|------------------|

Identificador único dentro de la BB.DD. (GUID)

**Users:** k\_users.gu\_user  
**Employees:** k\_fellows.gu\_fellow  
**Contacts:** k\_member\_address.gu\_address

## givenName

Nombre de pila

**Users:** k\_users.nm\_user  
**Employees:** k\_fellows.tx\_name  
**Contacts:** k\_member\_address.tx\_name

## sn

Surname

Apellidos

**Users:** (k\_users.tx\_surname1 + ' ' +  
k\_users.tx\_surname2), o bien  
k\_users.tx\_nickname  
**Employees:** k\_fellows.tx\_name  
**Contacts:** k\_member\_address.tx\_name

## userPassword

Contraseña de acceso (solo Usuarios)

**Users:** k\_users.pwd

## displayName

Se utiliza en el campo "Display" de 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

Empresa a la que pertenece el Contacto

**Users:** k\_users.nm\_company  
**Employees:** k\_fellows.tx\_company  
**Contacts:** k\_member\_address.nm\_legal

## telephonenumber

Número de teléfono principal. No disponible en k\_users.

**Employees:** k\_fellows.work\_phone  
**Contacts:** k\_member\_address.work\_phone

### homePhone

Número de teléfono particular. No disponible en k\_users.

**Employees:** k\_fellows.home\_phone

**Contacts:** k\_member\_address.home\_phone

### mobile

Número de teléfono móvil. No disponible en k\_users.

**Employees:** k\_fellows.mov\_phone

**Contacts:** k\_member\_address.mov\_phone

### facsimileTelephoneNumber

Número de fax. Solo disponible en k\_member\_address.

**Contacts:** k\_member\_address.fax\_phone

### postalAddress

Dirección Postal. Los retornos de carro se codifican con un *pipe* (ASCII 166), según la nomenclatura de Outlook Express/WAB.

No disponible en k\_users.

**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

Ciudad de la Dirección Postal. Solo disponible en k\_member\_address.

**Contacts:** k\_member\_address.nm\_city

### st

#### State

Provincia/Estado de la Dirección Postal. Solo disponible en k\_member\_address.

**Contacts:** k\_member\_address.nm\_state, ó bien  
k\_member\_address.id\_state

### postalCode

Código Postal de la Dirección Postal. Solo disponible en k\_member\_address.

**Contacts:** k\_member\_address.zipcode

## Cómo conectar hipergate con el directorio LDAP

Una vez que se ha creado el directorio, es preciso especificar los siguientes parámetros en hipergate.cnf para conectar hipergate con el servidor LDAP:

**ldapconnect** : URL de acceso al directorio.  
Debe ser de la forma:  
`ldap://192.168.1.1:389/dc=hipergate,dc=org`

**ldapuser** : `cn=Manager,dc=hipergate,dc=org`

**ldappassword** : `manager`

**ldapclass** : Clase Java que implementa el interfaz `com.knowgate.ldap.LDAPModel`. Por defecto es `com.knowgate.ldap.LDAPNovell` pero puede escribirse otra clase alternativa.

## Sincronización entre LDAP e hipergate

Cuando se activa la conexión con LDAP, hipergate sincroniza automáticamente los usuarios y los contactos con el directorio LDAP. Cuando se añade o modifica un usuario o un contacto en hipergate los cambios se reflejan en LDAP. Igualmente cuando se borra un usuario o un contacto de hipergate se borra de LDAP.

La sincronización es unidireccional desde hipergate hacia LDAP. Las modificaciones que se realicen directamente sobre LDAP no se reflejan en la base de datos de hipergate.

La sincronización se lleva a cabo en las páginas: `addr_edit_store.jsp`, `addr_edit_delete.jsp`, `contact_new_store.jsp`, `usernew_store.jsp`, `useredit_modify.jsp`, `fellow_edit_store.jsp`, `fellow_edit_delete.jsp`

Si la base de datos de hipergate y el directorio se des-sincronizan es posible agregar o eliminar entradas individuales con los métodos `addOrReplaceAddress()`, `addOrReplaceUser()`, `deleteAddress()` y `deleteUser()`.

## Cómo cargar un dominio o un área de trabajo en LDAP

El interfaz LDAPModel tiene dos métodos para cargar un Dominio o un Área de Trabajo completos en LDAP.

`LDAPModel.loadDomain (Connection oJdbc, int iDomainId)`

Carga todos los usuarios, contactos y empleados de un dominio de hipergate en LDAP.

El primer parámetro es una conexión JDBC a la base de datos de hipergate.

El segundo parámetro es el identificador numérico del dominio a cargar (campo `k_domains.id_domain`).

`LDAPModel.loadWorkArea (Connection oJdbc, String sDomainNm, String sWorkAreaNm)`

Carga todos los usuarios, contactos y empleados de un área de trabajo de hipergate en LDAP.

El primer parámetro es una conexión JDBC a la base de datos de hipergate.

El segundo parámetro es el nombre del dominio a cargar (campo `k_domains.nm_domain`).

El tercer parámetro es el nombre del área de trabajo a cargar (campo `k_workareas.nm_workarea`).

## Cómo borrar un área de trabajo

El método `LDAPModel.deleteWorkArea()` sirve para eliminar todas las entradas de hipergate pertenecientes a un Área de Trabajo en un directorio LDAP.

## Cómo borrar un directorio completo

El método `LDAPModel.dropAll()` sirve para eliminar todas las entradas de hipergate en un directorio LDAP.

## Cómo acceder al directorio desde Outlook Express

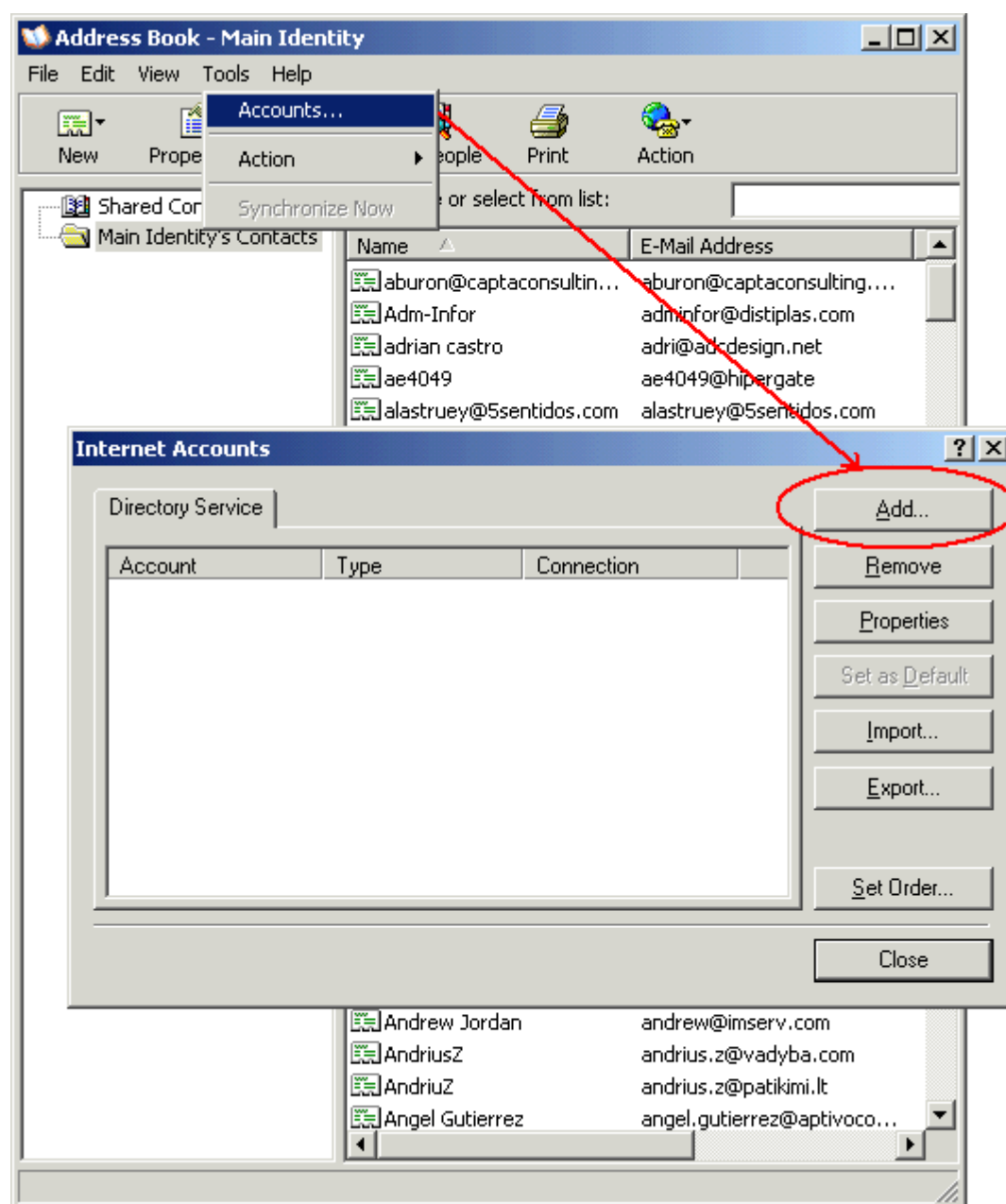
La configuración de un cliente LDAP (Outlook Express, Evolution, Mozilla, etc) depende más de cómo esté configurado el servidor LDAP que de la exportación de datos de hipergate.

En primer lugar de deben configurar los usuarios y contraseñas que pueden autenticarse en el dominio y realizar búsquedas. La instalación de OpenLDAP por defecto solo permite realizar autenticaciones (*bind*) a

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

usuarios no autenticados. Una vez el usuario se ha impersonado las reglas de seguridad suelen ser muy laxas (se realizan búsquedas en todo el directorio, sin reparar qué usuario es dueño de qué rama)

## Capturas de pantalla de Outlook Express





**Internet Connection Wizard**

**Internet Directory Server Name**

Type the name of the Internet directory (LDAP) server your Internet service provider or system administrator has given you.

Internet directory (LDAP) server:

If your Internet service provider or system administrator has informed you that they require you to log on to your LDAP server and has provided you with an LDAP account name and password, select the check box below.

☐ My LDAP server requires me to log on

< Back   Next >   Cancel

**Internet Connection Wizard**

**Check E-mail Addresses**

Your e-mail program checks the e-mail addresses of your message recipients using one or more directory service address lists.

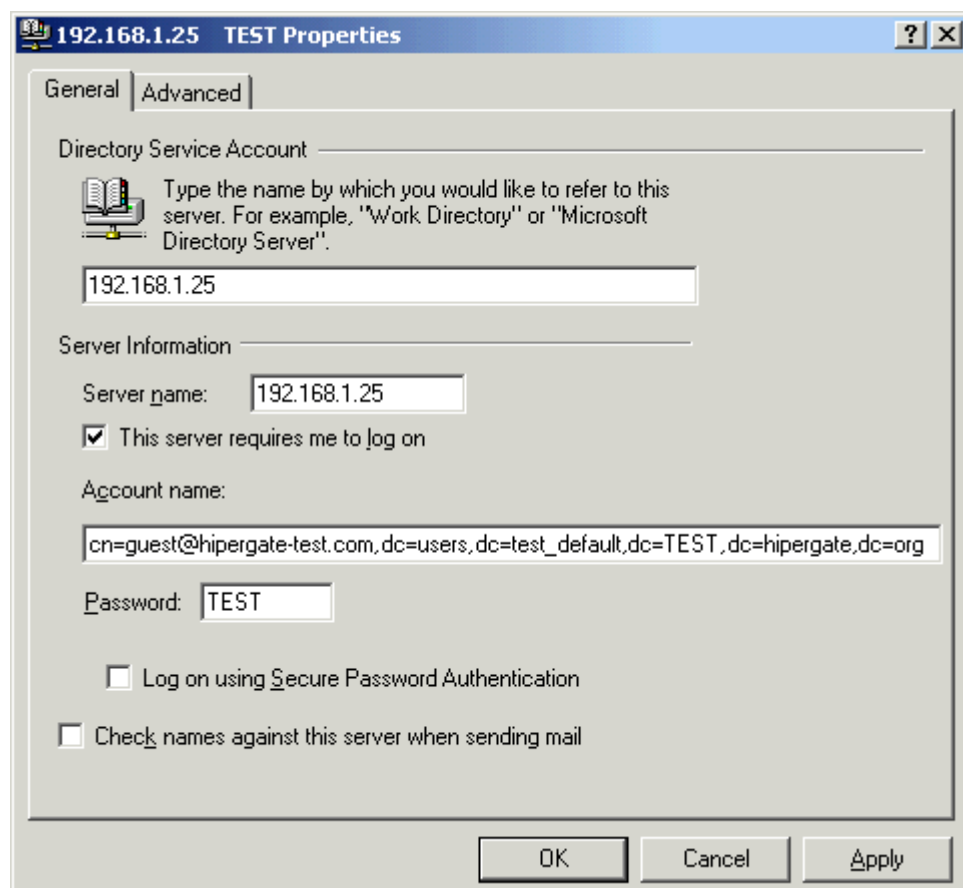
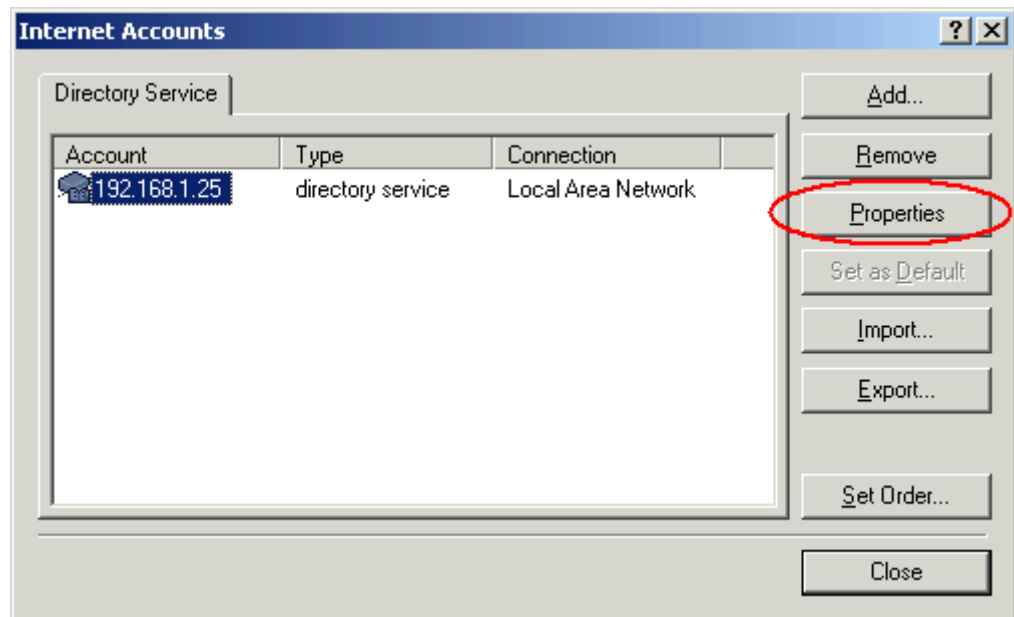
Using a directory service to check the e-mail addresses of your message recipients may slow down the performance of your e-mail program.

Do you want to check addresses using this directory service?

☐ Yes

☒ No

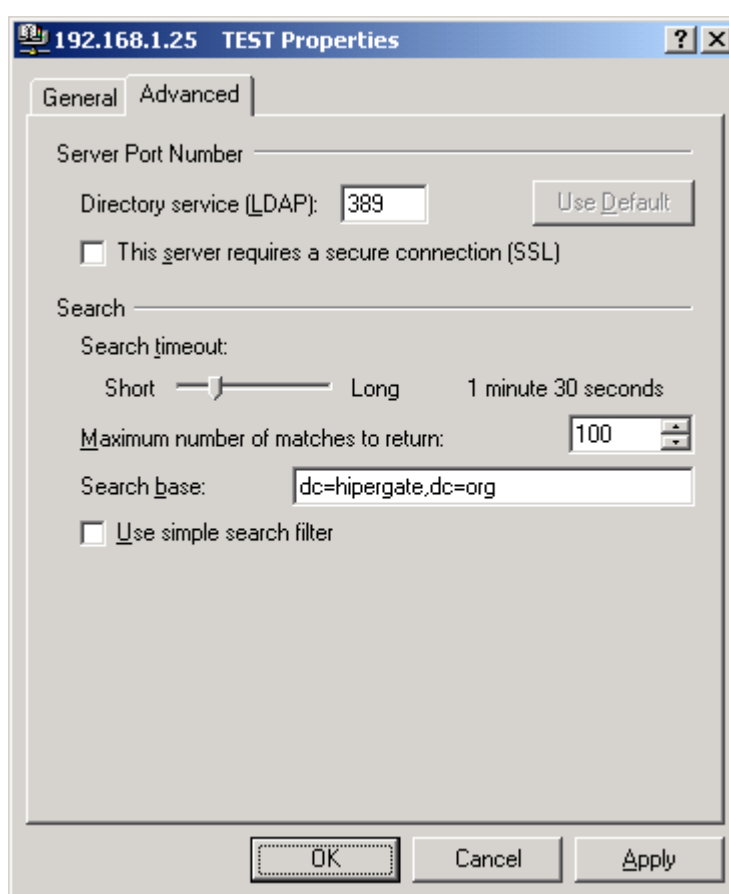
< Back   Next >   Cancel

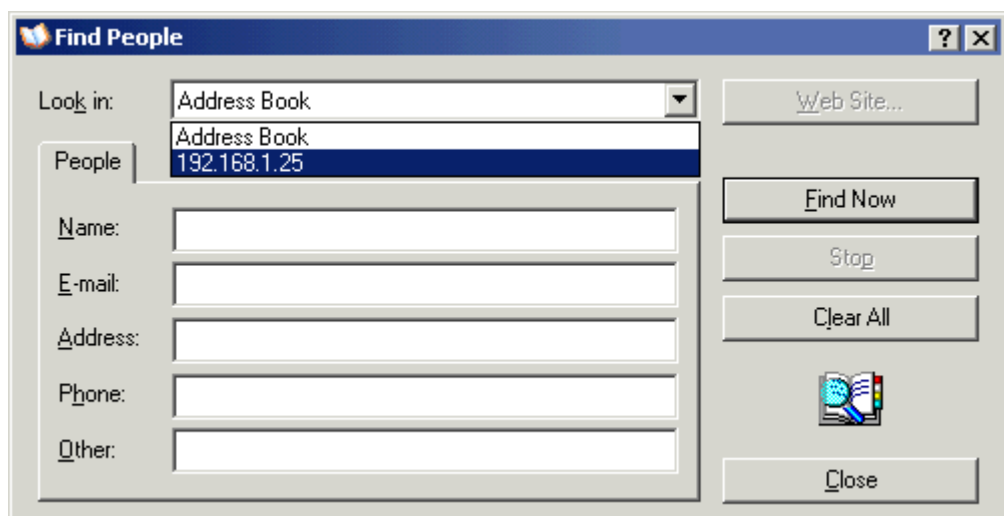
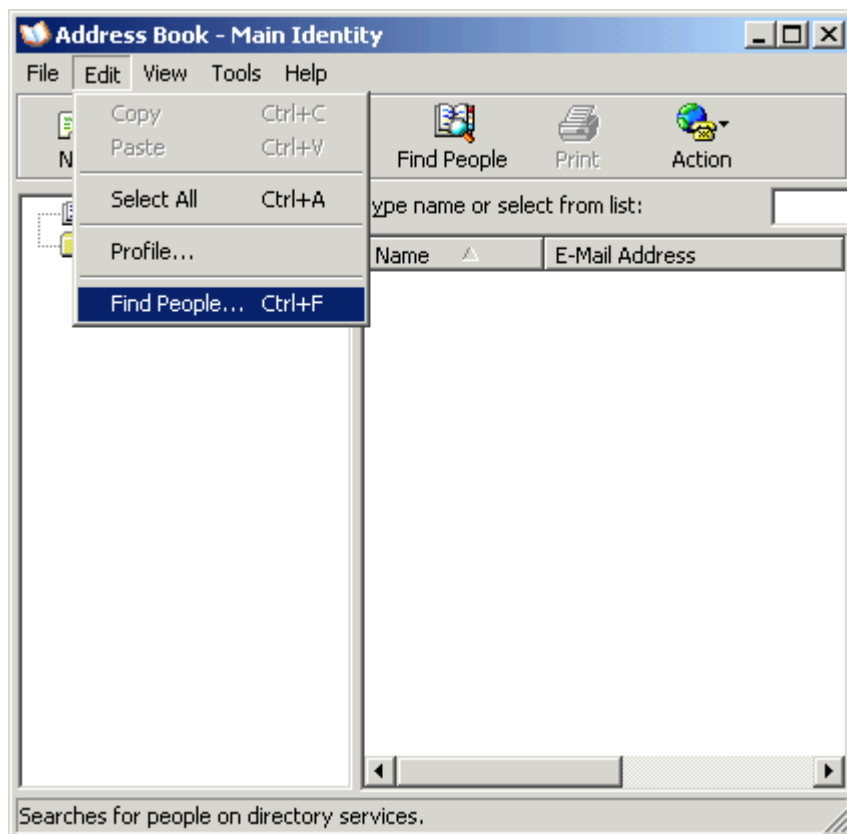


En la ficha de configuración del Servicio de Directorio de nuestro cliente de correo debemos especificar el nombre del host y el puerto de acceso al servicio (por defecto 389). Además, se debe utilizar un usuario y contraseña válido en el servicio. Al cargar desde hipergate, todas las entradas de la tabla k\_users contienen el campo userPassword, lo que les permite autenticarse contra el servicio de directorio. Así pues, cualquier miembro de la tabla k\_users es un usuario válido en LDAP.

Para especificar un usuario y una contraseña, se debe añadir la ruta completa de LDAP (*Distinguished Name*) al nombre de usuario. Por ejemplo:

cn=user@hipergate.org,dc=users,dc=workarea1,dc=domain1,dc=hipergate,dc=org





Mediante reglas de reescritura es posible hacer que OpenLDAP busque la dirección de email como login único en todos los dominios y áreas de trabajo. Con ello se consigue que solo haya que usar el email como

nombre de usuario en el acceso a LDAP. Para más información, consultar una respuesta de Buchan Milne en la lista de correo OpenLDAP-Software: <http://www.openldap.org/lists/openldap-software/200404/msg00910.html>

## Autenticación de usuarios basada en LDAP

Es posible configurar hipergate para que las contraseñas de usuario de acceso a la aplicación sean leídas de un directorio LDAP y no de la tabla `k_users` de la base de datos estándar.

LDAP tan sólo sirve para verificar contraseña pero no reemplaza la seguridad nativa, aunque se autentifique a los usuarios contra LDAP, es preciso que estos sigan existiendo en la base de datos de hipergate. Además, el modelo de seguridad basado en roles de hipergate se sigue manteniendo la base de datos relacional aunque se use LDAP para validar las contraseñas.

## Cómo autenticar a los usuarios usando LDAP

Es necesario llevar a cabo los siguientes pasos:

1. Tener creado el esquema LDAP. La comprobación de contraseña se realiza contra la entrada `userPassword` de LDAP en `cn=user@domain.com,dc=users,dc=workarea_name,dc=domain_name,dc=hipergate,dc=org`. El método de verificación de contraseña consiste en llamar al método `LDAPConnection.bind()` contra la entrada anterior usando la contraseña suministrada por el usuario. En LDAP debe haber una entrada cuyo common name sea el e-mail del usuario a autenticar bajo `dc=users...` esta entrada se puede crear automáticamente si se activa la sincronización con LDAP.
2. Configurar los parámetros `ldapconnect`, `ldapuser`, `ldappassword` y `ldapclass` de `hipergate.cnf` como se describe anteriormente en este capítulo.
3. Asignar la propiedad `authmethod=ldap` en `hipergate.cnf`.

La conexión con LDAP sólo se realiza durante el proceso de login inicial a la aplicación. La información de acceso se extrae de LDAP y a

partir de dicho punto el resto de las páginas validan cookies de sesión para verificar las credenciales de acceso.

# 13

## Integración de seguridad con NTLM

**Si se usa Microsoft Internet Explorer, hipergate puede autenticar usuarios directamente a través de sus credenciales de Windows sin necesidad de solicitar nuevamente la contraseña de acceso.**

El proceso de integración de seguridad con NTLM se realiza mediante un filtro de servidor que convierte las credenciales NTLM en cookies de sesión de hipergate.

Para que funcione la autenticación integrada con NTLM los campos `tx_nickname` y `tx_pwd` de la tabla `k_users` deben coincidir con el par usuario/clave utilizado para iniciar la sesión en Windows y el usuario debe pertenecer a un dominio que se llame igual en Windows que en hipergate.

☞ El nombre para un dominio de hipergate puede cambiarse conectándose como administrador del dominio SYSTEM (por defecto [administrador@hipergate-system.com](mailto:administrador@hipergate-system.com) / hipergate).

### Instalación de filtro de integración NTLM

El filtro de autenticación es la clase `com.knowgate.jcifs.http.NtlmHiperGateFilter` que se encuentra contenida dentro de `hipergate.jar`.

El filtro se configura instalando el siguiente fragmento de XML en la sección `<web-app>` del archivo `/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>

```

Hay dos parámetros que configurar:

**jcifs.http.domainController** : La dirección IP del controlador de dominio de Windows.

**jcifs.smb.client.logonShare** : El nombre de un directorio compartido en el controlador de dominio.

La página loginntlm.html sustituye a login.html. Cuando se entra por loginntlm.html el filtro obtiene las credenciales de NTLM y la página se redirige login\_chk.jsp redirige para validar el usuario y contraseña de la sesión de Windows contra la base de datos de hipergate.

## Configuración de hipergate.cnf

Una vez que se ha instalado el filtro se debe establecer la propiedad authmethod=ntlm en hipergate.cnf para activar la seguridad integrada con NTLM.

## Acceso remoto a la BB.DD. por HTTP

# 14

El servlet HttpDataObjsServlet del paquete com.knowgate.http proporciona acceso de lectura y escritura contra la base de datos de hipergate por HTTP POST.

## Funcionalidades

La clase HttpDataObjsServlet proporciona la posibilidad de acceder por HTTP a la base de datos de hipergate. Esta funcionalidad sirve, por ejemplo, para crear hojas Excel vinculadas con hipergate mediante objetos XMLHttpRequest capaces de actualizar automáticamente información en hipergate que se está actualizando de forma local en la hoja Excel del usuario.

## Instalación

HttpDataObjsServlet no viene instalado por defecto con hipergate.  
En la sección <web-app> del archivo /WEB-INF/web.xml hay que añadir :

```
<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>
```

y

```
<servlet-mapping>
 <servlet-name>HttpDataObjsServlet</servlet-name>
 <url-pattern>/servlet/HttpDataObjsServlet</url-pattern>
</servlet-mapping>
```

Tras rearrancar el contenedor de servlets puede probarse el acceso mediante:

`http://nombre_del_host/servlet/HttpDataObjsServlet?command=ping`

El servlet debe devolver la respuesta :

HttpDataObjsServlet ping OK

## Parámetros de entrada

El servlet debe llamarse por HTTP POST para escribir datos y por POST o GET para leerlos.

Los parámetros de entrada son los siguientes:

**profile** : Nombre del archivo de configuración del cual se leerán las propiedades de conexión a la base de datos. Este parámetro es opcional. El valor por defecto es "hipergate".

**user** : GUID o e-mail del usuario de la tabla `k_users` que se usará para leer o escribir en la base de datos. Este NO es el valor de la propiedad `dbuser` del archivo de configuración `.cnf` sino la identificación de un usuario de hipergate. Este parámetro es obligatorio.



**password** : Contraseña para el usuario anterior. Este parámetro es obligatorio.

**command** : Debe ser `query`, `update` o `ping`. Este parámetro es obligatorio.

**class** : Nombre de una clase derivada de `com.knowgate.dataobjs.DBPersist` o, alternativamente, el nombre de una implementación del interfaz `com.knowgate.hipergate.datamodel.ImportLoader`. Este parámetro es opcional. El valor por defecto es `com.knowgate.dataobjs.DBPersist`.

**table** : Nombre de una tabla o vista en la base de datos. Este parámetro es obligatorio para leer datos, se usa para escribirlos sólo cuando se omite el parámetro `class`.

**fields** : Nombres de columnas de la tabla anterior separados por comas. Este parámetro es obligatorio para leer datos, no se usa para escribirlos.

**where** : Cláusula de filtrado en la tabla anterior. Este parámetro es obligatorio para leer datos, no se usa para escribirlos.

**maxrows** : Número máximo de registros a leer en una consulta. Este parámetro es opcional para leer datos, no se usa para escribirlos. El valor por defecto es 500.

**skip** : Número de registros a saltar antes de leer el primero en una consulta. Este parámetro es opcional para leer datos, no se usa para escribirlos. El valor por defecto es 0.

**coldelim** : Delimitador de columnas. Este parámetro es opcional para leer datos, no se usa para escribirlos.

**rowdelim** : Delimitador de filas. Este parámetro es opcional para leer datos, no se usa para escribirlos.

## Leer datos

Esto es un ejemplo de cómo leer datos desde un cliente VBA. Recupera todas las empresas de una determinada área de trabajo cuyo identificador de sector de actividad no sea nulo.

```
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

```

Los datos se devuelven en formato de texto delimitado, en este ejemplo algo como:

```
ACME|GADGETS|ACTIVE;ASTROTECH|SPACE|ACTIVE;IBM|COMPUTERS|ACTIVE
```

En VBA estos datos pueden procesarse fácilmente con sentencias Split:

```

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

```

## Escribir datos

Para escribir datos los nombres de las columnas deben pasarse como parámetros de POST. Si las columnas son de tipo numérico o fecha, además del nombre hay que añadir el tipo SQL y, para las fechas, la máscara de formato.

El siguiente ejemplo escribe en la tabla k\_companies usando la clase Company del paquete 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=01234567890123456
7890123456789AB&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

```

Los tipos de datos que pueden seguir a los nombres de las columnas son: CHAR, VARCHAR, DATE, DATETIME, TIMESTAMP, SMALLINT, INTEGER, FLOAT, DOUBLE, DECIMAL, NUMERIC.


## Cómo se graban los datos

Los datos se graban bien instanciando un objeto de la subclase de DBPersist especificada y llamando al método `store()` de dicha subclase, bien mediante una llamada a una implementación del interfaz `com.knowgate.hipergate.datamodel.ImportLoader`

## Seguridad

El usuario especificado debe tener permisos suficientes sobre el área de trabajo para poder leer y escribir datos.

La clase `HttpDataObjsServlet` llama internamente primero al método `authenticate()` en la clase `ACL` del paquete `com.knowgate.acl`. A continuación, si la tabla a leer o escribir contiene una columna cuyo nombre sea `gu_workarea`, entonces llama a los métodos `isAdmin()`, `isPowerUser()` e `isUser()` de la clase `WorkArea` en `com.knowgate.workareas` para determinar si el usuario tiene privilegios suficientes para leer o escribir en la tabla dada.

 Se recomienda limitar el uso de `HttpDataObjsServlet` mediante medidas adicionales de seguridad como autenticación básica del servidor web y restricción por IP.

## Ejemplos adicionales

### Variables globales y función genérica AjaxPost para enviar peticiones HTTP POST desde VBScript

```
WORKAREA = "5262a821135070db7b3100126c066ced" ' Área de
Trabajo de TEST
USERID = "5262a821135070db7d310012ac122ac7" ' Identificador
del usuario del entorno de TEST
PASSWD = "TEST" ' Password del usuario del entorno de TEST
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
```

### Buscar un contacto dado su e-mail

```
' Devuelve: Identificador Único del Contacto en
HiperGate|Nombre|Apellidos|Razon Social
EMAIL = "usuario@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"
```

### Buscar un contacto dado un identificador procedente de un sistema externo y previamente grabado en hipergate

```
' Devuelve: Identificador Único del Contacto en
HiperGate|Nombre|Apellidos
```

```
ID = "101" ' Identificador Único del Cliente en el sistema
externo 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"
```

## Grabar o actualizar un contacto

```
' Usa el email como clave primaria para no duplicarlos
' Devuelve: Identificador Único del Contacto en Hipergate
CHAR(32)
ID = "104" ' Identificador Único del Contacto en el sistema
externo VARCHAR(50)
NOMBRE = "Fulanito"
APELLIDOS = "de Tal"
EMAIL = "fulanito@losdetal.com"
EMPRESA = "ACME"
NACIONALIDAD = "es" ' Código ISO 2 letras minúsculas país
PAIS = "us" ' Código ISO 2 letras minúsculas país
CLIENTE = AjaxPost (CONNECTION_PARAMETERS &
"&command=update&class=com.knowgate.crm.ContactLoader" &
"&id_contact_ref=" & ID & "&tx_name=" & NOMBRE &
"&tx_surname=" & APELLIDOS & "&tx_email=" & EMAIL &
"&nm_legal=" & EMPRESA & "&id_nationality=" & NACIONALIDAD &
"&id_country=" & PAIS)
```

## Grabar una nueva oportunidad

```
' Devuelve: Identificador Único de la oportunidad en
Hipergate CHAR(32)
ID = "104" ' Identificador Único del Contacto en el sistema
externo VARCHAR(50)
' Para actualizar una oportunidad añadir el campo
"&gu_opportunity=" & GUID_DE_LA_OPORTUNIDAD
OBJETIVO = "Producto o Servicio 1" ' Valor para el campo
k_opportunities.id_objetivo
IMPORTE = "5" ' Importe de la venta
INTERES = "1" ' Grado de interés del cliente: 0=Ninguno,
1=Poco, 2=Bastante, 3=Mucho
ESTADO = "NUEVA" ' Estado de la oportunidad: NUEVA | ABIERTA
| GANADA | PERDIDA | APLAZADA | ABANDONADA
NOTAS = "" ' Notas y comentarios a la oportunidad
OPORTUNIDAD = AjaxPost (CONNECTION_PARAMETERS &
"&command=update&class=com.knowgate.crm.OpportunityLoader" &
"&id_ref=" & ID & "&bo_private SMALLINT=0&id_ref=" & ID &
"&id_objetivo=" & OBJETIVO & "&im_revenue FLOAT=" & IMPORTE
& "&tx_company=" & EMPRESA & "&tx_contact=" & NOMBRE & "%20"
& APELLIDOS & "&tl_opportunity=" & OBJETIVO & "%20/%20" &
NOMBRE & "%20" & APELLIDOS & "&lv_interest SMALLINT=" &
INTERES & "&id_status=" & ESTADO & "&tx_note=" & NOTAS)
```

## Cambiar estado de oportunidad "NUEVA" a "GANADA"

```
ID = "104" ' Identificador Único del Contacto en el sistema
externo VARCHAR(50)
OBJETIVO = "Producto o Servicio 1" ' Valor para el campo
k_oportunities.id_objetivo
ESTADO = "GANADA"
' Primero recuperar el Identificador Único del Individuo
(GUIID) en Hipergate a partir del Identificador Único en el
sistema externo
GUID_CONTACTO = Left(AjaxPost(CONNECTION_PARAMETERS &
"&command=query&table=k_contacts" & "&where=gu_workarea%3D'"
& WORKAREA & "'%20AND%20id_ref%3D'" & ID & "'" &
"&fields=gu_contact"), 32)
' Hay que recuperar y regregar todos los campos en cada
transaccion
CAMPOS =
"gu_oportunidad,gu_writer,bo_private,dt_next_action,dt_last_c
all,lv_interest,nu_oportunities,gu_campaign,gu_company,gu_co
ntact,tx_company,tx_contact,tl_oportunidad,tp_oportunidad,tp_o
rigin,im_revenue,im_cost,id_objetivo,id_message,tx_note"
' Con el GUID del individuo en Hipergate buscar la
oportunidad por objetivo (si hay varias considerar sólo la
primera)
OPORTUNIDADES = AjaxPost(CONNECTION_PARAMETERS &
"&command=query&table=k_oportunities" &
"&where=gu_workarea%3D'" & WORKAREA &
"'%20AND%20gu_contact%3D'" & GUID_CONTACTO &
"'%20AND%20id_objetivo%3D'" & OBJETIVO & "'&fields=" &
CAMPOS)
' Almacenar la oportunidad en una array unidimensional
haciendo split del string devuelto por filas y columnas
OPORTUNIDAD = Split(Split(OPORTUNIDADES, ROW_DELIM,
MAX_ROWS)(0), COL_DELIM)
PARAMETROS = "" ' Este string contendrá los parámetros de
ida para regregar excepto id_status y tx_cause que se re-
asignaran programaticamente en la URL final
' Componer los parámetros de entrada a partir de los campos
leídos
CAMPOS = Split(CAMPOS,",")
For o = 0 To UBound(OPORTUNIDAD)
 If OPORTUNIDAD(o)="null" Or IsNull(OPORTUNIDAD(o)) Then
 OPORTUNIDAD(o) = ""
 ' No mandar parámetros con cadenas vacías
 If Len(OPORTUNIDAD(o))>0 Then
 TIPO = ""
 ' Para los campos que no son de tipo VARCHAR,
 especificar su tipo SQL en el nombre del parametro
 If CAMPOS(o)="bo_private" Or CAMPOS(o)="lv_interest"
Then TIPO = " SMALLINT"
 If CAMPOS(o)="nu_oportunities" Then TIPO = " INTEGER"
 If CAMPOS(o)="im_cost" Or CAMPOS(o)="im_revenue" Then
TIPO = " FLOAT"
 If CAMPOS(o)="dt_next_action" Or
```

```

CAMPOS(o)="dt_last_call" Then TIPO = " DATE"
PARAMETROS = PARAMETROS & "&" & CAMPOS(o) & TIPO & "=" &
OPORTUNIDAD(o)
End If
Next
' Con el GUID de la oportunidad actualizar sus datos
AjaxPost CONNECTION_PARAMETERS &
"&command=update&table=k_oportunities" &
"&where=gu_workarea%3D'" & WORKAREA &
"'%20AND%20gu_opportunity%3D'" & OPORTUNIDAD(0) & "'" &
PARAMETROS & "&id_status=" & ESTADO & "&tx_cause=VENTA"

```

# 15

## Calendario

El calendario de hipergate se puede sincronizar con el de Google utilizando la clase `com.knowgate.gdata.GCalendarSynchronizer`.

La sincronización es bidireccional: las actividades de hipergate se escriben en el calendario de Google y los eventos de Google se escriben en el calendario de hipergate.

La correspondencia unívoca entre actividades de hipergate y eventos de Google se lleva a cabo utilizando el identificador único iCalendar como clave primaria.

Cuando la sincronización automática está activada, cada usuario de hipergate solo puede tener un único calendario de Google asociado. Si una cuenta de Google tiene varios calendarios entonces solo uno de ellos puede ser sincronizado simultáneamente con el calendario de un usuario de hipergate.

Para habilitar la sincronización automática, hacer lo siguiente:

1. Poner la propiedad `gdatasync=1` en `hipergate.cnf` y re-iniciar el servidor web.
2. Ir a la configuración del Área de Trabajo y asegurarse de que los módulos de Herramientas Colaborativas y Gestión de Contraseñas están habilitados para un grupo al que pertenezca el usuario de hipergate que manejará el calendario.

3. Estando conectado como el usuario de hipergate que manejará el calendario, ir al Gestor de Contraseñas y crear una nueva entrada para GMail. Añadir en ella el e-mail de la cuenta de GMail, la contraseña de acceso y el nombre del calendario de Google que se debe sincronizar con el calendario personal del usuario actualmente conectado.
4. Una vez que la ficha de acceso a GMail ha sido creada en el Gestor de Contraseña queda habilitada la sincronización automática entre hipergate y Google para todas las operaciones de lectura y escritura realizadas a través del interfaz web. Las llamadas directas a métodos del paquete `com.knowgate.addrbook` sin pasar por el interfaz web no disparan ninguna sincronización con Google Calendar.

## Google Maps

A partir de la versión 4.1, hipergate trae integrada la visualización de direcciones con Google Maps.

Para activar la conexión con Google es preciso añadir la clave de Google Maps API en la propiedad `googlemapskey` de `hipergate.cnf`.

Las posiciones en el mapa se generan a partir de las direcciones en hipergate.

Las direcciones de hipergate se pueden enviar al API de Google. Desde el interfaz web para mostrar sus posiciones en un mapa. Para habilitar la integración con Google Maps hacer lo siguiente:

1. Poner la propiedad `googlemapskey` en `hipergate.cnf` Si no se dispone de una clave de Google Maps, registrarse para obtener una [aquí](#).
2. Para que una dirección sea enviada al API de Google debe contener al menos un nombre de vía y municipio al que pertenece.

La página JSP que genera los mapas de Google es `common/google_map.jsp`

Esta página toma como parámetro el GUID de la dirección e intenta posicionarla en el mapa de Google usando el Tipo de Vía, Nombre de Vía, Número de Vía, Ciudad, Provincia y País.



La página `google_map.jsp` es llamada como un pop-up bien desde el listado de contactos o compañías (en el icono de la bola terrestre) bien desde el enlace Mapa del formulario de edición de direcciones.

## Cálculo de distancias

hipergate dispone de otra página JSP en `common/distance_gmap.jsp` para el cálculo de distancias entre dos puntos.

La distancia en kilómetros entre dos puntos primero se calcula usando Google Maps y luego se almacena cacheada en la tabla `k_distances_cache`. Los puntos de origen y destino se identifican con cadenas arbitrarias que son válidas siempre y cuando Google Maps las reconozca de forma inequívoca.

Cuando se pide nuevamente la distancia entre dos puntos que han sido previamente calculados, la página `distance_gmap.jsp` intenta primero consultar el cache de la tabla `k_distances_cache` y, si no encuentra los puntos de origen y destino indicados, entonces realiza una llamada a Google Maps.

`distance_gmap.jsp` es una página diseñada para ser llamada dentro de un FRAME, lo que hace es escribir el valor calculado para la distancia en el INPUT indicado como parámetro en la URL de llamada. Este INPUT debe pertenecer al primer FRAME de la ventana padre de `distance_gmap.jsp`.

# 16

## API de acceso externo al calendario

El calendario de hipergate es accesible remotamente a través de HTTP.

### Modelo lógico del calendario.

La unidad básica de asignación en el calendario es la actividad, representada por la clase `Hipergate.CalendarMeeting` en el API cliente .NET y por la clase `com.knowgate.addrbook.client.CalendarMeeting` en el API cliente Java.

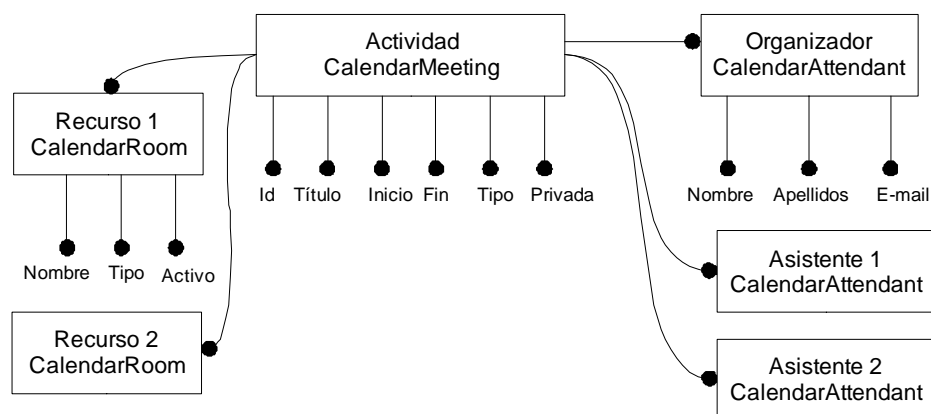
Las actividades se identifican de forma unívoca por una cadena de caracteres que sigue la recomendación de iCalendar para la composición de claves únicas.

Cada actividad tiene un organizador, título, fecha y hora de inicio y fin y, opcionalmente una descripción extendida.

Las actividades pueden opcionalmente tener personas asistentes y recursos asignados.

Los asistentes se identifican de forma unívoca por su email.

Los recursos deben tener un nombre localmente único para cada calendario.



## Modelo de seguridad

El control de acceso al calendario se realiza verificando un email de usuario y su correspondiente contraseña. Para acceder al calendario, el usuario debe haber sido previamente creado en hipergate. En hipergate existe una relación uno a uno entre usuarios y calendarios, de manera que para cada calendario hay un único usuario y contraseña de acceso.

## Proceso de autenticación

El servicio web del calendario no mantiene estados ni sesiones en el lado del servidor. El proceso de autenticación consiste en pedir al servicio web del calendario un token de seguridad. Este token de seguridad se obtiene pasando como parámetros el email y la contraseña del usuario. Si el email y la contraseña son correctos el servidor devuelve un token que debe ser utilizado en todas las llamadas posteriores de la misma sesión.

## Instalación en el servidor

Para que el servicio web del calendario esté disponible hay que activar el servlet `com.knowgate.http.HttpCalendarServlet`

La activación del servlet se logra añadiendo las siguientes líneas en el archivo `/WEB-INF/web.xml` de la webapp de hipergate

```
<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>
```

## Tipos de datos

Los tipos de datos y longitudes máximas permitidas son los siguientes:

**id:** Alfanumérico máximo 50 caracteres.  
**gu:** Alfanumérico máximo 32 caracteres.  
**type:** Alfanumérico máximo 16 caracteres.  
**name:** Alfanumérico máximo 100 caracteres.  
**surname:** Alfanumérico máximo 100 caracteres.  
**title:** Alfanumérico máximo 100 caracteres.  
**active:** Booleano 0 ó 1.  
**privacy:** Booleano 0 ó 1.  
**email:** Alfanumérico sólo minúsculas máximo 100 caracteres. Debe ser una dirección de email válida que cumpla con la expresión regular `[\w\X2E_-]+@[ \w\X2E_-]+\X2E\D{2,4}`  
**description:** Alfanumérico máximo 254 caracteres.  
**comments:** Alfanumérico máximo 254 caracteres.  
**startdate:** Fecha y hora. Para la entrada vía HTTP GET o POST en formato es yyyyMMddHHmmss La salida es en formato estándar XML yyyy-MM-ddTHH:mm:ss  
**enddate:** Mismo formato que startdate  
**timezone:** 6 caracteres con formato `[+|-]hh:mm`

## API REST

El acceso directo a través de HTTP GET permite los siguientes comandos:

### connect

Obtiene un token de seguridad para un email de usuario y contraseña. En Hipergate existe un calendario y sólo uno para cada usuario de la aplicación. El usuario utilizado para llamar al método connect, es por consiguiente el organizador de todas las actividades que se convoquen o modifiquen durante la sesión.

#### Petición

`http://servidor/hipergate/servlet/HttpCalendarServlet?command=connect&user=usuario@dominio.com&password=xxxxxxx`

Respuesta XML (éxito) con el token de seguridad en el tag <value>

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="connect" code="0">
<error></error>
<value>pur74y7abpckbpq4h8twxkv94rf8fhjebuyvb8vj</value>
</calendarresponse>
```

Respuesta XML (error)

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="connect" code="-1">
<error>User not found</error>
<value></value>
</calendarresponse>
```

## disconnect

Cierra la sesión e invalida el token de seguridad.

### Petición

`http://servidor/hipergate/servlet/HttpCalendarServlet?command=disconnect&token=xxxxxxx`

### Respuesta XML

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="disconnect" code="0">
<error></error>
<value>true</value>
</calendarresponse>
```

## getMeetings

Obtiene el listado de actividades de un calendario entre dos fechas.

### Petición

`http://servidor/hipergate/servlet/HttpCalendarServlet?command=getMeetings&token=xxxxxxx&startdate=19900101000000&enddate=20200101000000&type=meeting`

El parámetro type es opcional.

El formato para la fecha de inicio y fin debe ser yyyyMMddHHmmss

### Respuesta 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>Algo el lunes de nueva</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>El miércoles de nueve a tres</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

Obtiene el listado de actividades de un calendario entre dos fechas y que utilizan un determinado recurso.

Petición

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command=getMeetings&token=xxxxxxx&startdate=19900101000000&enddate=20200101000000&room=EINSTEIN
```

## getMeeting

Obtiene el detalle de una actividad dado su identificador iCalendar.

Petición

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command=getMeetings&token=xxxxxxx&meeting=icalendar_id_of_meeting@hipergate.org
```

Respuesta 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>Encuentro X</title>
 <description>Descripción del encuentro 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>
 </meeting>
 </meetings>
 <rooms count="1">
 <room type="CLASSROOM" active="1">
 <name>EINSTEIN</name>
 <comments></comments>
 </room>
 </rooms>
</calendarresponse>
```

```

 <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

Obtiene el listado de todos los recursos.

### Petición

`http://servidor/hipergate/servlet/HttpCalendarServlet?command=getRooms&token=xxxxxxx&type=CLASSROOM`

El parámetro type es opcional.

### Respuesta 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

Obtiene el listado de recursos disponibles entre dos fechas.

Petición

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command=getAvailableRooms&token=xxxxxxx&startdate=19900101000000&enddate=20200101000000
```

El formato para la fecha de inicio y fin debe ser yyyyMMddHHmmss

Respuesta 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

Obtiene si un recurso está disponible entre dos fechas.

Petición

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command=isAvailableRoom&token=xxxxxxx&startdate=19900101000000&enddate=20200101000000&room=EINSTEIN
```

El formato para la fecha de inicio y fin debe ser yyyyMMddHHmmss

Respuesta XML (true en el elemento <value> si el recurso está disponible, false en caso contrario)

```
<?xml version="1.0" encoding="UTF-8"?>
<calendarresponse command="isAvailableRoom" code="0">
 <error></error>
 <value>true</value>
```

</calendarresponse>

## storeMeeting

Inserta una nueva actividad o actualiza una actividad ya existente.

### Petición

```
http://servidor/hipergate/servlet/HttpCalendarServlet?command=storeMeeting&token=xxxxxx&meeting=idicalendar@hipergate.org&title=Activity%20Title&startdate=20101117184000&enddate=20101118195000&rooms=ROOM1,ROOM2&attendants=guest@mail.com
```

Los parámetros meeting, title, startdate y enddate son obligatorios.

El parámetro meeting debe ser el identificador iCalendar de la actividad.

Los parámetros rooms y attendants son opcionales. Deben contener una lista de valores separados por comas con los nombres de los recursos y los emails de los invitados. Los emails deben pertenecer a cuentas de usuario previamente existentes en el mismo dominio de hipergate y que se puedan asignar a la actividad.

El usuario actualmente conectado es siempre añadido a la actividad como asistente y organizador, aunque su email no se incluyó explícitamente en la lista attendants.

### Respuesta

```
<?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>
 </organizer>
 </meeting>
 </meetings>
</calendarresponse>
```

© KnowGate 2003-2013. Esta documentación se distribuye bajo la licencia Creative Commons Attribution-NoDerivs-NonCommercial. <http://creativecommons.org/licenses/by-nd-nc/1.0/> Se permite copiar, redistribuir y modificar el documento sólo según los siguientes términos: 1º) Debe aparecer la atribución original a KnowGate. 2º) No se permite el uso del original ni ninguna modificación con fines comerciales. 3º) No se permiten trabajos derivados basados en esta documentación. 4º) Cualquier redistribución debe contener estos términos.

```

 <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>

```

## Librería cliente .NET

En la librería HipergateCalendarClient.dll pueden encontrarse las clases CalendarMeeting, CalendarRoom, CalendarAttendant y CalendarClient.

La clase principal de acceso al servicio web es Hipergate.CalendarClient.

## Clase Hipergate.CalendarClient

Esta es la clase principal de acceso al calendario. Mantiene la sesión en el lado del cliente almacenando internamente un token de seguridad que se envía con cada petición.

### **Connect (sServiceUrl As String, sUserEmail As String, sPassword As String) As Boolean**

Antes de ejecutar ningún método de CalendarClient es preciso llamar a método Connect para crear una sesión.

**sServiceUrl:** URL base del servicio. Típicamente  
`http://hostname.com/hipergate/servlet/HttpCalendarServlet`

**sUserEmail:** E-mail del usuario propietario del calendario al cual se quiere acceder. Este e-mail debe estar presente en las tablas k\_users y k\_fellows de hipergate.

**sPassword:** Contraseña del usuario. La misma que figure en la tabla k\_users de hipergate.

---

### **Disconnect () As Boolean**

Cierra la sesión del objeto CalendarClient.

---

### **IsAvailable (sRoom As String, dtStart As Date, dtEnd As Date) As Boolean**

Devuelve si un recurso está disponible entre dos fechas dadas.

**sRoom:** Nombre del recurso.

**dtStart:** Fecha de inicio.

**dtEnd:** Fecha de Fin.

---

### **GetRooms () As CalendarRoom()**

Devuelve un array con todos los recursos existentes, estén disponibles o no. Si no hay recursos devuelve Nothing.

---

### **GetRooms(sType As String) As CalendarRoom()**

Devuelve un array con todos los recursos existentes de un determinado tipo, estén disponibles o no. Si no hay recursos devuelve Nothing.

**sType:** Tipo de recurso.

---

#### **GetAvailableRooms(dtStart As Date, dtEnd As Date) As CalendarRoom()**

Devuelve un array con todos los recursos disponibles entre dos fechas dadas. Si no hay recursos disponibles devuelve Nothing.

**dtStart:** Fecha de inicio.

**dtEnd:** Fecha de fin.

---

#### **GetAvailableRooms(sType As String, dtStart As Date, dtEnd As Date) As CalendarRoom()**

Devuelve un array con todos los recursos disponibles de un determinado tipo entre dos fechas dadas. Si no hay recursos disponibles de dicho tipo devuelve Nothing.

**sType:** Tipo de recurso.

**dtStart:** Fecha de inicio.

**dtEnd:** Fecha de fin.

---

#### **GetMeeting (sMeetingId As String) As CalendarMeeting**

Devuelve un objeto de tipo CalendarMeeting a partir de su identificador único.

**sMeetingId:** Identificador iCalendar o GUID de la actividad (cualquiera de ambos).

---

### **GetMeetings(dtStart As Date, dtEnd As Date) As CalendarMeeting()**

Devuelve un array con todas las actividades que tengan lugar entre dos fechas.

**dtStart:** Fecha de inicio.

**dtEnd:** Fecha de fin.

---

### **GetMeetingsOfType(dtStart As Date, dtEnd As Date, sType As String) As CalendarMeeting()**

Devuelve un array con todas las actividades de un determinado tipo que tengan lugar entre dos fechas.

**dtStart:** Fecha de inicio.

**dtEnd:** Fecha de fin.

**sType:** Tipo de actividad. Las actividades estándar son:

<b>Tipo</b>	<b>Descripción</b>
meeting	Reunión
call	Llamada
followup	Seguimiento
breakfast	Desayuno
lunch	Comida
course	Curso
demo	Demostración de producto
workshop	Taller de trabajo
congress	Congreso
tradeshow	Feria

---

### **GetMeetingsForRoom (dtStart As Date, dtEnd As Date, sRoom As String) As CalendarMeeting()**

Devuelve un array con todas las actividades que hacen uso de un determinado recurso entre dos fechas.

**dtStart:** Fecha de inicio.

**dtEnd:** Fecha de fin.

**sRoom:** Nombre del recurso.

---

### StoreMeeting (oMeet As CalendarMeeting) As CalendarMeeting

Graba una nueva actividad o actualiza una ya existente. Las actividades se identifican de forma unívoca mediante el identificador iCalendar (propiedad id del objeto CalendarMeeting).

El identificador iCalendar puede ser establecido a priori por el cliente, o dejar que sea la propia librería quien asigne uno automáticamente. En cualquier caso, todas las actividades tienen siempre un identificador único iCalendar.

Esta función devuelve el objeto pasado como parámetro tras ser grabado en el servidor. El objeto grabado puede contener cambios con respecto al objeto original. En particular, se le añaden automáticamente el identificador iCalendar y el GUID si no existían previamente.

**oMeet:** Actividad a grabar.

---

### DeleteMeeting (sMeetingId As String) As CalendarMeeting

Borra una actividad.

**oMeet:** Identificador iCalendar o GUID de la actividad.

---

## Ejemplo de uso del cliente .NET desde VisualBASIC

```
' Crear un objeto cliente de calendario
Dim c As New Hipergate.CalendarClient

' Conectarse al servicio
c.Connect("http://localhost/hipergate/servlet/HttpCalendarServlet",
"user@test.com", "TEST")

' Comprobar si el recurso de nombre NEWTON está disponible ahora mismo
Dim a As Boolean = c.IsAvailable("NEWTON", DateValue(Now), DateValue(Now))

' Obtener un array con todos los recursos
Dim r() As Hipergate.CalendarRoom = c.GetRooms()
```

```

' Obtener un array con todos los recursos disponibles ahora mismo
Dim d() As Hipergate.CalendarRoom = c.GetAvailableRooms(DateValue(Now),
DateValue(Now))

' Obtener un listado de todas las actividades entre el 1/1/2000 y el
31/12/2020
Dim m() As Hipergate.CalendarMeeting = c.GetMeetings(New Date(2000, 1, 1,
0, 0, 0), New Date(2020, 12, 31, 23, 59, 59))

' Graba una nueva actividad de 2h e duración en el Aula Newton con 1
asistente
Dim e As New Hipergate.CalendarMeeting
Dim f As Hipergate.CalendarMeeting
e.title = "Titulo de la actividad hasta 100 caracteres"
e.description = "Descripcion larga de la actividad hasta 1000 caracteres"
e.startdate = Now ' Fecha Inicio
e.enddate = DateAdd(DateInterval.Hour, 2, e.startdate)
e.AddRoom("NEWTON") ' Llamar a AddRoom una vez por cada recurso
e.AddAttendant("administrator@hipergate-test.com")
f = c.StoreMeeting(e)
' Tras grabar la actividad algunos valores adicionales pueden volver en el
objeto retornado por el método StoreMeeting
' En particular, es relevante la propiedad id que es la que identifica a la
actividad unívocamente para futuros accesos

' Recuperar la actividad anterior
Dim g As Hipergate.CalendarMeeting = c.GetMeeting(f.id)
' Atrasar 10 minutos la fecha de inicio y regrabar
g.startdate = DateAdd(DateInterval.Minute, 10, g.startdate)
c.StoreMeeting(g)

' Cerrar la sesión
c.Disconnect()

```

## Librería cliente Java

En el paquete `com.knowgate.addrbook.client` pueden encontrarse las clases del API cliente para Java del servicio web del calendario.

La clase principal de acceso al servicio web es `CalendarServices`.

## Ejemplo de uso del cliente desde Java

```

CalendarServices oCal = new CalendarServices();

// Conectarse al servicio
oCal.connect("http://localhost/hipergate/servlet/HttpCalendarServlet","admin
istrator@hipergate-test.com","TEST");

// Comprobar si el recurso de nombre NEWTON esta disponible ahora mismo
boolean a = oCal.isAvailableRoom("NEWTON", new Date(),new Date ());

// Obtener un array con todos los recursos
ArrayList<CalendarRoom> r = oCal.getRooms();

// Obtener un array con los recursos disponibles ahora mismo
ArrayList<CalendarRoom> d = oCal.getAvailableRooms(new Date (),new Date ());

// Obtener un array de todas las actividades entre el 1/1/2000 y el
31/12/2020

```



```

ArrayList<CalendarMeeting> m = oCal.getMeetings(new Date (100,0,1), new
Date (120,11,31));

// Graba una nueva actividad de 2h de duracion en el Aula Newton con 1
asistente
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()+3600000));
e.addRoom("NEWTON");
e.addAttendant("administrator@hipergate-test.com");
oCal.storeMeeting(e);

```