Gnome BD

Michael Lausch
michael.lausch@1012surf.net

Rodrigo Moya
rmoya@chez.com

Gnome BD
por Michael Lausch y Rodrigo Moya

Tabla de contenidos

Lista de tablas

Tabla de figuras

Capítulo 1. Introducción

GNOME-DB es un completo conjunto de librerías y aplicaciones que permiten acceder a bases de datos de diferentes vendedores, usando una API uniforme y un kit de herramientas. Ofrece un alto nivel de abstracción del sistema de base de datos recubierto, incluyendo también algunas herramientas de ayuda para los usuarios y los desarrolladores.

El acceso a las bases de datos se realiza por medio de una librería cliente. OBDC y SQL son los estándares establecidos. El problema es que ODBC no especifica el protocolo de enlace y que no existen manejadores ODBC para algunas bases de datos. Se puede usar RPC, TCP/IP, memoria compartida y señales para enviar las peticiones del cliente al servidor. Deberá usar la librería ODBC particular de la base de datos. Esta librería puede no estar disponible para la CPU o sistema operativo en el cual se ejecute el cliente.

El propio SQL no esta del todo estandarizado, así que la compatibilidad a nivel de código no puede asegurarse para todos los servidores de bases de datos. Y para algunos tipos de servidores, SQL no es siquiera una opción (piense en LDAP).

GNOME-DB intenta atajar el problema de ODBC y ayudar con el problema de SQL. Es una especie de intermediario (o puede ser ampliado para ser esta capa intermedia) de acceso a distintas fuentes de datos. He echado una larga mirada a la implementación de ADO de Microsoft y GNOME-DB esta muy inspirado por este modelo. Ofrece una visión de alto nivel de la fuente de datos y dispone de algunos puntos con acceso de bajo nivel a la base de datos para tareas especiales.

Capítulo 2. Descripción de la arquitectura

GNOME-DB se compone de 3 capas independientes. Los suministradores GDA cubren la capa inferior, y son servidores Corba con la tarea de asociar la API particular del RDBMS al modelo GDA.

Después, en la capa intermedia, tenemos las librerías cliente, compuestas por dos módulos: una librería Corba, que esconde toda la complejidad de Corba al las aplicaciones cliente, acompañada de la librería de 'widget' GDA, que proporcionan una forma sencilla de incorporar la arquitectura GNOME-DB a las aplicaciones con interfaz usuario GNOME.

Finalmente, la capa superior acoge todas la apiicaciones de cliente proporcionadas en la distribución, así como cualquier aplicacion que haga uso de las librerías de cliente.

Gnome-DB esta basado en Corba y usa ORBit (http://www.labs.redhat.com/ORBit) como implementación de Corba. Básicamente el sistema proporciona un fichero IDL como interfaz para el cliente. Para cada tipo diferente de fuente de datos (ODBC, Sybase, Informix, MySQL, LDAP, ...) se debe escribir un servidor. Este servidor es idealmente una librería compartida que se enlaza con un pequeño programa manejador para formar un servidor Corba auto suficiente. La aproximación vía librería compartida hace posible el uso del servidor como una librería para el cliente, reduciendo el tiempo de las llamadas (no hay sockets ni se necesitan cambios de contexto). La librería compartida puede ejecutarse como un programa auto suficiente con la librería Goad, que de forma mágica envuelve la librería compartida con un sencillo programa principal que ejecuta el servidor en su propio espacio de memoria, aislando el servidor de las caídas del cliente.

La ventaja de usar un servidor Corba es que el servidor puede ejecutarse en otra maquina, repartiendo la carga. Esto es posible porque Orbit es pequeño y rápido. Puede usar el mismo servidor desde un lenguaje de guiones preparado para Corba y puede trazar y controlar las transacciones en el servidor, sin necesitar cambios en el cliente.

2.1. Servidor ODBC

El tipo de servidor mas complicado es el servidor ODBC. La razón es OBBC proporciona por si mismo la capacidad de usar diferentes implementaciones de ODBC ('mapas' a diferentes servidores CORBA). Así que hay dos puntos en los que se puede configurar erróneamente el sistema. Véase la figura gda-arch-odbc

Figura 2-1. El servidor ODBC

Para configurarlo, primero deberá configurar la fuente de datos ODBC, disponiendo del fichero .odbc.ini en su directorio privado. Después, obtener la siguiente información:

- Nombre GDA: el que más se adapte a sus necesidades
- Proveedor: gda-odbc
- DSN: es cualquier DSN ODBC valido que se pasa directamente al manejador general ODBC
- Descripción: a su elección
- Configurador: Ninguno

2.2. Servidor MySQL

El servidor MySQL, como algunos de los demás servidores Gnome-Db, proporciona una proyección de la API nativa de MySQL sobre le modelo Gnoem-Db.

Para configurar una fuente de datos GDA para acceder a una base de datos MySQL, necesitara proporcionar la siguiente información:

- Nombre GDA: el que más se adapte a sus necesidades
- Proveedor: gda-mysql
- DSN: es una lista separada por ';' de las opciones que deben pasarse al servidor MySQL. Esto incluye:
    - Ordenador: nombre del ordenador en el que se ejecuta el servidor
    - Base de datos: nombre de la base de datos a acceder
    - Puerto: numero del puerto usado para conectar al servidor
    - UNIX_SOCKET: ruta al socket UNIX (consulte los documentos MySQL)
    - FLAGS: parámetros de la conexión

En todos los casos, necesitara chequear su instalación de MySQL y consultar la documentación para introducir valores validos para cada uno de esos parámetros.

- Descripción: a su elección
- Configurador: Ninguno

2.3. Servidor Postgres

El servidor PostgreSQL usa la API nativa para acceder a la base de datos recubierta.

2.4. Servidor Oracle

El servidor Oracle es la primera base de datos comercial soportada por Gnome-Db, gracias a Stephan Heinze (mailto:stephan.heinze@xcom.de). La versión de Oracle soportada es la 8.0 o superior, ya que es la primera versión disponible para Linux.

2.5. Servidor Sybase

2.6. Servidor LDAP

2.7. Estructura de directorios

Si va a trabajar con el código fuente de Gnome-Db, podrá encontrase algo perdido al principio, ya que hay un buen numero de directorios. He aquí una pequeña guia:

doc

Documentación. Este directorio contiene un subdirectorio para cada localización.

gda

Los ficheros IDL. Puede usarlos para generar un cliente personalizado (o un servidor).

gda-clnt

La librería cliente. Un librería que envuelve (aisla|independiza) de las peculiaridades de Corba. Implementa la mayoría de "housekeeping, buffering and cursor control". Observe que el control del puntero podrá también implementarse en el servidor en futuras versiones.

gda-clnt-ui

librería de widget clientes. Esta la librería contiene los widget Gtk+/Gnome que permiten una sencilla integración de la Api Gnome-Db en las aplicaciones Gnome.

gda-dev

Guiones y programas de ayuda para el desarrollo de aplicaciones GDA.

gda-ldap-server

Servidor Corba que implementa el acceso a LDAP. Actualmente vacío.

gda-manager

Herramientas generales de configuración y puesta en servicio. Contendrá el gestor ODBC y cosas de este tipo. Actualmente vació.

gda-mysql-server

Servidor Corba para la base de datos MySql. Actualmente solo esta implementada una parte muy pequeña. Mas cosas en camino.

gda-odbc-server

Servidor Corba para bases de datos ODBC. Actualmente el servidor mas desarrollado. Permite acceder a las bases de datos Solid y Postgres. Si su base de datos dispone de un manejador ODBC, puede probar este servidor para acceder a ella.

gda-postgres-server

Servidor Corba para las bases de datos PostgreSQL. Usa la API nativa de PostgreSQL para proporcionar el acceso a la base de datos.

gda-sybase-server

Servidor Corba para las bases de datos Sybase. Actualmente vacio.

gda-oracle-server

Servidor Corba para las bases de datos Oracle.

iodbc

Gestor de la librería cliente ODBC. Analiza el fichero ~/.odbc.ini y enlaza la librería compartida adecuada con el servidor. Funciona, pero se tendría que rediseñar para usar los estándares Gnome para ficheros de configuración.

postgres

Librería de vinculo ODBC para Postgres. Debería ser una interfaz nativa, pero estaba ya disponible así que se uso.

fe

Aplicación de acceso a SQL. Es una aplicación que hace uso de toda la potencia facilitada por el cliente GDA y las librerías de UI (Interfaz de usuario) GDA. Proporciona una forma fácil de interrogar y gestionar su base de datos.

testing

Programas para probar las distintas piezas del sistema. Reviselas para hacerse una idea de las cosas que pueden hacerse con las librerías GDA.

Capítulo 3. Instalación

3.1. Introducción

La instalación de Gnome-Db no es una tarea trivial en estos momentos. No es difícil, pero hay bastantes cosas a las que hay que prestar atención antes de intentar su ejecución. Este documento intenta afrontar todas estas "dificultades" con las que seguramente se encontrara.

Tan pronto como se disponga de paquetes RedHat y Debian, todo este tedioso proceso no sera necesario, ya que el mismo paquete se hará responsable de todo el proceso de instalación. Para una instalación estándar, no hay pasos adicionales; sin embargo, seria mejor conocer los entresijos de configuración de Gnorba, Odbc y Gda, para el caso que nos crucemos con una instalación no estándar, o en el caso de que surjan problemas durante una instalación estándar.

3.2. Instalación

La instalación depende del formato en el cual se escogido realizar la copia. Si ha obtenido un paquete como RPM o DEB, consulte como instalar nuevos paquetes en la documentación de su gestor de paquetes.

Si se obtuvo el código fuente (en un paquete tar "tarball"), deberá compilar el software. Para hacerlo, una vez desempaquetado el árbol de fuentes, deberá:

```
./configure
make
make install
```

Esto generara los makefiles para su plataforma especifica, compilara todo el árbol de fuentes, e instalara los binarios y la documentación en su sistema.

Si no encuentra el fichero llamado configure, debería haber uno llamado autogen.sh. En este caso, ejecute autogen.sh, que creara y ejecutara el fichero configure generado.

Puede especificar varios parámetros para configure (o autogen.sh). Los mas significativos son (puede consultar todos los parámetros disponibles ejecutando configure –help):

- –prefix=<directorio>: Prefijo donde se instalara el paquete
- –with-gnome=<directorio>, –with-gnome-includes, –with-gnome-libs: Directorio donde se instalaran los fichero de desarrollo Gnome
- –with-mysql=<directorio>: Indica el directorio donde esta instalado el servidor MySQL
- –with-postgres=<directorio>: Indica el directorio donde esta instalado el servidor PostgreSQL
- –with-sybase=<directory>: Indica el directorio donde esta instalado el servidor Sybase
- –with-ldap=<directory>: Indica el directorio donde esta instalado el servidor LDAP
- –with-oracle=<directory>: Indica el directorio donde esta instalado el servidor ORACLE

Si encuentra algún problema durante el proceso de configuración, compilación o instalación, no dude en dirigirse a la lista de correo Gnome-Db (gnome-db-list@gnome.org, envié primero un mensaje de e-correo a gnome-db-list-request@gnome.org con SUBSCRIBE en el campo motivo si no esta todavía suscrito).

3.3. Configuración

En primer lugar, debe tener un fichero llamado gdalib en su directorio $HOME/.gnome. Este fichero contiene los parámetros de configuración de todas las fuentes de datos a las que accederá. Si no lo tiene, puede coger el fichero gdalib.sample que viene con la distribución y modificarlo para que se adapte a la configuración de su sistema, o mejor aun, puede ejecutar la aplicación de acceso y crear las fuentes de datos desde una sencilla interfaz gráfica.

La primera cosa que debe hacer es disponer de una base de datos configurada y en funcionamiento, para la que exista un proveedor Gda. Compruebe estos aspectos en la documentación de su base de datos.

Después deberá copiar el fichero gnome_db.gnorba incluido con la distribución en el directorio etc Gnorba de su sistema, que normalmente es:

- /etc/CORBA/servers en RedHat 6.0 y derivados
- /usr/etc/CORBA/servers
- $PREFIX/etc/CORBA/servers, siendo PREFIX el directorio donde se haya instalado Gnome

Este fichero debe copiarse en el directorio correcto, con el fin de conseguir que los servidores Corba Gda sean visibles para la librería Goad. (Goad es el sistema de activación de objetos de Gnome, usado para activar objetos Corba).

3.4. Ejecutando aplicaciones

Una vez que todo esta correctamente instalado, es el momento de ver a Gnome-db en acción. Antes de hacerlo, sin embargo, hay una ultima cosa que debe hacer, añadir el directorio donde ha instalado las librerías Gnome-Db en la variable de entorno LD_LIBRARY_PATH, y el directorio donde se situaron los binarios en la variable de entorno PATH.

Entonces, simplemente ejecute una de las aplicaciones incluidas, como gdafe (la interfaz de SQL), gda-mgr (el configurador de Gda) o rolodex (una sencilla aplicación que muestra como puede usar de forma fácil la Api Gda para crear sus propias aplicaciones).

Si quiere desarrollar aplicaciones usando Gnome-Db, ejecute el constructor Gda, un completo Ide para desarrollar aplicaciones Gnome-Db, o eche un vistazo a la documentación de la Api y a los ejemplos de código para ver como puede integrar Gnome-Db en sus aplicaciones.

Capítulo 4. Administración y configuración de Gda

4.1. Proveedores y fuentes de datos

Cada diferente tipo de fuente de datos es manejada por un servidor especializado, llamado proveedor. Cada proveedor accede a un tipo de datos. Este puede ser una base de datos, pero no tiene porque ser así. Es teóricamente posible acceder a servidores LDAP, colas de correo y otras fuentes de datos, tan pronto como se implemente un proveedor. Por lo tanto la librería cliente no asume demasiadas cosas acerca de las características del proveedor. Por lo tanto la interpretación del significado del nombre de la fuente de datos es también responsabilidad del proveedor. Para disponer de un único y fácilmente configurable sistema de nombrado se usa un fichero de configuración. En este fichero los nombres se asocian a los proveedores y a la cadena de texto que necesita el proveedor para localizar la fuente de datos correcta. Como el proveedor interpreta la cadena de texto de conexión se describe en la sección dedicada a los proveedores concretos.

4.1.1. Configuración de un proveedor

Los proveedores Gda son servidores Corba que se activan usando Goad. Goad obtiene su información de configuración de una serie de ficheros situados en el directorio ${prefix}/etc/CORBA/servers, donde ${prefix} es la ruta de la instalación Gnome, o del directorio ~/.gnome/CORBA/servers. Estos ficheros tienen normalmente la extensión .gnorba. Los situados en el directorio de usuario tienen preferencia sobre el directorio general del sistema. El nombre del fichero de configuración no es significativo, pero los paquetes gnome-db usan gnome_db.gnorba como nombre de fichero del fichero de configuración estándar.

Cada proveedor se define en un registro de este fichero. El actual fichero entregado por defecto con gnome-db contiene un registro de ejemplo para cada uno de los posibles proveedores.

```
<10122>    [gda-odbc]
<10123>    type=exe
<10124>    repo_id=IDL:GDA/ConnectionFactory:1.0
<10125>    description=Acceso a la fuente de datos via Odbc
<10126>    location_info=gda-odbc-srv

    [gda-mysql]
    type=exe
    repo_id=IDL:GDA/ConnectionFactory:1.0
    description=Acceso a la fuente de datos via MySQL
    location_info=gda-mysql-srv

    [gda-postgres]
    type=exe
    repo_id=IDL:GDA/ConnectionFactory:1.0
    description=Acceso a la fuente de datos via Postgres
    location_info=gda-postgres-srv

    [gda-oracle]
    type=exe
    repo_id=IDL:GDA/ConnectionFactory:1.0
    description=Acceso a la fuente de datos via Oracle
    location_info=gda-oracle-srv
```

Registros del fichero de configuracion Goad

<10122>Es el nombre del proveedor. La combinación de este nombre y campo tipo debe ser única. Este significa que es posible tener un proveedor como librería compartida y como ejecutable independiente con el mismo nombre. Como el usuario de la librería puede escoger entre la implementación como librería compartida y el ejecutable independiente no esta todavía solucionado.

<10123>Es el tipo de servidor. si este campo toma el valor "exe", el proveedor esta implementado como un ejecutable independiente. Esto significa que el proveedor esta desacoplado de la aplicación que hace uso de Gda y por tanto a salvo de punteros que lo cuelguen y otros errores de la aplicación. La desventaja es que la latencia entre la aplicación y el servidor es mayor que con los servidores en la modalidad de librería compartida. Los servidores implementados como librería compartida están definidos con el valor "shlib" en este campo y se enlazan con la aplicación en tiempo de ejecución.

<10124>El valor de este campo debe ser "IDL:GDA/ConnectionFactory:1.0". Se usa para localizar los proveedores Gda en todos los ficheros de configuración Goad.

<10125>Es una pequeña descripción del servidor en una linea. Se visualiza en la aplicación gda-mgr y debería describir el proveedor. Destino únicamente a las personas.

<10126>Esta es la ruta al proveedor. Si la ruta no empieza por "/", la implemenatcion se busca dependiendo de su tipo. Si es de tipo "exe", se busca el proveedor en el PATH del usuario que arranco el servidor. Si es de tipo "shlib" se usa el método de localización de slibrerias compartidas. La forma exacta en esto de hace depende del sistema operativo, pero en Linux se usa la variable de entorno LD_LIBRARY_PATH, y después los directorios del fichero /etc/ld.so.conf.

Los proveedores ejecutables ofrecen algunas ventajas. Pueden están hospedados en una maquina diferente [1] y solo se necesita arrancar, con su tiempo asociado, el proveedor una vez. La desventaja es que si un proveedor de este tipo se cae, todos los clientes quedan desconectados de la base de datos. Para solventar este problema es posible que cada cliente solicite una nueva copia del proveedor (pero la Api de cliente no proporciona este indicador todavía).

El otro tipo de proveedores son los proveedores implementados como librerías compartidas. Estos proveedores se enlazan con el cliente cuando el cliente solicita algo del proveedor. La desventaja es que el servidor esta albergado en la misma maquina que el cliente. El tiempo de arranque es similar al de un proveedor ejecutable. La gran ventaja de este tipo de servidores es la sencillez de las pruebas del código del proveedor, ya que no debe conectar gda con un proveedor en ejecución y puede también capturar errores durante la inicialización del servidor.

Por omisión cada proveedor Gda esta disponible como ejecutable y como librería compartida. El proceso de generación y las convenciones usadas para implementar el proveedor aseguran la creación e instalación del ejecutable y la librería compartida. Así, los proveedores Gda están implementados en forma de librería compartida, la cual se enlaza después con un pequeño programa que actúa como un manejador para la librería compartida.

4.1.2. Nombres de las fuentes de datos

Uno de los problemas que resuelve Gda es el nombrado de fuentes de datos. Cada sistema de base de datos tiene su propia forma de definir nombres para sus bases de datos. Por ejemplo Mysql usa el nombre de máquina (hostname), numero de puerto, y el nombre de la base de datos. Otras bases de datos, como Solid usan solo el nombre de maquina y el numero de puerto. No hay soporte de múltiples bases de datos por servidor. Como el cliente no necesita todos estos detalles, existe un fichero de configuración llamado gdalib que nombra las fuentes de datos disponibles y define todas las propiedades de cada base de datos, de forma que se contacte con el servidor de base de datos correcto. La librería cliente accede a la información de este fichero de configuración y la envía al proveedor.

4.1.2.1. El fichero gdalib

El formato del fichero gdalib sigue las convenciones Gnome y describe cada fuente de datos y que proveedor esta disponible para usarse como servidor.

A continuación se muestra una parte de un fichero gdalib

```
<10122> [Datasources]
Solid=Solid-entry
Postgres-Native=Postgres-Native
```

```
MySQL-Native=MySQL-Native
<10123>
<10124>  [Solid-entry]
<10125>  Provider=gda-odbc
<10126>  DSN=Solid
<10127>  Description=Solid Test Database for ODBC
Configurator=MODULE=odbc;ENTRY=Solid

[Postgres-Native]
Provider=gda-postgres
DSN=DATABASE=test;HOST=localhost
Description=Postgres Test Database in native mode
Configurator=None

[Postgres-Native]
Provider=gda-mysql
DSN=DATABASE=test;HOST=localhost;PORT=1111
Description=MySQL Test Database in native mode
Configurator=None
```

Secciones del ficheros gdalib

<10122>Esta seccion nombra todas loas fuentes de datos. La clave es el nombre de la fuente de datos que debe usar el cliente. Su valor es el nombre de la sección que describe en este fichero la fuente de datos.

<10123>Esta sección describe una fuente de datos que representa una base de datos Solid. Solid es una base de datos con poco uso de memoria y poca escritura a disco, que proporciona una librería cliente ODBC.

<10124>El proveedor para esta base de datos es el proveedor gda-odbc. El valor de este registro se usa como nombre del servidor para la activación Goad.

<10125>Este es el registro mas importante. El valor de este registro es la cadena de texto enviado al proveedor para que sepa que base de datos acceder. La

forma en que los proveedores interpretan este registro se describe en la sección de proveedores.

<10125>El valor de este registro es una concisa descripción de la fuente de datos. Esta solo por conveniencia y no se usa para ningún propósito.

<10127>Este registro denota un modulo glib usado para realizar la configuración de este proveedor. Algunos proveedores como gda-odbc y gda-sybase necesitan ficheros adicionales de configuarcion. Este registro indica la librería que usara la librería cliente para acceder a la administración de la base de datos.

Esta funcionalidad no esta actualmente implementada. Entre tanto se deberán usar las herramientas de configuración de la fuente de datos o un editor de texto para proporcionar la configuración específica del proveedor.

4.1.2.2. El gestor y configurador Gda

Gda se entrega con un entorno gráfico de configuración (GUI) para realizar tareas de administración de la base de datos. Esta herramienta es la aplicación gda-mgr.

Como se indico anteriormente, cada proveedor se entrega con un modulo de configuración, bajo la forma de un modulo glib, que cargara la aplicación gda-mgr.

Notas

1. No tengo idea de como hacerlo todavía. Pienso que se deberían usar mensajes GIOP_FORWARD en esta situación. No he encontrado la forma de hacerlo todavía. Lo haré mas tarde.

Capítulo 5. La librería cliente Gda

5.1. Introducción

La librería cliente es una fina capa alrededor de la interfaz Corba. La capa se ocupa de la reserva y devolución de memoria cuando ya no necesite mas los resultados de una consulta, o cuando cierre la conexión a la base de datos. Su segundo propósito es disponer de un tampón ("cache"), para que sea posible desplazarse hacia delante y hacia atrás dentro de una consulta, aun si la base de datos no soporte este tipo de deslizamiento en el cursor. Por supuesto no se soportara las estrategias de consistencia y bloqueo de la base de datos en estos accesos, pero pienso que encontrare una solución para esto.

Se dispone también de soporte para manejar diferentes fuentes de datos con diferentes servidores y características de conexión. Para conseguir este objetivo se leen varios ficheros de configuración en el directorio inicial del usuario (no hay soporte para configuraciones globales todavía). El programa gda-manager puede usarse para visualizar estos ficheros y editar su contenido.

También contiene funciones utilidad para ayudarle a afrontar diferentes tareas relacionadas con las bases de datos. Esto incluye ejecución de comandos en modo "batch" (transacciones), acceso a proveedores Gda, configuración de acceso por medio del sistema de "plug-in" de gda-manager, ...

El principal concepto de la librería cliente es que el proveedor le permite conectarse a varias fuentes de información. Este proveedor es un servidor Corba implementado con Orbit. Un proveedor proporciona una factoría de interfaces que permiten al cliente solicitar conexiones no inicializadas al proveedor. La activación del proveedor se realiza a través del la librería Gnorba. Esta librería gestiona la identificación por el servidor de nombres Gnome y el registro del servidor Corba (el proveedor Gda), una vez que esta activo. Gnorba también permite al cliente elegir, si para cada instancia de la factoría se debe arrancar un nuevo servidor, o si se debería usar un servidor ya en ejecución. Este tiene repercusiones en el tiempo de arranque usado y en el numero de procesos conectados al servidor de base de datos (y por tanto al numero de licencias usado).

Con el fin de conseguir una conexión a una fuente de datos, el cliente debe usualmente proporcionar cierta información al proveedor. En primer lugar debe proporcionar el nombre de la fuente de datos al proveedor. El proveedor usa entonces este nombre en su modo especifico para localizar las partes desconocidas necesarias para establecer una conexión con el autentico servidor de datos. Esto puede ser un nombre de maquina y un numero de puerto, el nombre de una librería compartida que deba cargarse, o cualquier cosa que se necesite para acceder el sistema de base de datos o la fuente de datos. Adicionalmente el cliente debe suministrar un nombre de usuario y una clave para ser autenticados por el servidor de base de datos. El propio proveedor no realiza ningún tipo de pruebas de autenticación. Observe también que la conexión con el cliente no esta encriptada, así que los datos se mandan en claro. Esto cambiara cuando Orbit soporte comunicaciones encriptadas entre el servidor Corba y sus clientes. Si esta circunstancia no es aceptable, use el proveedor como librería compartida.

Una vez que la conexión haya sido abierta, el cliente puede solicitar información de diccionarios de datos a la fuente de datos. La semántica de los datos devueltos por estas peticiones esta muy ligada al proveedor y normalmente no se puede migrar de una fuente de datos a otra. Si el proveedor pertenece a la misma clase (base de datos por ejemplo), esta información podrá ser intercambiable, pero no espere que un mismo programa cliente interrogue a un proveedor Ldap o a un proveedor Odbc sin cambios en el código fuente del cliente.

El cliente puede entonces crear objetos Gda_commands que se usaran para realizar las verdaderas consultas a los proveedores. Para los proveedores Obdc y Mysql la consulta es la misma sentencia Sql. Por supuesto el proveedor es libre de cambiar las sentencias (conversión entre dialectos Sql) y algunos proveedores pueden incluso especificar su propio lenguaje (el proveedor Ldap haría esto). Es también posible llamar a procedimientos almacenados. Estos procedimientos pueden guardarse en el servidor (el sistema de base de datos) o pueden estar implementados por el proveedor. Otra posibilidad es enviar únicamente el nombre de una tabla al proveedor. El proveedor devolverá todas las filas y columnas de la tabla al cliente.

El resultado de una consulta se accede a través de un objeto de tipo Gda_Recorset. Los registros de datos actuales pueden almacenarse en el lado del cliente, el lado del servidor o en el servidor de base de datos. Esto depende de varios indicadores y opciones usados cuando se crea el juego de registros. Las posiciones en el juego de registros se definen a través de señaladores, que son tipos de datos ocultos. Un señalador puede usarse para situar el puntero actual de acceso en el juego de registros. Debido a que algunos sistemas de bases de datos no permiten situar de forma arbitraria el puntero de acceso en el conjunto de registros de una consulta, el proveedor almacena y "cachea" todos los registros de datos recibidos. Si el conjunto de registros esta configurado para almacenar los datos en el lado del cliente, entonces el señalador es simplemente un puntero a los registros. Puede leer mas acerca de como almacenar registros de datos, la cache y las transacciones en la sección dedicada al los conjuntos de registros.

5.2. Presentación de los objetos Gda

La principal clase de Gda es la clase Gda_Connection. Los objetos de esta clase establecen la conexión con el servidor Corba y se necesitan para casi cualquier actividad en el lado del cliente. Cada objeto tiene un puntero directo o indirecto a un objeto de conexión.

Otra clase muy importante el la clase Gda_Command. Se usa para almacenar el texto de un comando y para ejecutar este comando en el servidor Corba. Dispone de una referencia a un objeto conexión. La función de ejecución devuelve un objeto de la clase Gda_Recordset. Esta clase es responsable de "bufferear y cachear" y manejar la memoria de los ítem de datos devueltos por el servidor.

Los objetos Gda_Recorset no tienen objetos correspondientes en su servidor Corba. Si se llama a la función gda_recordset_move(), un objeto Gda_Command, que puede construir de forma implícita el objeto Gda_Recordset, se usara para comunicar con el servidor y recibir los registros de datos.

5.3. Tipos de datos

Un problema que Gda trata de ocultar al usuario es la diferencia entre los tipos de datos tal y como son almacenados en la base de datos y como son utilizados pro la aplicación. Un ejemplo sencillo es la representación de precios. Son números de tipo coma flotante con dos dígitos después del punto. Muchas bases de datos permiten especificar un tipo de dato decimal con una determinada longitud y una precisión. En C no existe este tipo de dato. Puede querer almacenar este valor como un texto, que es a menudo adecuado para la presentación, pero no para calcular tasas y otras cosas. Gda separa por tanto el tipo de almacenamiento de una columna del tipo deseado por el usuario. Esto se realiza a través del GDAType de un valor. Se usa la capacidad de conversión entre varios tipos de datos de la base de datos si esta disponible. En otro caso la conversión de datos se realiza en el cliente. Convertir los datos en el servidor puede conducir a resultados impredecibles cuando el servidor es una maquina de 64 bit y el cliente es una maquina de 32 bit.

5.3.1. Tipos de datos disponibles

A continuación se muestra una lista de los tipos de datos Gda, sus equivalentes C y Sql, y las conversiones permitidas.

5.4. Meta-datos de las fuentes de datos

Todas las fuentes de datos deben proporcionar funciones para acceder a su meta-datos. Si la fuente de datos es una base de datos, estos meta-datos son los nombres de las tablas, las columnas, derechos de usuarios, y cosas así. El resultado de una consulta de meta información se representa como Gda_Recordset. El problema es que la mayor parte del tiempo el cliente solo esta interesado en alguna parte de esta información. El cliente quiere consultar a la fuente de datos acerca de los índices de una tabla, no todos los índices de todas las tablas. Por lo tanto es posible enviar restricciones a las consultas de meta-datos. Pero esto es sumamente complicado debido al hecho de que cada consulta deberá tener sus propias restricciones. La siguiente lista muestra que tipos de consulta están implementados y que restricciones son validas para cada consulta. Las restirciones de envían como parejas nombre-valor. El nombre es de tipo enumerado y el valor es una cadena de texto. La función de consulta es una función con numero de argumentos variable y la lista de argumentos debe finalizar con un variable enumerada de valor 0.

La siguiente tabla muestra las restricciones mas corrientes y cuando usarlas. algunos esquemas pueden requerir que entregue una restricción.

Tabla 5-1. Significado de las restricciones estándar

Restricción	Uso	Observaciones
GDA_Connection_OBJECT_CATALOG	Usada para especificar la base de datos	
GDA_Connection_OBJECT_SCHEMA	Usada para especificar el propietario	
GDA_Connection_OBJECT_NAME	Usada para indicar el nombre del objeto a consultar (tabla, ...)	
GDA_Connection_EXTRA_INFO	Activela para obtener información detallada del proveedor	Dele como valor una cadena no nula (eje. "")

La siguiente tabla muestra los esquemas estándar que debe soportar cada proveedor Gda, aunque algún proveedor especifico puede no soportar alguno de ellos. Para comprobar si se soporta un esquema, vea la función gda_connection_supports().

Tabla 5-2. Esquemas estándar y restricciones soportadas

Tipos de objeto	Identificador	Restricciones soportadas	Campos devueltos	Información adicional
Tablas	GDA_Connection_GDCN_SCHEMA_TABLES	GDA_Connection_OBJECT_NAME, GDA_Connection_OBJECT_CATALOG, GDA_Connection_OBJECT_SCHEMA, GDA_Connection_EXTRA_INFO	nombre, comentarios	nombre, propietario, comentarios, definición Sql
Tabla padre (para los proveedores que soportan la herencia de tablas)	GDA_Connection_GDCN_SCHEMA_TAB_PARENTS	GDA_Connection_OBJECT_NAME (required), GDA_Connection_OBJECT_CATALOG	Nombre, orden de la herencia	No soportada
Vistas	GDA_Connection_GDCN_SCHEMA_VIEWS	GDA_Connection_OBJECT_NAME, GDA_Connection_OBJECT_CATALOG, GDA_Connection_OBJECT_SCHEMA, GDA_Connection_EXTRA_INFO	nombre, comentarios	nombre, propietario, comentarios, definición Sql
Columnas de la tabla (o vista)	GDA_Connection_GDCN_SCHEMA_COLUMNS	GDA_Connection_OBJECT_NAME (required), GDA_Connection_OBJECT_CATALOG, GDA_Connection_OBJECT_SCHEMA, GDA_Connection_COLUMN_NAME	nombre, tipo, tamaño, precisión, posible nulo, es clave, valor por omisión, comentarios	No soportado
Secuencias	GDA_Connection_GDCN_SCHEMA_SEQUENCES	GDA_Connection_OBJECT_NAME, GDA_Connection_OBJECT_CATALOG, GDA_Connection_OBJECT_SCHEMA, GDA_Connection_EXTRA_INFO	nombre, comentarios	nombre, propietarios, comentarios, definición Sql
Procedimientos	GDA_Connection_GDCN_SCHEMA_PROCS	GDA_Connection_OBJECT_NAME, GDA_Connection_OBJECT_CATALOG, GDA_Connection_OBJECT_SCHEMA, GDA_Connection_EXTRA_INFO	Nombre, Id del objeto, comentarios	Nombre, Id del objeto, propietario, comentarios, numero de argumentos, definición Sql
Parametros del procedimiento	GDA_Connection_GDCN_SCHEMA_PROC_PARAMS	GDA_Connection_OBJECT_NAME (required)	Uso (entrada, salida o E/S), tipo	No soportado
Agregados	GDA_Connection_GDCN_SCHEMA_AGGREGATES	GDA_Connection_OBJECT_NAME, GDA_Connection_OBJECT_CATALOG, GDA_Connection_OBJECT_SCHEMA, GDA_Connection_EXTRA_INFO	Nombre, Id del objeto, tipo de entrada, comentarios	Nombre, Id del objeto, tipo de entrada, comentarios, definición Sql
Tipos	GDA_Connection_GDCN_SCHEMA_PROV_TYPES	GDA_Connection_OBJECT_NAME, GDA_Connection_OBJECT_CATALOG, GDA_Connection_OBJECT_SCHEMA, GDA_Connection_EXTRA_INFO	Nombre, propietario, comentarios, tipo Gda, tipo de proveedor	nombre, comentarios

Debe prestar especial atención a las restiriciones usadas (tanto en las aplicaciones cliente, como en los proveedores Gda), ya que es un requisito impuesto a los proveedores devolver un error si se envía una restricción invalida al servidor. Esto es especialmente importante, ya que hay esquemas que pueden significar diferentes cosas en función del conjunto de restricciones usadas. Como puede imaginar, esto puede llevar aq ue el cliente reciba datos no solicitados.

5.5. Trabajos en modo batch

La librería cliente proporciona un objeto muy interesante: el objeto Gda_Batch, que es una clase abstracta que permite tratar una serie de comando como una única entidad. Ofrece esta funcionalidad conjuntamente con un autentico soporte de transacciones (si la base de datos lo soporta). Todos los cambios realizados en la base de datos son afianzados si no se encuentra errores, sin embargo la ejecución se cancela (y todos los cambios se descartan) si la base de datos devuelve un error. Este modelo permite proporcionar soporte a algo similar a las transacciones en sus aplicaciones.

Además, el objeto Gda_Batch es suficientemente flexible para usarse como un simple trabajo batch, ignorando los errores si esto se ajusta a sus necesidades, y deshabilitar las transacciones de la base de datos. En este caso, la mejor forma de usar el objeto es capturar sus diferentes señales, que le permiten conocer en cualquier momento el estado de la ejecución de cada comando.

5.6. Referencia de funciones

Las siguientes secciones describen las funciones disponibles para losa usuarios de la librería cliente Gda. Algunos de los elementos de esta librería, en especial los enumerados, están importados del directorio de fuentes IDL.

gda_batch_new

Nombre

gda_batch_new —

Synopsis

Gda_Batch * gda_batch_new (void);

Arguments

void

– undescribed –

Description

Creates a new #Gda_Batch object, which can be used in applications to simulate a transaction, that is, a series of commands which will be committed only if only all of them succeed. If any of the commands return an error when executed, all the changes are rolled back (by calling #gda_connection_rollback_transaction).

Although, this behavior is configurable. You can also use it as a way of sending several commands to the underlying database, regardless of the errors found in the process.

Returns

a pointer to the new object, or NULL on error

gda_batch_free

Nombre

gda_batch_free —

Synopsis

```
void gda_batch_free (Gda_Batch * job);
```

Arguments

job
: a #Gda_Batch object

Description

Destroy the given batch job object

gda_batch_load_file

Nombre

gda_batch_load_file —

Synopsis

gboolean gda_batch_load_file (Gda_Batch * job, const gchar * filename, gboolean clean);

Arguments

job

    a #Gda_Batch object

filename

    file name

clean

    clean up

Description

Load the given file as a set of commands into the given job object. The clean parameter specifies whether to clean up the list of commands before loading the given file or not.

Returns

TRUE if successful, or FALSE on error

gda_batch_add_command

Nombre

gda_batch_add_command —

Synopsis

```
void gda_batch_add_command (Gda_Batch * job, const gchar * cmd);
```

Arguments

job

a #Gda_Batch object

cmd

command string

Description

Adds a command to the list of commands to be executed

gda_batch_clear

Nombre

gda_batch_clear —

Synopsis

```
void gda_batch_clear (Gda_Batch * job);
```

Arguments

job

– undescribed –

job

a #Gda_Batch object

Clears the given #Gda_Batch object. This means eliminating the list of commands

gda_batch_start

Nombre

gda_batch_start —

Synopsis

```
gboolean gda_batch_start (Gda_Batch * job);
```

Arguments

job
: a #Gda_Batch object

Description

Start the batch job execution. This function will return when the series of commands is completed, or when an error is found

Returns

TRUE if all goes well, or FALSE on error

gda_batch_stop

Nombre

gda_batch_stop —

Synopsis

```
void gda_batch_stop (Gda_Batch * job);
```

Arguments

job

a #Gda_Batch object

Description

Stop the execution of the given #Gda_Batch object. This cancels the the transaction (discarding all changes) if the #Gda_Batch object is in transaction mode

gda_batch_is_running

Nombre

gda_batch_is_running —

Synopsis

```
gboolean gda_batch_is_running (Gda_Batch * job);
```

Arguments

job

a #Gda_Batch object

gda_batch_get_connection

Nombre

gda_batch_get_connection —

Synopsis

Gda_Connection * gda_batch_get_connection (Gda_Batch * job);

Arguments

job
: a #Gda_Batch object

Description

Return the #Gda_Connection object associated with the given batch job

gda_batch_set_connection

Nombre

gda_batch_set_connection —

Synopsis

```
void gda_batch_set_connection (Gda_Batch * job, Gda_Connection * cnc);
```

Arguments

job
: a #Gda_Batch object

cnc
: a #Gda_Connection object

Description

Associate a #Gda_Connection object to the given batch job

gda_batch_get_transaction_mode

Nombre

gda_batch_get_transaction_mode —

Synopsis

```
gboolean gda_batch_get_transaction_mode (Gda_Batch * job);
```

Arguments

job
: a #Gda_Batch object

Description

Returns the transaction mode for the given #Gda_Batch object. This mode specifies how the series of commands are treated. If in transaction mode (TRUE), the execution is stopped whenever an error is found, discarding all changes, whereas all data is committed if no error is found.

On the other hand, if transaction mode is disabled, the execution continues until the end regardless of any error found.

gda_batch_set_transaction_mode

Nombre

gda_batch_set_transaction_mode —

Synopsis

```
void gda_batch_set_transaction_mode (Gda_Batch * job, gboolean mode);
```

Arguments

job

a #Gda_Batch object

mode

transaction mode

Description

Enable/disable transaction mode for the given #Gda_Batch object. Transaction mode is enabled by default

gda_command_new

Nombre

gda_command_new —

Synopsis

Gda_Command * gda_command_new (void);

Arguments

void
   – undescribed –

Description

This function allocates the memory for a command object.

Returns

a pointer to the command object

gda_command_free

Nombre

gda_command_free —

Synopsis

```
void gda_command_free (Gda_Command* cmd);
```

Arguments

cmd
: The command object which should be deallocated.

Description

This function frees the memory of command object and cuts the association with its connection object.

gda_command_set_connection

Nombre

gda_command_set_connection —

Synopsis

```
gint gda_command_set_connection (Gda_Command* cmd, Gda_Connection* cnc);
```

Arguments

cmd
: The command object

cnc
: The connection object

Description

Associates a connection with a command. All functions with this command will use the connection to talk to the data source. If the command is already associated with a connection, this association is destroyed.

Returns

-1 on error, 0 on success

gda_command_get_connection

Nombre

gda_command_get_connection —

Synopsis

```
Gda_Connection * gda_command_get_connection (Gda_Command* cmd);
```

Arguments

cmd
: the command object

Description

Returns the #gda_Connection object which is used by the command.

Returns

a pointer to the #gda_Connection object

gda_command_set_text

Nombre

gda_command_set_text —

Synopsis

```
void gda_command_set_text (Gda_Command* cmd, gchar* text);
```

Arguments

cmd

the #Gda_Command object

text

the command to perform. There are some special texts which are reckognized by the servers. See the server documantation for a list of special commands.

Description

Sets the command which is executed when the gda_command_execute() function is called.

gda_command_get_text

Nombre

gda_command_get_text —

Synopsis

gchar * gda_command_get_text (Gda_Command* cmd);

Arguments

cmd

the #Gda_Command object

Description

Gets the command string which is executed when the gda_command_execute() function is called.

Returns

a reference to the command string.

gda_command_set_cmd_type

Nombre

gda_command_set_cmd_type —

Synopsis

```
void gda_command_set_cmd_type (Gda_Command* cmd, gulong type);
```

Arguments

cmd

the #Gda_Command object

type

the type of the command. See the provider specification which command type is understood and the semantics of the different types.

Description

Sets the command which is executed when the gda_command_execute() function is called.

gda_command_get_cmd_type

Nombre

gda_command_get_cmd_type —

Synopsis

```
gulong gda_command_get_cmd_type (Gda_Command* cmd);
```

Arguments

cmd
: the #Gda_Command object

Description

Gets the type of the command

Returns

the type of the command

gda_command_execute

Nombre

gda_command_execute —

Synopsis

Gda_Recordset * gda_command_execute (Gda_Command* cmd, gulong* reccount,
gulong flags);

Arguments

cmd

the command object

reccount

a pointer to a gulong variable. In this variable the number of affected records is stored.

flags

flags to categorize the command.

Description

This function executes the command which has been set with gda_command_set_text(). It returns a #Gda_Recordset pointer which holds the results from this query. If the command doesn't return any results, like insert, or updaste statements in SQL, an empty result set is returnd.

Returns

a pointer to a recordset or a NULL pointer if there was an error.

gda_command_create_parameter

Nombre

gda_command_create_parameter —

Synopsis

```
void gda_command_create_parameter (Gda_Command* cmd, gchar* name,
GDA_ParameterDirection inout, GDA_Value* value);
```

Arguments

cmd

the command object

name

the name of the parameter

inout

direction of the parameter

value

value of the parameter

Description

This function creates a new parameter for the command. This is important if you want to use parameterized statements like "select * from table where key = ?" and substitute the value for key at a later time. It also comes handy if you don't want to convert each parameter to it's string representation (floating point values).

The inout parameter defines if the value parameter is used to pass a value to the statement or to receive a value. In the exampel abov the input parameter will have a value of #PARAM_IN. The value #PARAM_OUT is used for output parameters when the command executes a precoedure.

value points to a GDA_Value variable (there will be helper functions to create such varbales from the basic C types). This variable must hold a valid value when the gda_command_execute() function is called and the inout parameter is either #PARAM_IN or #PARAM_INOUT. If the inoput parameter is

#PARAM_OUT, memory is allocated by the library and the user must free it using CORBA_free().

IMPORTANT

use the CORBA functions to allocate the memory for the value parameter if it's not a simple data type (integer, float, ...)

gda_command_get_timeout

Nombre

gda_command_get_timeout —

Synopsis

```
glong gda_command_get_timeout (Gda_Command * cmd);
```

Arguments

cmd

– undescribed –

Description

cmd the command object

This function returns the timeout associated with the given command object

gda_command_set_timeout

Nombre

gda_command_set_timeout —

Synopsis

void gda_command_set_timeout (Gda_Command * cmd, glong timeout);

Arguments

cmd

– undescribed –

timeout

– undescribed –

Description

cmd the command object timeout timeout

Sets the timeout for the given command

gda_connection_new

Nombre

gda_connection_new —

Synopsis

```
Gda_Connection* gda_connection_new (CORBA_ORB orb);
```

Arguments

orb
: the ORB

Description

Allocates space for a client side connection object

Returns

the pointer to the allocated object

gda_connection_free

Nombre

gda_connection_free —

Synopsis

```
void gda_connection_free (Gda_Connection* cnc);
```

Arguments

cnc

the connection

Description

If the connection is open the connection is closed and all associated objects (commands, recordsets, errors) are deleted. The memory is freed. The connection object cannot be used any more.

gda_connection_set_provider

Nombre

gda_connection_set_provider —

Synopsis

```
void gda_connection_set_provider (Gda_Connection* cnc, gchar* name);
```

Arguments

cnc

connection object

name

name of the provider

Description

Registers name as the provider for this connection. This should be the GOAD id of the CORBA server to use

gda_connection_get_provider

Nombre

gda_connection_get_provider —

Synopsis

```
const gchar* gda_connection_get_provider (Gda_Connection* cnc);
```

Arguments

cnc
: Connection object

Description

Retuns the provider used for this connection

Returns

a reference to the provider name

gda_connection_list_datasources

Nombre

gda_connection_list_datasources —

Synopsis

```
GList* gda_connection_list_datasources (Gda_Connection* cnc);
```

Arguments

cnc
: the connection object:

Description

List all datasources which can be used with this connection object.

Returns

a GList with the DSNs of all known data sources.

gda_connection_supports

Nombre

gda_connection_supports —

Synopsis

```
gboolean gda_connection_supports (Gda_Connection * cnc,
GDA_Connection_Feature feature);
```

Arguments

cnc

    the connection object

feature

    feature to be tested

Description

Return information on features supported by the underlying database system.

Returns

whether the requested feature is supported or not

gda_connection_set_default_db

Nombre

gda_connection_set_default_db —

Synopsis

```
void gda_connection_set_default_db (Gda_Connection* cnc, gchar* dsn);
```

Arguments

cnc
: the connection object

dsn
: the DSN of the default database

Description

Registers the DSN as the name of the default DB to access when the connection is opened

gda_connection_open

Nombre

gda_connection_open —

Synopsis

gint gda_connection_open (Gda_Connection* cnc, gchar* dsn, gchar* user, gchar* pwd);

Arguments

cnc

the connection object which describes the server and the database name to which the connection shuld be opened

dsn

The DSN of the database. Can be NULL.

user

The username for authentication to the database. Can be NULL.

pwd

The password for authentication to the database. Can be NULL.

Description

The function activates the correct CORBA server (defined with he #gda_connection_set_provider() function. Then it tries to open the database using the DSN or the default database as a data source. If user or pwd is not NULL, it will overwrite the appropriate entry in the DSN passed as par2. Entries in the DSN have the form = seperated from the database name . Currently the DSN is not parsed.

Returns

0 on success, -1 on error

gda_connection_close

Nombre

gda_connection_close —

Synopsis

```
void gda_connection_close (Gda_Connection* cnc);
```

Arguments

cnc

The connection object which should be closed

Description

Closes the connection object and all associated #Gda_Recordset objects. #Gda_Command objects are not closed, but can't be used anymore. Transactions pending on this objects are aborted and an error is returned.

Returns

0 on success, -1 on error

gda_connection_open_schema_array

Nombre

gda_connection_open_schema_array —

Synopsis

Gda_Recordset* gda_connection_open_schema_array (Gda_Connection* cnc, GDA_Connection_QType t, Gda_Constraint_Element* elems);

Arguments

cnc

Connection object

t

Query type

elems

Constraint values

Description

Retrieves meta data about the data source to which the connection object is connected. The elems is an array of Constraint_Element item. The last item must have it's type element set to GDA_Connection_no_CONSTRAINT If elems is NULL, no constraints are used.

Returns

The recordset which holds the results.

gda_connection_open_schema

Nombre

gda_connection_open_schema —

Synopsis

```
Gda_Recordset* gda_connection_open_schema (Gda_Connection* cnc,
GDA_Connection_QType t, ...);
```

Arguments

cnc

Connection object

t

Query type

...

– undescribed –

Description

Retrieves meta data about the data source to which the connection object is connected. The constraints is a list of enum/string pairs. The end of the list must be marked by the special value GDA_Connection_no_CONSTRAINT

Returns

The recordset which holds the results.

gda_connection_get_errors

Nombre

gda_connection_get_errors —

Synopsis

```
GList* gda_connection_get_errors (Gda_Connection* cnc);
```

Arguments

cnc

    the connection object

Description

Returns a list of all errors for this connection object. This function also clears the error list. The errors are stored in LIFO order, so the last error which happend is stored as the first element in the list. Advancing the list means to get back in time and retrieve earlier errors.

The farther away from the client the error happens, the later it is in the queue. If an error happens on the client and then in the CORBA layer and then on the server, you will get the server error first.

Returns

a #Glist holding the error objects.

gda_connection_begin_transaction

Nombre

gda_connection_begin_transaction —

Synopsis

```
gint gda_connection_begin_transaction (Gda_Connection* cnc);
```

Arguments

cnc
: the connection object

Description

Start a new transaction on cnc

Returns

0 if everything is okay, -1 on error

gda_connection_commit_transaction

Nombre

gda_connection_commit_transaction —

Synopsis

```
gint gda_connection_commit_transaction (Gda_Connection* cnc);
```

Arguments

cnc
: the connection object

Description

Commit a pending transaction on cnc

Returns

0 if everything is okay, -1 on error

gda_connection_rollback_transaction

Nombre

gda_connection_rollback_transaction —

Synopsis

```
gint gda_connection_rollback_transaction (Gda_Connection* cnc);
```

Arguments

cnc
: the connection object

Description

Abort and rollback a pending transaction on cnc

Returns

0 if everything is okay, -1 on error

gda_connection_execute

Nombre

gda_connection_execute —

Synopsis

Gda_Recordset* gda_connection_execute (Gda_Connection* cnc, gchar* txt, gulong* reccount, gulong flags);

Arguments

cnc

the connection

txt

the statement to execute

reccount

a pointer to a gulong. The number of affected records is stored in the space where the pointer points to

flags

falgs (currently unused

Description

Executes txt. An internal Gda_Command object is used for execution. This command object is destroyed automatically after the commadn has finished.

Returns

a recordset on success or NULL on error.

gda_connection_start_logging

Nombre

gda_connection_start_logging —

Synopsis

```
gint gda_connection_start_logging (Gda_Connection* cnc, gchar* filename);
```

Arguments

cnc
: the connection

filename
: the name of the logfile

Description

This function causes the server to log the statements it executes in filename.

Returns

-1 on error and 0 if everything is okay

gda_connection_stop_logging

Nombre

gda_connection_stop_logging —

Synopsis

```
gint gda_connection_stop_logging (Gda_Connection* cnc);
```

Arguments

cnc
: the connection

Description

This function stops the database server logs

Returns

-1 on error and 0 if everything is okay

gda_error_free

Nombre

gda_error_free —

Synopsis

```
void gda_error_free (Gda_Error* error);
```

Arguments

error
: the error object.

Description

Frees the memory allocated by the error object.

gda_error_list_free

Nombre

gda_error_list_free —

Synopsis

```
void gda_error_list_free (GList* errors);
```

Arguments

errors

a glist holding error objects.

Description

Frees all error objects in the list and the list itself. After this function has been called, the errors parameter doesn't point to valid storage any more.

gda_fieldtype_2_string

Nombre

gda_fieldtype_2_string —

Synopsis

```
gchar* gda_fieldtype_2_string (gchar* bfr, gint length, GDA_ValueType type);
```

Arguments

bfr

bufferspace where the printed name of the type of the field is placed into.

length

the size of the buffer

type

– undescribed –

Returns

a pointer to the buffer.

gda_string_2_fieldtype

Nombre

gda_string_2_fieldtype —

Synopsis

GDA_ValueType gda_string_2_fieldtype (gchar * type);

Arguments

type

– undescribed –

Description

type GDA type string

Returns the GDA type identifier for the given string (returned by gda_fieldtype_2_string

gda_field_new

Nombre

gda_field_new —

Synopsis

```
Gda_Field * gda_field_new ( void);
```

Arguments

void
: – undescribed –

Description

Allocate space for a new field object.

Returns

the pointer to the new field object.

gda_field_free

Nombre

gda_field_free —

Synopsis

```
void gda_field_free (Gda_Field* f);
```

Arguments

f
: a pointer to a Gda_Field object

Description

Free the memory allocated for this field object.

gda_field_actual_size

Nombre

gda_field_actual_size —

Synopsis

```
gint gda_field_actual_size (Gda_Field* f);
```

Arguments

f
: a pointer to the field.

Description

Calculates the number of bytes the value fo the field needs.

Returns

the number of bytes the value fo the field needs or 0 if the field has a NULL value.

gda_recordset_new

Nombre

gda_recordset_new —

Synopsis

```
GtkObject* gda_recordset_new ( void);
```

Arguments

void

– undescribed –

Description

Allocates space for a new recordset.

Returns

the allocated recordset object

gda_recordset_free

Nombre

gda_recordset_free —

Synopsis

```
void gda_recordset_free (Gda_Recordset* rs);
```

Arguments

rs
: the recordset which should be destroyed.

Description

This function frees all memory allocated by the recordset and destroys all associations with commands and connections.

gda_recordset_close

Nombre

gda_recordset_close —

Synopsis

```
void gda_recordset_close (Gda_Recordset* rs);
```

Arguments

rs
: the recordset to close

Description

This function closes the recordset and frees the memory occupied by the actual data items. The recordset can be opened again, doing the same query as before. It is guaranteeed that the data cached by the recordset is refetched from the server. Use this function is some characteristics of the recordset must be changed

gda_recordset_bof

Nombre

gda_recordset_bof —

Synopsis

```
gint gda_recordset_bof (Gda_Recordset* rs);
```

Arguments

rs

Recordset which should be checked

Description

This function is used to check if the recordset cursor is beyond the first row. If this function returns TRUE any of the functions which actually returns the value of a field an error is returned because the cursor doesn't point to a row.

Returns

1 if the cursor is beyond the first record or the recordset is empty, 0 otherwise.

gda_recordset_eof

Nombre

gda_recordset_eof —

Synopsis

```
gint gda_recordset_eof (Gda_Recordset* rs);
```

Arguments

rs

Recordset which should be checked

Description

This function is used to check if the recordset cursor is after the last row. If this function returns TRUE any of the functions which actually returns the value of a field an error is returned because the cursor doesn't point to a row.

Returns

1 if the cursor is after the last record or the recordset is empty, 0 otherwise.

gda_recordset_move

Nombre

gda_recordset_move —

Synopsis

```
gulong gda_recordset_move (Gda_Recordset* rs, gint count, gpointer bookmark);
```

Arguments

rs

the recordset

count

The number of records to skip

bookmark

if not NULL, the cursor is positioned relative to the record described by this paramter. seee gda_recordset_get_bookmark() how to get bookmark values for records.

Description

Moves the cursor of the recordset forward or backward. count is the number of records to move. If count is negative the cursor is moved towards the beginning. The function causes the recordset to actually fetch records from the data source. Each fetch from the data source fetches #cachesize rows in one turn. A maximum of #maxrows rows can be fetched.

If the cursor is on the second row and the count parameter is -10, then the cursor is position in front of the first record available. gda_rcordset_bof() will return TRUE and the return value of the function is two, because the cursor actually moved two records.

Returns

the number of the record the cursor is addressing after the move or GDA_RECORDSET_INVALID_POSITION if there was an error fetching the rows.

gda_recordset_move_first

Nombre

gda_recordset_move_first —

Synopsis

```
gulong gda_recordset_move_first (Gda_Recordset* rs);
```

Arguments

rs
: the recordset

Description

Moves the cursor of the recordset to the first record.

If the cursor is already on the the first record nothing happen.

Returns

the position of the cursor, or GDA_RECORDSET_INVALID_POSITION if there was an error.

gda_recordset_move_last

Nombre

gda_recordset_move_last —

Synopsis

```
gulong gda_recordset_move_last (Gda_Recordset* rs);
```

Arguments

rs

the recordset

Description

Moves the cursor of the recordset to the last record.

If the cursor is already on the the last record nothing happen.

Returns

the position of the cursor, or GDA_RECORDSET_INVALID_POSITION if there was an error.

gda_recordset_move_next

Nombre

gda_recordset_move_next —

Synopsis

```
gulong gda_recordset_move_next (Gda_Recordset* rs);
```

Arguments

rs
: the recordset

Description

Moves the cursor of the recordset to the next record.

Has the same effect as calling gda_recordset_move() with count set to 1.

Returns

the position of the cursor, or GDA_RECORDSET_INVALID_POSITION if there was an error.

gda_recordset_field_idx

Nombre

gda_recordset_field_idx —

Synopsis

```
Gda_Field* gda_recordset_field_idx (Gda_Recordset* rs, gint idx);
```

Arguments

rs
: the recordset

idx
: the index of the field in the current row

Description

Returns a pointer to the field at position idx of the current row of the recordset.

Returns

a pointer to a field structor or NULL if idx is out of range

gda_recordset_field_name

Nombre

gda_recordset_field_name —

Synopsis

Gda_Field * gda_recordset_field_name (Gda_Recordset* rs, gchar* name);

Arguments

rs

the recordset

name

– undescribed –

Description

Returns a pointer to the field with the name name of the current row of the recordset.

Returns

a pointer to a field structor or NULL if idx is out of range

gda_recordset_rowsize

Nombre

gda_recordset_rowsize —

Synopsis

gint gda_recordset_rowsize (Gda_Recordset* rs);

Arguments

rs
: the recordset

Description

Returns the number of fields in a row of the current recordset

Description

Returns the number of fields in a row of the current recordset

Returns

the number of rows in the recordset or 0 if there was an error.

gda_recordset_affected_rows

Nombre

gda_recordset_affected_rows —

Synopsis

```
gulong gda_recordset_affected_rows (Gda_Recordset * rs);
```

Arguments

rs
: the recordset

Description

Return the number of affected rows in the recordset

gda_recordset_set_connection

Nombre

gda_recordset_set_connection —

Synopsis

```
gint gda_recordset_set_connection (Gda_Recordset* rs, Gda_Connection* cnc);
```

Arguments

rs
: the recordset

cnc
: the conneciton

Description

Associates a recordset with a connection. This function must not be called on a already opened recordst. This function must be called before any of the the gda_recordset_open_txt() or gda_recordset_open_cmd() functions is called.

Returns

0 if everyhting is okay, -1 on error

gda_recordset_get_connection

Nombre

gda_recordset_get_connection —

Synopsis

Gda_Connection* gda_recordset_get_connection (Gda_Recordset* rs);

Arguments

rs
: the rcordset

Returns

the connection object used with this recordset. NULL if the recordset isn't open or there's no connection assoctiated with this recordset.

gda_recordset_open

Nombre

gda_recordset_open —

Synopsis

gint gda_recordset_open (Gda_Recordset* rs, Gda_Command* cmd,
GDA_CursorType cursor_type, GDA_LockType lock_type, gulong options);

Arguments

rs

the recordset

cmd

the command object

cursor_type

the type of the cursor used. Currently only a forward only cursor is implemented

lock_type

the type of locks used for the command. Currently only readonly type of locking is supported.

options

currently not implemented. Used to identify how to parse the txt.

Description

This function opens a recordset. The recordset is filled with the output of cmd. Before this function is called various parameters of the recordst might be changed. This is the most featurefull function for retrieving results from a database.

Returns

0 if everytrhing is okay, -1 on error

gda_recordset_open_txt

Nombre

gda_recordset_open_txt —

Synopsis

gint gda_recordset_open_txt (Gda_Recordset* rs, gchar* txt, GDA_CursorType cursor_type, GDA_LockType lock_type, gulong options);

Arguments

rs

the recordset

txt

the command text

cursor_type

the type of the cursor used. Currently only a forward only cursor is implemented

lock_type

the type of locks used for the command. Currently only readonly type of locking is supported.

options

currently not implemented. Used to identify how to parse the txt.

Description

This function opens a recordset. The recordset is filled with the output of txt. Before this function is called various parameters of the recordst might be changed. This is the most featurefull function for retrieving results from a database.

Returns

0 if everytrhing is okay, -1 on error

gda_recordset_get_cursorloc

Nombre

gda_recordset_get_cursorloc —

Synopsis

```
GDA_CursorLocation gda_recordset_get_cursorloc (Gda_Recordset* rs);
```

Arguments

rs
: the recordset

Returns

the current value of the cursor location attriburte

gda_recordset_set_cursorloc

Nombre

gda_recordset_set_cursorloc —

Synopsis

```
void gda_recordset_set_cursorloc (Gda_Recordset* rs, GDA_CursorLocation loc);
```

Arguments

rs

the recordset

loc

the cursor location

Description

Set the cursor location attribute to loc.

gda_recordset_get_cursortype

Nombre

gda_recordset_get_cursortype —

Synopsis

```
GDA_CursorType gda_recordset_get_cursortype (Gda_Recordset* rs);
```

Arguments

rs
: the recordset

Returns

the current value of the cursor type attriburte

gda_recordset_set_cursortype

Nombre

gda_recordset_set_cursortype —

Synopsis

```
void gda_recordset_set_cursortype (Gda_Recordset* rs, GDA_CursorType type);
```

Arguments

rs
: the recordset

type
: the cursor type

Description

Set the cursor type attribute to type.

gda_server_new

Nombre

gda_server_new —

Synopsis

```
Gda_Server * gda_server_new ( void);
```

Arguments

void

– undescribed –

Description

Allocates memory for a new Gda_Server object and initializes struct members.

Returns

a pointer to a new Gda_Server object.

gda_server_copy

Nombre

gda_server_copy —

Synopsis

Gda_Server * gda_server_copy (Gda_Server* server);

Arguments

server
    the server to be copies.

Description

Make a deep copy of all the data needed to describe a gGDA server.

Description

Make a deep copy of all the data needed to describe a gGDA server.

Returns

a pointer to the newly allocated server object

gda_server_free

Nombre

gda_server_free —

Synopsis

void gda_server_free (Gda_Server* server);

Arguments

server

the server to de-allocate.

Description

Frees the memory allocated with gda_server_new() and the memory allocated to struct members.

Description

Frees the memory allocated with gda_server_new() and the memory allocated to struct members.

gda_server_list

Nombre

gda_server_list —

Synopsis

GList * gda_server_list (void);

Arguments

void

– undescribed –

Description

Searches the GNORBA database for GDA servers and returns a Glist of Gda_Server structs.

Returns

a GList of GDA servers structs

gda_server_free_list

Nombre

gda_server_free_list —

Synopsis

```
void gda_server_free_list (GList * list);
```

Arguments

list
: list of #Gda_Server structures

Description

Frees a list of #Gda_Server structures previously returned by a call to #gda_server_list

gda_server_find_by_name

Nombre

gda_server_find_by_name —

Synopsis

```
Gda_Server * gda_server_find_by_name (const gchar * provider);
```

Arguments

provider

provider name

Description

Returns a #Gda_Server structure describing the given provider. This function searches all the servers present on your system and tries to find the specified provider.

Returns

a pointer to the server structure, or NULL on error

gda_list_datasources

Nombre

gda_list_datasources —

Synopsis

```
GList * gda_list_datasources ( void);
```

Arguments

void
: – undescribed –

Description

Lists all datasources configured on the system.

Returns a GList with the names of all data sources configured.

gda_list_datasources_for_provider

Nombre

gda_list_datasources_for_provider —

Synopsis

```
GList * gda_list_datasources_for_provider (gchar* provider);
```

Arguments

provider
: the provider which should be used to look for datasources

Returns

a GList of all datasources available to a specific provider.

gda_dsn_list

Nombre

gda_dsn_list —

Synopsis

GList * gda_dsn_list (void);

Arguments

void

  – undescribed –

Description

Returns a list of all available data sources. The returned value is a GList of #Gda_Dsn structures

gda_dsn_free

Nombre

gda_dsn_free —

Synopsis

```
void gda_dsn_free (Gda_Dsn * dsn);
```

Arguments

dsn

    data source

gda_dsn_free_list

Nombre

gda_dsn_free_list —

Synopsis

void gda_dsn_free_list (GList * list);

Arguments

list
  – undescribed –

gda_xml_database_new

Nombre

gda_xml_database_new —

Synopsis

Gda_XmlDatabase * gda_xml_database_new (const gchar * name);

Arguments

name

The name to represent the database

Description

Allocates memory for a new Gda_XmlDatabase object

Returns

a pointer to the newly allocated object, or NULL on error

gda_xml_database_new_from_file

Nombre

gda_xml_database_new_from_file —

Synopsis

```
Gda_XmlDatabase * gda_xml_database_new_from_file (const gchar * filename);
```

Arguments

filename

path to an existing file

Description

Loads a XML file containing the representation of a database

gda_xml_database_free

Nombre

gda_xml_database_free —

Synopsis

```
void gda_xml_database_free (Gda_XmlDatabase * db);
```

Arguments

db

a Gda_XmlDatabase object

Description

Cleans up all memory associated with the XML representation of the given database

gda_xml_database_save

Nombre

gda_xml_database_save —

Synopsis

```
gboolean gda_xml_database_save (Gda_XmlDatabase * db, const gchar * filename);
```

Arguments

db

a Gda_XmlDatabase object

filename

file name

Description

Saves the given XML database to a disk file. You may specify NULL for the filename parameter, in which case the current file name for the database is used

Returns

TRUE if successful, FALSE on error

gda_xml_database_changed

Nombre

gda_xml_database_changed —

Synopsis

```
void gda_xml_database_changed (Gda_XmlDatabase * db);
```

Arguments

db
: a Gda_XmlDatabase object

Description

Emits the "changed" signal for the given XML database. Applications do not need to call this function, since the library emits any needed signal for the Gda_XmlDatabase object.

gda_xml_database_get_name

Nombre

gda_xml_database_get_name —

Synopsis

```
const gchar * gda_xml_database_get_name (Gda_XmlDatabase * db);
```

Arguments

db

a Gda_XmlDatabase object

Description

Returns the name of the database

gda_xml_database_set_name

Nombre

gda_xml_database_set_name —

Synopsis

```
void gda_xml_database_set_name (Gda_XmlDatabase * db, const gchar * name);
```

Arguments

db

a Gda_XmlDatabase object

name

new name for the database

Description

Sets the name for the given database

gda_xml_database_get_date

Nombre

gda_xml_database_get_date —

Synopsis

```
const gchar * gda_xml_database_get_date (Gda_XmlDatabase * db);
```

Arguments

db
: a Gda_XmlDatabase object

Description

Returns a string representation of the last modification date. This date is updated each time the XML tree is modified

gda_xml_database_set_date

Nombre

gda_xml_database_set_date —

Synopsis

```
void gda_xml_database_set_date (Gda_XmlDatabase * db, const gchar * dt);
```

Arguments

db

a Gda_XmlDatabase object

dt

date string

Description

Sets the modification time of the given XML database. The format used for this date is up to the programmer, as no further operations are done on this date. Anyway, if you want to follow the same rules as the library, use the format as returned by strftime()

gda_xml_database_get_filename

Nombre

gda_xml_database_get_filename —

Synopsis

```
const gchar * gda_xml_database_get_filename (Gda_XmlDatabase * db);
```

Arguments

db

– undescribed –

db

a Gda_XmlDatabase object

Returns the filename associated with the given XML database representation. This filename will then be used when calling gda_xml_database_save

gda_xml_database_set_filename

Nombre

gda_xml_database_set_filename —

Synopsis

```
void gda_xml_database_set_filename (Gda_XmlDatabase * db, const gchar *
filename);
```

Arguments

db

a Gda_XmlDatabase object

filename

– undescribed –

Description

Associates a file name with the given XML database representation

gda_xml_database_add_object

Nombre

gda_xml_database_add_object —

Synopsis

Gda_XmlObject * gda_xml_database_add_object (Gda_XmlDatabase * db, const gchar * name, const gchar * type);

Arguments

db

    a Gda_XmlDatabase object

name

    name of the object

type

    type of the object

Description

Adds an object of the given type. It is better for application programmers to call the specific-object creation functions (gda_xml_table_new, gda_xml_view_new, ...), than calling this function directly. In any case, this function is public to allow you the choice

gda_xml_database_delete_object

Nombre

gda_xml_database_delete_object —

Synopsis

void gda_xml_database_delete_object (Gda_XmlDatabase * db, Gda_XmlObject * obj);

Arguments

db

a Gda_XmlDatabase object

obj

the object to be deleted

Description

Deletes the given object from the XML database

gda_xml_database_find_object

Nombre

gda_xml_database_find_object —

Synopsis

Gda_XmlObject * gda_xml_database_find_object (Gda_XmlDatabase * db, const gchar * name, const gchar * type);

Arguments

db

a Gda_XmlDatabase object

name

name of the object

type

type of the object

Description

Searches for a given object of a given type in the XML database specified.

Returns

a pointer to the object if found, or NULL

gda_xml_database_enum_objects

Nombre

gda_xml_database_enum_objects —

Synopsis

gboolean gda_xml_database_enum_objects (Gda_XmlDatabase * db,
Gda_XmlEnumFunc func, const gchar * type, gpointer user_data);

Arguments

db

a Gda_XmlDatabase object

func

user function

type

object type to be enumerated

user_data

– undescribed –

Description

This function calls the user-provided function for each object contained in the given XML database representation. You can specify which type of objects you want to get (by using the GDA_XML_OBJECT_TYPE_* #defines) or all objects of any type (by supplying NULL as the type parameter.

Returns

TRUE if successful, FALSE on error

gda_xml_object_add_child

Nombre

gda_xml_object_add_child —

Synopsis

Gda_XmlObject * gda_xml_object_add_child (Gda_XmlObject * obj, const gchar * type);

Arguments

obj

a #Gda_XmlObject

type

type of the child

Description

Adds a new child to the given #Gda_XmlObject. This function is mainly for library internal use, you should better use the specific object management functions

gda_xml_object_delete_child

Nombre

gda_xml_object_delete_child —

Synopsis

void gda_xml_object_delete_child (Gda_XmlObject * obj, Gda_XmlObject * child);

Arguments

obj

a #Gda_XmlObject

child

child object

Description

Removes the given #Gda_XmlObject child, freeing all associated memory. This function is recursive in that, it calls itself to remove the child object's children.

Again, you shouldn't use this function, but the specific object management functions instead

gda_xml_object_enum_children

Nombre

gda_xml_object_enum_children —

Synopsis

gboolean gda_xml_object_enum_children (Gda_XmlObject * obj,
Gda_XmlEnumFunc func, const gchar * type, gpointer user_data);

Arguments

obj

a #Gda_XmlObject

func

callback function

type

type of the children

user_data

data to pass to the callback

Description

This function enumerates all the children (or the children of type, if not NULL)
by calling the given callback function for each child object found

Returns

TRUE if successful, FALSE on error

gda_xml_object_get_property

Nombre

gda_xml_object_get_property —

Synopsis

```
const gchar * gda_xml_object_get_property (Gda_XmlObject * obj, const gchar *
prop);
```

Arguments

obj
: a #Gda_XmlObject

prop
: property name

Description

Returns the value for the given property on the specified object

gda_xml_object_set_property

Nombre

gda_xml_object_set_property —

Synopsis

void gda_xml_object_set_property (Gda_XmlObject * obj, const gchar * prop, const gchar * value);

Arguments

obj
: a #Gda_XmlObject

prop
: property name

value
: value to be set

gda_xml_object_enum_properties

Nombre

gda_xml_object_enum_properties —

Synopsis

```
gboolean gda_xml_object_enum_properties (Gda_XmlObject * obj,
Gda_XmlEnumPropFunc func, gpointer user_data);
```

Arguments

obj

a #Gda_XmlObject

func

callback function

user_data

data to be passed to callback

Description

Enums the properties defined for the given object by calling a user-supplied callback function

Returns

TRUE if success, FALSE on error

gda_xml_object_type

Nombre

gda_xml_object_type —

Synopsis

```
const gchar * gda_xml_object_type (Gda_XmlObject * obj);
```

Arguments

obj
: a #Gda_XmlObject

Description

Returns the type of the given object

gda_xml_table_new

Nombre

gda_xml_table_new —

Synopsis

Gda_XmlObject * gda_xml_table_new (Gda_XmlDatabase * db, const gchar * name);

Arguments

db

a Gda_XmlDatabase object

name

name of the table to be created

Description

Adds a new table to the given XML database representation. This function just creates the XML node. If you want to set properties for it, you should call the gda_xml_table_set_* functions on the Gda_XmlTable object returned by this function

gda_xml_table_new_from_gda_recordset

Nombre

gda_xml_table_new_from_gda_recordset —

Synopsis

Gda_XmlObject * gda_xml_table_new_from_gda_recordset (Gda_XmlDatabase * db, const gchar * name, Gda_Recordset * recset, gboolean export_data);

Arguments

db

a Gda_XmlDatabase object

name

name of the table to be created

recset

an existing #Gda_Recordset object

export_data

– undescribed –

Description

Adds a new table to the given XML database from an existing #Gda_Recordset object. This function adds a tag for each field in the recordset, and a

gda_xml_table_delete

Nombre

gda_xml_table_delete —

Synopsis

void gda_xml_table_delete (Gda_XmlDatabase * db, Gda_XmlObject * table);

Arguments

db
: a Gda_XmlDatabase object

table
: table to be deleted

gda_xml_table_delete_by_name

Nombre

gda_xml_table_delete_by_name —

Synopsis

```
void gda_xml_table_delete_by_name (Gda_XmlDatabase * db, const gchar * name);
```

Arguments

db
: a Gda_XmlDatabase object

name
: table name

gda_xml_table_get_name

Nombre

gda_xml_table_get_name —

Synopsis

```
const gchar * gda_xml_table_get_name (Gda_XmlObject * table);
```

Arguments

table
: table object

Description

Returns the name of the given table

gda_xml_table_set_name

Nombre

gda_xml_table_set_name —

Synopsis

```
void gda_xml_table_set_name (Gda_XmlObject * table, const gchar * name);
```

Arguments

table
: table object

name
: table name

Description

Sets the name of the given table

Capítulo 6. librería de widget Gda

6.1. Introducción

Esta librería contiene un rico conjunto de widget Gnome que proporcionan un forma fácil de incluir el modelo Gda en sus aplicaciones Gnome.

Todos los widget son widget Gtk, lo que significa que puede seguir un modelo orientado a objetos en su interfaz. Esto le permitirá, por ejemplo, escribir un widget personalizado para su aplicación implementando únicamente la funcionalidad extra. El comportamiento básico esta ya implementado en el widget Gda.

Esto también significa que puede asociar procedimientos a las señales recibidas por los widget, como se hace en los widget Gtk/Gnome. De hecho, Los widget Gda informa de su comportamiento emitiendo señales que puede ser capturadas por su aplicación para obtener información de los eventos señalados.

Por ejemplo, el widget GnomeDbBrowser contiene numerosos widget hijos que a su vez capturan eventos de usuario, que a su vez son capturados por el widget el mismo GnomeDbBriwser. Así, puede, por ejemplo, conocer las interacciones del usuario con el widget para actualizar un barra indicativa de estado, mostrar una caja de dialogo cuando ocurre un error, etc.

Los widget proporcionados por la librería Gda abarcan todo lo que necesita para desarrollar interfaces para sus aplicaciones orientadas a base de datos. Si sus necesidades no estuvieran cubiertas, tiene dos opciones: pedirnos que creemos el widget que necesita, o, implementar su propia subclase del widget Gda mas apropiado. Si prefiere el segundo método, deberá hojear la documentación Gtk, especialmente el capitulo 'Como escribir su propio widget'. [1] Los widget contenidos en la librería son los siguientes:

GnomeDbLogin

GnomeDbLoginDlg

GnomeDbBrowser

GnomeDbGrid

GnomeDbCombo

GnomeDbList

GnomeDbIconList

GnomeDbXmlViewer

GnomeDbReport

GnomeDbError

GnomeDbErrorDlg

6.2. Introducción a los widget Gda

Esta sección contiene una introducción a lo que puede hacerse con los widget suministrados con la librería UI (Interfaz gráfico) Gda. Contiene rápidas explicaciones de la funcionalidad y uso básico de cada widget.

6.2.1. widget GnomeDbLogin

El widget GnomeDbLogin proporciona una forma personalizada de conectar gráficamente su aplicación a un proveedor Gda. Puede añadir a cualquier widget contenedor, así que puede crear, por ejemplo, un formulario y añadir entonces en su interior el widget GnomeDbLogin.

Al iniciarse, lee el fichero de configuración Gda ($HOME/.gnome/gdalib), y proporciona una combo box que contiene todas las fuentes de datos configuradas para ese usuario. Este solo tiene que seleccionar la fuente de datos y teclear su nombre de usuario y su clave. También muestra una lista de las mas recientes fuentes de datos accedidas, de forma que pueda conectar rápidamente con sus bases de datos favoritas.

6.2.2. widget GnomeDbLoginDlg

Este es un widget de conveniencia que simplemente contiene el widget GnomeDbLogin. Es una subclase de GnomeDialog, así que puede esperar que tenga el mismo aspecto que le resto de su aplicación.

Este widget no solo contiene el widget GnomeDbLogin, si no que captura el mismo sus señales, de forma que si lo usa, deberá conectarse a las señales de GnomeDbLoginDlg en lugar de las de GnomeDbLogin.

6.2.3. widget GnomeDbBrowser

Este es un muy completo widget que permite navegar por la toda la base de datos, desde tablas y vistas hasta usuarios y procedimientos almacenados. El caso mas sencillo en el que le interesara usar este widget es la creación de un contenedor y la inclusión del widget GnomeDbBrowser en el. Haciendo simplemente esto brindara un completo navegador de base datos a su aplicación.

6.2.4. widget GnomeDbGrid

Este es el widget mas usado de la librería. Proporciona una forma de inspeccionar gráficamente el contenido de un Gda_Recordset's (conjunto de registros Gda). El widget asume plenamente la responsabilidad de la parte de acceso a la base de datos. Solo debe pedirle que muestre un conjunto de registros valido.

Los datos de cargan de forma asíncrona, de forma que su programa no bloquee el acceso del usuario a otras partes de la aplicación. Una vez que el ultimo registro haya sido recogido, GnomeDbGrid emite la señal "loaded" (cargado).

6.2.5. widget GnomeDbCombo

Este widget es de mucha utilidad. Se usa principalmente en las pantallas de entrada de datos, proporcionando al usuario una forma de seleccionar un valor para un campo dado de una lista de valores.

Funciona prácticamente como el widget GnomeDbGrid, ya que funciona basándose en el Gda_Recorset que se le pasan, pero en este caso, debe especificar que columna del conjunto de registros debe mostrarse.

6.2.6. widget GnomeDbList

El widget GnomeDbList permite mostrar una lista de valores para un campo dado de un conjunto de registros. También le permite decorar la lista con iconos, obteniéndose un widget con una buena presentación para que lo incluya en sus pantallas de acceso a la base de datos.

6.2.7. widget GnomeDbReport

El widget GnomeDbReport proporciona una forma de presentar a los usuarios informes personalizabes basados en datos provenientes de una Gda_Connection.

La estructura de guarda en formato Xml, lo que hace posible compartir datos con otras aplicaciones desconocidas. Este widget únicamente lee el fichero Xml, toma los datos especificados de una base de datos dada y los presenta al usuario basándose en las instrucciones especificadas en la estructura del informe.

6.2.8. widget GnomeDbIconList

El widget GnomeDbIconList funciona como los widget GnomeDbList y GnomeDbCombo. Pero, en este caso, los valores incluidos en un campo dado se representan como iconos. Este le ofrece - al desarrollador - otra forma de presentar datos al usuario de sus aplicaciones.

6.2.9. widget GnomeDbXmlViewer

El widget GnomeDbXmlViewer permite presentar de forma visual al usuario un fichero generado con las funciones Xml de la librería cliente Gda. También le permite modificar el fichero en tiempo de ejecución, mostrando de forma instantánea cualquier cambio realizado en el fichero Xml.

Este widget lo usa, por ejemplo, la herramienta Diseñador en la aplicación de acceso.

6.2.10. widget GnomeDbError

El widget GnomeError proporciona una manera de navegar a través de todos los errores ocurridos en un objeto Gda_Connection dado. Puede usarlo para informar al usuario cuando ocurran errores, o como un widget informativo.

6.2.11. widget GnomeDbErrorDlg

Como el widget GnomeDbLoginDlg, el widget GnomeDbErrorDlg; es simplemente una subclase GnomeDialog de conveniencia que contiene un widget GnomeError.

6.3. Referencia de funciones

Las siguientes secciones describen las funciones disponibles para los usurarios de la librería de widget Gda.

gnome_db_browser_new

Nombre

gnome_db_browser_new —

Synopsis

```
GtkWidget * gnome_db_browser_new (Gda_Connection * cnc);
```

Arguments

cnc
: the connection to be used for browsing

Description

Create a new GnomeDbBrowser widget, binding it to an existing GDA connection if given

Returns

a pointer to the new widget if success, NULL if error

gnome_db_browser_destroy

Nombre

gnome_db_browser_destroy —

Synopsis

```
static void gnome_db_browser_destroy (GnomeDbBrowser * brw);
```

Arguments

brw
: the browser widget to destroy

Description

Destroy the widget pointed to by the brw parameter

gnome_db_browser_get_connection

Nombre

gnome_db_browser_get_connection —

Synopsis

```
Gda_Connection * gnome_db_browser_get_connection (GnomeDbBrowser * brw);
```

Arguments

brw
: the browser widget

Description

Return the Gda_Connection object actually being used by this browser

Returns

a pointer to a Gda_Connection object, or NULL on error

gnome_db_browser_set_connection

Nombre

gnome_db_browser_set_connection —

Synopsis

void gnome_db_browser_set_connection (GnomeDbBrowser * brw, Gda_Connection * cnc);

Arguments

brw

the browser widget

cnc

the Gda_Connection object to be used

Description

Dynamically change the Gda_Connection object being used by the GnomeDbBrowser widget

gnome_db_browser_clear

Nombre

gnome_db_browser_clear —

Synopsis

```
void gnome_db_browser_clear (GnomeDbBrowser * brw);
```

Arguments

brw
: the browser widget

Description

Clear all widgets in the browser

gnome_db_browser_refresh

Nombre

gnome_db_browser_refresh —

Synopsis

```
void gnome_db_browser_refresh (GnomeDbBrowser * brw);
```

Arguments

brw
: the browser widget

Description

Refresh the browser widgets' contents to display possible changes in the underlying recordset

gnome_db_browser_show_procs

Nombre

gnome_db_browser_show_procs —

Synopsis

```
void gnome_db_browser_show_procs (GnomeDbBrowser * brw);
```

Arguments

brw
: the GnomeDbBrowser widget

gnome_db_browser_show_tables

Nombre

gnome_db_browser_show_tables —

Synopsis

void gnome_db_browser_show_tables (GnomeDbBrowser * brw);

Arguments

brw
: the GnomeDbBrowser widget

gnome_db_browser_show_types

Nombre

gnome_db_browser_show_types —

Synopsis

void gnome_db_browser_show_types (GnomeDbBrowser * brw);

Arguments

brw
: the GnomeDbBrowser widget

gnome_db_browser_show_views

Nombre

gnome_db_browser_show_views —

Synopsis

```
void gnome_db_browser_show_views (GnomeDbBrowser * brw);
```

Arguments

brw
: the GnomeDbBrowser widget

gnome_db_combo_new

Nombre

gnome_db_combo_new —

Synopsis

```
GtkWidget * gnome_db_combo_new (Gda_Recordset * recset, gint pos);
```

Arguments

recset
: recordset to be shown

pos
: position of the field being shown

Description

Creates a new GnomeDbCombo widget. You can optionally specify a Gda_Recordset object which will be used by the GnomeDbCombo widget

Returns

a pointer to the new widget, or NULL on error

gnome_db_combo_get_editable

Nombre

gnome_db_combo_get_editable —

Synopsis

```
gboolean gnome_db_combo_get_editable (GnomeDbCombo * dbcombo);
```

Arguments

dbcombo
: the GnomeDbCombo widget

Returns

TRUE if the combo box is editable , or FALSE if not

gnome_db_errordlg_new

Nombre

gnome_db_errordlg_new —

Synopsis

GtkWidget* gnome_db_errordlg_new (Gda_Connection* cnc, gchar* title);

Arguments

cnc

Gda_Connection object to display errors from

title

– undescribed –

title

title of the dialog box

Create a new GnomeDbErrorDlg, which is a subclass of the GnomeDialog widget, just acting as a container for a GnomeDbError widget

Returns

a pointer to the new widget, or NULL on error

gnome_db_errordlg_show_errors

Nombre

gnome_db_errordlg_show_errors —

Synopsis

void gnome_db_errordlg_show_errors (GnomeDbErrorDlg* dialog);

Arguments

dialog

the GnomeDbErrorDlg widget

Description

Pops up the given GnomeDbErrorDlg widget, displaying any errors that may be reported for the connection being used by this widget

gnome_db_errordlg_set_title

Nombre

gnome_db_errordlg_set_title —

Synopsis

```
void gnome_db_errordlg_set_title (GnomeDbErrorDlg * dialog, gchar * title);
```

Arguments

dialog
: the GnomeDbErrorDlg widget

title
: title to be shown

Description

Change the title for the given GnomeDbErrorDlg widget

gnome_db_error_new

Nombre

gnome_db_error_new —

Synopsis

```
GtkWidget* gnome_db_error_new (Gda_Connection* cnc);
```

Arguments

cnc

Gda_Connection to show error from

Description

Create a new GnomeDbError widget, which is a special widget used to traverse errors occurred for a given connection. It is a simple container widget that you can pack into any widget you want.

Returns

a pointer to the new widget, or NULL on error

gnome_db_error_show

Nombre

gnome_db_error_show —

Synopsis

```
void gnome_db_error_show (GnomeDbError* error_widget);
```

Arguments

error_widget
: the GnomeDbError widget

Description

Make the given GnomeDbError widget visible

gnome_db_error_prev

Nombre

gnome_db_error_prev —

Synopsis

```
void gnome_db_error_prev (GnomeDbError* error_widget);
```

Arguments

error_widget
: the GnomeDbError widget

Description

Display the previous error in the given GnomeDbError widget

gnome_db_error_next

Nombre

gnome_db_error_next —

Synopsis

```
void gnome_db_error_next (GnomeDbError* error_widget);
```

Arguments

error_widget
: the GnomeDbError widget

Description

Display the next error in the given GnomeDbError widget

gnome_db_grid_new

Nombre

gnome_db_grid_new —

Synopsis

```
GtkWidget * gnome_db_grid_new (Gda_Recordset * recset);
```

Arguments

recset
: recordset to be shown

Description

Creates a new GnomeDbGrid widget, which displays the given recordset, if given

Returns

a pointer to the new widget, or NULL on error

gnome_db_grid_destroy

Nombre

gnome_db_grid_destroy —

Synopsis

void gnome_db_grid_destroy (GnomeDbGrid * grid);

Arguments

grid
: the grid to be destroyed

Description

Destroys the given grid, freeing any data associated with it, included the Gda_Recordset object

gnome_db_grid_set_background

Nombre

gnome_db_grid_set_background —

Synopsis

void gnome_db_grid_set_background (GnomeDbGrid * grid, gint row, GdkColor *
color);

Arguments

grid

the grid widget

row

row number

color

color to be used as background

Description

Sets the background color of an entire row. The "color" parameter must already be allocated

gnome_db_grid_set_foreground

Nombre

gnome_db_grid_set_foreground —

Synopsis

void gnome_db_grid_set_foreground (GnomeDbGrid * grid, gint row, GdkColor * color);

Arguments

grid

    the grid widget

row

    row number

color

    color to be used as foreground

Description

Sets the foreground color of an entire row. The "color" parameter must already be allocated

gnome_db_grid_select_all

Nombre

gnome_db_grid_select_all —

Synopsis

```
void gnome_db_grid_select_all (GnomeDbGrid * grid);
```

Arguments

grid
: the grid widget

Description

Selects all cells in the given grid

gnome_db_grid_clear_selection

Nombre

gnome_db_grid_clear_selection —

Synopsis

```
void gnome_db_grid_clear_selection (GnomeDbGrid * grid);
```

Arguments

grid
: the grid widget

Description

Clears the current selection in the given grid

gnome_db_grid_get_selected_rows

Nombre

gnome_db_grid_get_selected_rows —

Synopsis

```
gint gnome_db_grid_get_selected_rows (GnomeDbGrid * grid);
```

Arguments

grid
: the grid widget

Returns

the number of selected rows in the given grid widget

gnome_db_grid_copy_clipboard

Nombre

gnome_db_grid_copy_clipboard —

Synopsis

```
void gnome_db_grid_copy_clipboard (GnomeDbGrid * grid);
```

Arguments

grid
: the grid widget

Description

Copies the current selection to the clipboard

gnome_db_grid_get_cell

Nombre

gnome_db_grid_get_cell —

Synopsis

```
const gchar * gnome_db_grid_get_cell (GnomeDbGrid * grid, gint col, gint row);
```

Arguments

grid
: the grid widget

col
: column number

row
: row number

Description

Returns the text of the given cell

gnome_db_grid_set_column_visibility

Nombre

gnome_db_grid_set_column_visibility —

Synopsis

void gnome_db_grid_set_column_visibility (GnomeDbGrid * grid, gint col, gint vis);

Arguments

grid

   the grid object

col

   column to be hidden/shown

vis

   hide (0) or show (1)

Description

Shows or hides a column in the given grid widget. The default is to show all columns

gnome_db_grid_set_row_visibility

Nombre

gnome_db_grid_set_row_visibility —

Synopsis

```
void gnome_db_grid_set_row_visibility (GnomeDbGrid * grid, gint row, gint vis);
```

Arguments

grid
: the grid object

row
: row to be hidden/shown

vis
: – undescribed –

vis

hide (0) or show (1)

Shows or hides a row in the given grid widget. The default is to show all rows

gnome_db_grid_hide_column_titles

Nombre

gnome_db_grid_hide_column_titles —

Synopsis

```
void gnome_db_grid_hide_column_titles (GnomeDbGrid * grid);
```

Arguments

grid
: the grid widget

Description

Hides columns in the given grid

gnome_db_grid_show_column_titles

Nombre

gnome_db_grid_show_column_titles —

Synopsis

```
void gnome_db_grid_show_column_titles (GnomeDbGrid * grid);
```

Arguments

grid
: the grid object

Description

Shows columns in the given grid

gnome_db_grid_hide_row_numbers

Nombre

gnome_db_grid_hide_row_numbers —

Synopsis

```
void gnome_db_grid_hide_row_numbers (GnomeDbGrid * grid);
```

Arguments

grid
: the grid widget

Description

Hides row numbers in the given grid

gnome_db_grid_show_row_numbers

Nombre

gnome_db_grid_show_row_numbers —

Synopsis

```
void gnome_db_grid_show_row_numbers (GnomeDbGrid * grid);
```

Arguments

grid
: the grid widget

Description

Show row numbers in the given grid

gnome_db_grid_clear

Nombre

gnome_db_grid_clear —

Synopsis

```
void gnome_db_grid_clear (GnomeDbGrid * grid);
```

Arguments

grid
: the grid widget

Description

Removes all rows from the given grid

gnome_db_grid_refresh

Nombre

gnome_db_grid_refresh —

Synopsis

```
void gnome_db_grid_refresh (GnomeDbGrid * grid);
```

Arguments

grid
: the grid widget

Description

Refresh grid's contents

gnome_db_grid_get_recordset

Nombre

gnome_db_grid_get_recordset —

Synopsis

Gda_Recordset * gnome_db_grid_get_recordset (GnomeDbGrid * grid);

Arguments

grid
: the grid widget

Description

Returns the Gda_Recordset object being used by this grid

Returns

a pointer to the Gda_Recordsetobject, or NULL if none is being used or if there is an error

gnome_db_grid_set_recordset

Nombre

gnome_db_grid_set_recordset —

Synopsis

```
void gnome_db_grid_set_recordset (GnomeDbGrid * grid, Gda_Recordset * recset);
```

Arguments

grid
: the grid widget

recset
: recordset to be displayed

Description

Sets the recordset to be displayed on the given grid

gnome_db_icon_list_new

Nombre

gnome_db_icon_list_new —

Synopsis

```
GtkWidget * gnome_db_icon_list_new (Gda_Recordset * recset, gint pos);
```

Arguments

recset
: a Gda_Recordset object

pos
: field position

Description

Creates a new GnomeDbIconList widget and associates the given recordset object to it

gnome_db_icon_list_clear

Nombre

gnome_db_icon_list_clear —

Synopsis

```
void gnome_db_icon_list_clear (GnomeDbIconList * icon_list);
```

Arguments

icon_list
: a GnomeDbIconList widget

Description

Clears the GnomeDbIconList widget

gnome_db_icon_list_get_string

Nombre

gnome_db_icon_list_get_string —

Synopsis

```
const gchar * gnome_db_icon_list_get_string (GnomeDbIconList * icon_list);
```

Arguments

icon_list
: a GnomeDbIconList widget

Description

Returns currently selected string in the icon list widget

gnome_db_icon_list_get_recordset

Nombre

gnome_db_icon_list_get_recordset —

Synopsis

```
Gda_Recordset * gnome_db_icon_list_get_recordset (GnomeDbIconList * icon_list);
```

Arguments

icon_list
: a GnomeDbIconList widget

Description

Returns the associated recordset for the given widget

gnome_db_icon_list_set_recordset

Nombre

gnome_db_icon_list_set_recordset —

Synopsis

void gnome_db_icon_list_set_recordset (GnomeDbIconList * icon_list,
Gda_Recordset * recset, gint pos);

Arguments

icon_list

a GnomeDbIconList widget

recset

recordset to be shown

pos

field position

Description

Associates a Gda_Recordset object with the given object, forcing an immediate loading of the data from the underlying database. The data is loaded asynchronously, so control returns to your program immediately

gnome_db_icon_list_get_pixmap

Nombre

gnome_db_icon_list_get_pixmap —

Synopsis

```
const gchar * gnome_db_icon_list_get_pixmap (GnomeDbIconList * icon_list);
```

Arguments

icon_list
: a GnomeDbIconList widget

Description

Returns the pixmap file being used for the given icon list widget

gnome_db_icon_list_set_pixmap

Nombre

gnome_db_icon_list_set_pixmap —

Synopsis

void gnome_db_icon_list_set_pixmap (GnomeDbIconList * icon_list, gchar * pixmap);

Arguments

icon_list

a GnomeDbIconList widget

pixmap

pixmap file

Description

Sets the pixmap file to be used for the icons in the given GnomeDbIconList widget

gnome_db_list_new

Nombre

gnome_db_list_new —

Synopsis

```
GtkWidget * gnome_db_list_new (Gda_Recordset * recset, gint pos);
```

Arguments

recset
: recordset to be shown

pos
: position of the field being shown

Description

Creates a new GnomeDbList widget

Returns

a pointer to the new widget, or NULL on error

gnome_db_list_clear

Nombre

gnome_db_list_clear —

Synopsis

void gnome_db_list_clear (GnomeDbList * dblist);

Arguments

dblist
: GnomeDbList widget

Description

Clears the DbList widget. It clears all entries in the list and frees the recordset associated

gnome_db_list_get_row_count

Nombre

gnome_db_list_get_row_count —

Synopsis

```
gint gnome_db_list_get_row_count (GnomeDbList * dblist);
```

Arguments

dblist

GnomeDbList widget

Description

Returns number of rows actually loaded in the recordset. As the list widget loads data asynchronously, you may call this function before all rows have been fetched from the database, in which case, it won't reflect the "real" number of rows returned in the recordset

Returns

number of rows in the recordset , or -1 on error

gnome_db_list_get_string

Nombre

gnome_db_list_get_string —

Synopsis

```
gchar * gnome_db_list_get_string (GnomeDbList * dblist);
```

Arguments

dblist
: GnomeDbList widget

Description

Returns the currently selected item's text. That is, the literal of the first selected item

Returns

a string containing the literal selected, or NULL if none is selected or if there is an error

gnome_db_list_get_recordset

Nombre

gnome_db_list_get_recordset —

Synopsis

```
Gda_Recordset * gnome_db_list_get_recordset (GnomeDbList * dblist);
```

Arguments

dblist
: GnomeDbList widget

Description

Returns the Gda_Recordset object associated with the given GnomeDbList widget

Returns

a pointer to the Gda_Recordset object being used, or NULL if none or if there is an error

gnome_db_list_set_recordset

Nombre

gnome_db_list_set_recordset —

Synopsis

void gnome_db_list_set_recordset (GnomeDbList * dblist, Gda_Recordset * recset,
gint pos);

Arguments

dblist
: GnomeDbList widget

recset
: recordset to be shown

pos
: position of the field being shown

Description

Associates a Gda_Recordset object with the given GnomeDbList widget and shows its contents

gnome_db_list_get_pixmap

Nombre

gnome_db_list_get_pixmap —

Synopsis

GdkPixmap * gnome_db_list_get_pixmap (GnomeDbList * dblist, gint row);

Arguments

dblist
: GnomeDbList widget

row
: row number

Description

Returns the pixmap being used for the given row

gnome_db_list_set_pixmap

Nombre

gnome_db_list_set_pixmap —

Synopsis

void gnome_db_list_set_pixmap (GnomeDbList * dblist, gint row, GdkPixmap * pixmap, GdkBitmap * mask);

Arguments

dblist

GnomeDbList widget

row

row number

pixmap

GdkPixmap object

mask

– undescribed –

Description

Sets the pixmap to be shown for the given row

gnome_db_list_get_pixmap_mask

Nombre

gnome_db_list_get_pixmap_mask —

Synopsis

```
GdkBitmap * gnome_db_list_get_pixmap_mask (GnomeDbList * dblist, gint row);
```

Arguments

dblist
: GnomeDbList widget

row
: row number

gnome_db_logindlg_new

Nombre

gnome_db_logindlg_new —

Synopsis

```
GtkWidget* gnome_db_logindlg_new (GnomeDbLogin* login, gchar* title);
```

Arguments

login

the GnomeDbLogin widget to be displayed

title

title for the dialog box

Description

Create a new GnomeDbLoginDlg and display on it the given GnomeDbLogin widget. The GnomeDbLoginDlg is a convenience widget created to allow easy creation of a dialog box for connecting to any GDA datasource

Returns

a pointer to the new widget, or NULL on error

gnome_db_logindlg_popup

Nombre

gnome_db_logindlg_popup —

Synopsis

void gnome_db_logindlg_popup (GnomeDbLoginDlg* dialog);

Arguments

dialog

th GnomeDbLoginDlg widget

Description

Popup (open) the given GnomeDbLoginDlg widget, allowing the user to easily selecting the data source to which she wants to connect

gnome_db_login_new

Nombre

gnome_db_login_new —

Synopsis

GtkWidget* gnome_db_login_new (Gda_Connection* cnc, gchar* gda_provider,
gchar* gda_dbname);

Arguments

cnc

the Gda_Connection object to be used to connect

gda_provider

GDA provider

gda_dbname

GDA data source name

Description

Create a new GnomeDbLogin widget. This widget contains all the entry fields required for connecting to a GDA data source, as well as a history list, which allows the user to select from previously successful connections

Returns

a pointer to the new widget, or NULL on error

gnome_db_login_login

Nombre

gnome_db_login_login —

Synopsis

```
void gnome_db_login_login (GnomeDbLogin* login);
```

Arguments

login
: the GnomeDbLogin widget

Description

Attempts to establish a connection to a GDA provider by using the current information entered by the user in the GnomeDbLogin child widgets

gnome_db_report_new

Nombre

gnome_db_report_new —

Synopsis

GtkWidget * gnome_db_report_new (void);

Arguments

void

– undescribed –

Description

Create a new GnomeDbReport widget, which can be used both to dynamically design a report based on data from a GDA provider, and to be run displaying the output to the user

Returns

a pointer to the new widget, or NULL on error

gnome_db_report_new_from_file

Nombre

gnome_db_report_new_from_file —

Synopsis

```
GtkWidget * gnome_db_report_new_from_file (const gchar * file_name);
```

Arguments

file_name
: file to load the report from

Description

Create a new GnomeDbReport widget from an existing report definition
contained in the file_name file

Returns

a pointer to the new widget, or NULL on error

gnome_db_report_clear

Nombre

gnome_db_report_clear —

Synopsis

void gnome_db_report_clear (GnomeDbReport * rep);

Arguments

rep
: the GnomeDbReport widget

Description

Clears all data associated to the given GnomeDbReport widget

gnome_db_report_load

Nombre

gnome_db_report_load —

Synopsis

```
gboolean gnome_db_report_load (GnomeDbReport * rep, const gchar * file_name);
```

Arguments

rep
: the GnomeDbReport widget

file_name
: file to load the report from

Description

Load an existing report definition into the given GnomeDbReport widget

Returns

TRUE on success, FALSE on error

gnome_db_report_save

Nombre

gnome_db_report_save —

Synopsis

```
gboolean gnome_db_report_save (GnomeDbReport * rep);
```

Arguments

rep
: the GnomeDbReport widget

Description

Save the given report to a disk file

Returns

TRUE on success, FALSE on error

gnome_db_report_set_file_name

Nombre

gnome_db_report_set_file_name —

Synopsis

```
void gnome_db_report_set_file_name (GnomeDbReport * rep, const gchar *
file_name);
```

Arguments

rep

  the GnomeDbReport widget

file_name

  file name

Description

Sets the file name to be used when calling gnome_db_report_save

gnome_db_xml_viewer_new

Nombre

gnome_db_xml_viewer_new —

Synopsis

```
GtkWidget * gnome_db_xml_viewer_new (Gda_XmlDatabase * db);
```

Arguments

db
: an existing Gda_XmlDatabase object

Description

Creates a new GnomeDbXmlViewer widget, which allows you to present visually a file generated by the XML interface functions in the GDA client library. It also lets you modify the file at run-time, displaying all the changes you made

gnome_db_xml_viewer_new_from_file

Nombre

gnome_db_xml_viewer_new_from_file —

Synopsis

```
GtkWidget * gnome_db_xml_viewer_new_from_file (const gchar * filename);
```

Arguments

filename
: XML file name

Description

Creates a new GnomeDbXmlViewer from an existing XML file

gnome_db_xml_viewer_clear

Nombre

gnome_db_xml_viewer_clear —

Synopsis

```
void gnome_db_xml_viewer_clear (GnomeDbXmlViewer * viewer);
```

Arguments

viewer

– undescribed –

Description

Clears the given widget. That is, all the child widgets are cleared, and all the associated memory is freed.

gnome_db_xml_viewer_load_file

Nombre

gnome_db_xml_viewer_load_file —

Synopsis

void gnome_db_xml_viewer_load_file (GnomeDbXmlViewer * viewer, const gchar * filename);

Arguments

viewer

– undescribed –

filename

XML file name

Description

Loads an existing XML file and displays its structure in the given GnomeDbXmlViewer widget

gnome_db_xml_viewer_get_database

Nombre

gnome_db_xml_viewer_get_database —

Synopsis

```
Gda_XmlDatabase * gnome_db_xml_viewer_get_database (GnomeDbXmlViewer *
viewer);
```

Arguments

viewer

    a GnomeDbXmlViewer widget

Description

Returns the Gda_XmlDatabase object being used for the given widget

gnome_db_xml_viewer_set_database

Nombre

gnome_db_xml_viewer_set_database —

Synopsis

```
void gnome_db_xml_viewer_set_database (GnomeDbXmlViewer * viewer,
Gda_XmlDatabase * db);
```

Arguments

viewer

a GnomeDbXmlViewer widget

db

a Gda_XmlDatabase object

gnome_db_xml_viewer_get_selected

Nombre

gnome_db_xml_viewer_get_selected —

Synopsis

```
Gda_XmlObject * gnome_db_xml_viewer_get_selected (GnomeDbXmlViewer *
viewer);
```

Arguments

viewer

a GnomeDbXmlViewer widget

Description

Returns the currently selected object in the XML tree as a #Gda_XmlObjet structure. This allows you to directly call the GDA XML interface functions on the returned object

Notas

1. Este manual también esta disponible en Castellano (http://www.croftj.net/~barreiro//spanish/gtk/gtk_tut_12/gtk_tut_12_es.html) , gracias entre otros a Joaquín Cuenca. !visite nuestra web! (mensaje subliminal :-))

Capítulo 7. Proveedores GDA

7.1. Introducción

Los proveedores son ejecutables o librerías compartidas que realizan el trabajo de conversión. Asocian las llamadas Corba con las llamadas especificas de la fuente de datos. Pueden usar ODBC, una librería nativa de la base de datos, analizadores y generadores XML, LDAP, POP3, cualquier cosa que pueda imaginar.

Hay mucho trabajo involucrado alrededor de los proveedores GDA, especialmente si el servidor que va a usarse no es un base de datos. En este caso, además de asociar la capa de la API al modelo GDA, deberá prestar especial atención a la asociación de la estructura de tipo base de datos a una estructura de otro tipo. En ambos casos, esta es la parte mas compleja de la arquitectura GDA.

Para facilitar todo este proceso de añadir un nuevo servidor, se suministra un entorno de trabajo para servidores con GDA. Esto se plasma con la entrega de unos ficheros patrón y un guión llamado gda-buildserver, que puede usarse para crear la estructura básica de un nuevo servidor, y para generar el servidor mismo.

Otra forma de añadir un proveedor GDA, seria el uso de ficheros IDL para generar la implementación del servidor. De esta manera, podría codificar el servidor en su lenguaje y Orb Corba preferidos. Si quiere hacer algo parecido, por favor contacte con los desarrolladores Gnome-Db.

7.2. Pasos para generar un proveedor

Los pasos a seguir para generar un proveedor GDA son sencillos:

generar los esqueletos del servidor: esto se hace invocando el guión gda-buildserver con la opción –skels. Esto generara un conjunto de ficheros de coodigo fuente C:

- gda-srv-connection.[ch]: implementación del objeto conexión GDA
- gda-srv-command.[ch]: implementación del objeto comandos GDA
- gda-srv-recordset.[ch]: implementación del objeto manejo de registros GDA
- gda-srv-error.[ch]: implementación del objeto errores GDA
- gda-srv-field.[ch]: implementación del objeto campos GDA
- gda-srv.[ch]: implementación del servidor Orbit/Corba. ¡No edite estos ficheros a no ser que sepa lo que esta haciendo!
- main-srv.c: función main() del servidor

Implemente todos los objetos GDA, usando las llamadas especificas de su fuente de datos. Preste atención a lo que hace y donde lo hace. Lea en la siguiente sección una explicación concisa de cada una de loas cosas que debe implementar.

Creando el ejecutable: esto se hace de nuevo con el guión gda-buildserver, al que se le pasa una lista de los módulos objeto que conforman su proveedor (principalmente los ficheros gda-srv-* así como cualquier otro fichero fuente que pueda haber añadido para cualquier propósito), y, de forma opcional, un conjunto de opciones. El guión se ocupara de regenerar y compilar todos los ficheros relacionados con Corba (los fichero gda-srv.[ch]), y de llamar a su compilador para crear la librería compartida con la implementación del proveedor y un pequeño programa manejador que actúa como interfaz de la la librería compartida.

Añada un registro al fichero gnome_db.gnorba para indicar la situación del nuevo servidor, y cree una nueva fuente de datos para que las aplicaciones cliente sean capaces de conectar con el recién creado servidor.

7.3. Implementación de proveedores GDA

La asociación entre los objetos Corba y los distintos objetos que debe implementar se realiza a través de estructuras C (todos con el prefijo Gda_SRV_ en su nombre). Estas estructuras contienen numerosos miembros necesarios para distintos propósitos, así que, como se dijo para los ficheros de código fuente generados, puede añadir cualquier miembro necesario a estas estructuras pero respetando el orden dado y sin eliminar ninguno de los campos predefinidos. (Todas estas asociaciones se realizan en los ficheros generados gda-srv.[ch]).

La cosa es tan simple como esto: las funciones que implemente se llamaran con algunos parámetros (objetos Gda_SRV_* y algunas otras cosas). Realiza cualquier operación necesaria y rellena los campos de las estructuras siguiendo ciertas directivas. Entonces, la implementación real Corba convierte estos datos a cualquier formato necesario y los envía de vuelta al cliente Corba (en este caso, probablemente un aplicación que usa la librería cliente GDA).

Capítulo 8. Clientes GDA

8.1. Introducción

Las aplicaciones cliente GDA pueden ser de dos tipos:

- Aplicaciones que usan las librerías cliente GDA. Estas son las mas comunes, siendo su uso sencillo, ya que ocultan toda la complejidad de Corba con una API completa y fácil de usar.
- Clientes Corba. Puede desarrollar una aplicación cliente GDA en otro lenguaje o en otra plataforma donde se disponga de una implementación del ORB de Corba diferente. En este caso, deberá generar los adaptadores ("stubs") del cliente Corba usando el compilador IDL con los ficheros IDL que se entregan con Gnome-Db. Si hace algo similar, par favor contacte con los desarrolladores de Gnome-Db para que sepan como funciona en esas circunstancias.

8.2. Desarrollando clientes Gda

Hay un guión llamado gda-buildclient que tiene en cuenta todas las librerías necesarias para una aplicación cliente Gda. Eso es todo lo que hace, puede no encontrarlo interesante y puentearlo, pero se incluye por simplicidad.

Una llamada típica a este guión quedaría así:

```
gda-buildclient –with-ui –o cliente
          –f <lista de ficheros .o >
          [–l <lista de librerías adicionales>]
          [–L <lista de directorios de búsqueda de librerías>]
```

8.3. Creación de clientes CORBA GDA

Para mostrarle como hacerlo, usaremos el compilador IDL de Java como ejemplo. Como ya sabrá si ha escrito un cliente Corba, se usa un fichero IDL para definir la interfaz proporcionada por el servidor Corba. Este fichero IDL lo usa después un programa llamado "compilador IDL" para generar los adaptadores para el cliente y los esqueletos para el servidor implementados para la interfaz ORB particular. Este fichero se llama gda.idl y se encuentra en su distribución Gnome-Db.

Este compilador IDL es lo que debe usar para generar, en este caso, los adaptadores del cliente. Así, por ejemplo, usando el compilador IDL de Java, debería ejecutar:

```
java-idl gda.idl
```

Esto debería generar una lista de ficheros .java.

Capítulo 9. Aplicación de acceso

9.1. Introducción

La aplicación de acceso (gdafe) es una bonita aplicación que proporciona acceso a toda la potencia del modelo GDA a través de una interfaz de usuario consistente y agradable. La aplicación de acceso utiliza todas las interfaces proporcionadas por las librerías GDA. Se consigue de esta forma una presentación consistente con el resto de las aplicaciones incluidas en el paquete.

Puede utilizarla para interrogar a la base de datos, exportar e importar datos de diferentes formatos, etc. En en futuro soportara formularios e informes definidos por el usuario, que permitirán al usuario definir pantallas e informes personalizadas para ejecutarlos en cualquier momento.