Manual de Programación

Walter Pirri

Manual de Programación
por Walter Pirri

Tabla de contenidos

Lista de tablas

Lista de figuras

Lista de ejemplos

Acerca de este manual

Este manual está destinado a orientar al programador que desba utilizar el monitor transaccional Gnu-Monitor como base para el desarrollo de aplicaciones dentro de una arquitectura cliente/servidor de tres capas.

Capítulo 1. Introducción

En este capítulo se intentará dar una introducción a los conceptos básicos que les permitirán entender el funcionamiento de un sistema transaccional cliente servidor de tres capas.

1.1. Sistema cliente servidor

Es un sistema donde se separan las funcionalidades de una aplicación cliente (la que interactúa con el medio ya sea persona, dispositivo u otro software) y la parte servidora (la que interactúa con los datos o los genera). De esta forma la inteligencia del sistema se divide en dos o mas aplicaciones que pueden estar ejecutandose en diferentes computadores o no. Con esto se logra independizar el cliente del servidor permitiendo que un mismo servidor por ejemplo sirva a varios clientes de distinta naturaleza como pueden ser maquinarias de una linea de producción y terminales para la carga de datos en un laboratorio.

1.2. Sistema de tres capas

En un sistema cliente servidor básico de dos capas generalmente los datos forman parte de la capa servidora, separar en la capa servidora las aplicaciones server de los datos permitiría independizar los servers del medio utilizado para almacenar los datos, que podría ser una base de datos externa. Esto da lugar al sistema de tres capas donde una capa pertenece a las aplicaciones del cliente, otra a las aplicaciones servidoras y la tercera al medio de almacenamiento o generación de datos.

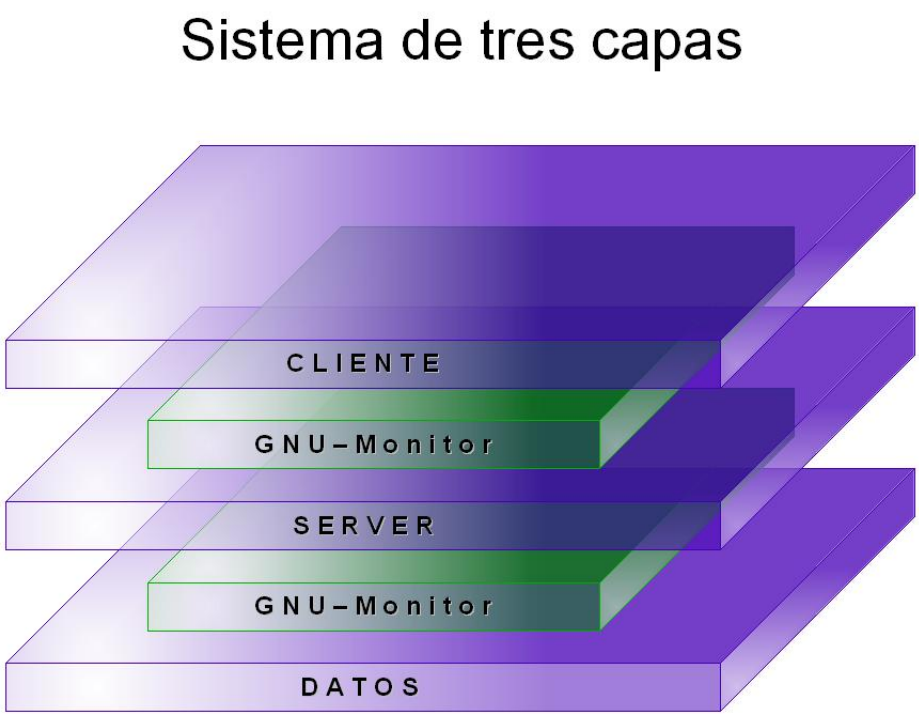
1.3. Sistema transaccional

Un sistema transaccional nos asegura la consistencia de los datos obtenidos y/o guardados al brindarnos las herramientas necesarias para el control del inicio y fin de una operación o transacción de forma de poder tomar las medidas necesarias en caso de fallo en alguno de los procesos intermedios por terminación anormal. Esto combinado con una base de datos relacional asegura una total integridad y consistencia de los datos.

1.4. Gnu-Monitor

GNU-Monitor es un monitor transaccional desarrollado en C/C++ que nos permite construir y mantener de forma muy sencilla aplicaciones de tipo cliente/servidor de tres capas a travez de una interface orientada a objetos y disponible bajo las licencias GNU GPL y LGPL.

Figura 1-1. Sistema cliente servidor de tres capas con Gnu-Monitor



Capítulo 2. Arquitectura del sistema

El sistema está compuesto principalmente por dos grupos de aplicaciones uno cliente y el otro servidor.

El módulo cliente lo integra una librería (libgmc) que resuelve la API de comunicación con el server. En capítulos siguientes se detallará con más profundidad este módulo.

El módulo servidor está formado por varias aplicaciones (gmd, gmt, gmq) y librerías (libgmq) con funciones específicas.

Para el funcionamiento del servidor se provee una serie de scripts (gmon_config, gmon_status, gmon_start, gmon_stop) que ayudan al mantenimiento, configuración y utilización del sistema.

2.1. Interface cliente

La interface cliente la compone una librería dinámica "libgmc.so" que debe ser linkeada con el programa que va a actuar como cliente, la declaración de la interface se encuentra en el archivo "gmc.h".

Esta librería brinda una API orientada a objetos que resuelve todas la opciones de comunicación con el sistema Gnu-Monitor.

Este cliente permite cuatro modalidades para realizar la comunicación, cada una de ellas se aplica a una necesidad diferente en cuanto a volumen de datos de consulta, respuesta o utilización de la información.

Los distintos modos (consulta/respuesta, mensaje, interactivo y encolado) seran detallados en capítulos siguientes.

2.2. Lístener

Es el encargado de recibir los pedidos de conexión, controlar los permisos según la procedencia del mensaje y enviarlos a la cola del monitor gmt para que este lo resuelva.

Este lístener puede funcionar de forma autónoma escuchando en el puerto TCP 5533 o a través de inetd / xinetd.

2.3. Ruteador de transacciones

Recibe el mensaje, lee el header y realiza los controles correspondientes a su integridad. Luego según el servicio que haya sido solicitado por el cliente determina el servidor de destino y envía el mensaje a la cola correspondiente.

Una vez recibida la respuesta genera el mensaje para el listener y se lo devuelve.

2.4. Servers a través de stdio

Este tipo de servidores permite que cualquier aplicación que admita la entrada de datos por stdin y produzca su salida por stdout o stderr pueda ejecutarse dentro del sistema Gnu-Monitor.

El módulo se encarga de levantar la aplicación y escribirle en su stdin los datos que se hayan pasado como mensaje en la consulta. Al finalizar la aplicación llamada (que también puede ser un script) se enviará como respuesta los datos obtenidos por stdout o stderr según cual haya sido el valor de retorno de esta aplicación.

2.5. Servers autónomos

En general es el tipo de aplicaciones que van a ser utilizadas para incorporar al sistema Gnu-Monitor y está compuesta por dos librerías de las cuales una debe ser linkeada con el programa de acuerdo al tipo de server que se necesite según las características que se dettallan en el manual del programador.

Estas librería (libgmq.so y libgmqw.so) se encargan del alta y mantenimiento de las colas de mensajes necesarias para la comunicación con el ruteador de transacciones.

El programa se deberá codificar utilizando como base el objeto CGMServer o CGMServerWaited y la librería llamara a sus miembros en las distintas instancias de tratamiento de los mensajes recibidos como veremos más adelante.

2.6. Scripts de soporte

Para facilitar la utilización y mantenimiento del sistema Gnu-Monitor se proveen una serie de scripts que realizan las tareas administrativas mas comunes.

2.6.1. gmond

Este script a través de sus parámetros permite subir y bajar el sistema Gnu-Monitor. El mismo se instalará en /etc/init.d y deberá ejecutarse con permiso de root o del usuario que se destine a correr el sistema Gnu-Monitor.

Vale la pena aclarar que el subir y bajar unicamente afecta a los servers asociados al sistema Gnu-Monitor y no al listener salvo que se utilice el propietario.

2.6.2. gmon_start

Permite arrancar un server en particular al pasarle su nombre como parametro o todos los servers si el parametro es all.

El servidor debe estar dado de alta previamente en la tabla de configuración de servidores, esto puede hacerse a través de gmon_config.

2.6.3. gmon_stop

Permite detener un server en particular al pasarle su nombre como parámetro o todos los servers si el parámetro es all.

2.6.4. gmon_status

Permite obtener el estado de ejecución de un server en particular al pasarle su nombre como parámetro o todos los servers si el parámetro es all.

2.6.5. gmon_config

Este es un script que a través de una interface interactiva permite realizar las mismas tareas que los antes mencionados.

A su vez también permite la administración de los servidores facilitando la configuración del sistema en el alta, baja y modificación de las tablas de parametrización.

Capítulo 3. Modos de comunicación

Cada uno de estos modos responde a una necesidad particular y los servicios que se resuelven en los servers se declaran para un modo específico. Esto significa que un mismo servicio no puede responder mensajes de dos modos diferentes, en ese caso serían dos servicios diferentes aunque tengan el mismo nombre y lo resuelva internamente la misma función.

3.1. Consideraciones

En adelante se utilizaran algunos términos para referirse a módulos, aplicaciones, funcionalidades, etc. que detallaremos para evitar confusiones.

3.1.1. modalidad

Para un server indica si levanta standalone o debe levantar una aplicación y comunicarse con esta a través de stdio.

3.1.2. aplicación

Corresponde al path completo del binario asociado al server o que este debe correr.

3.1.3. server

Es la representación del proceso que recibe los mensajes desde el ruteador de transacciones. Está asociado a una cola de mensajes y puede corresponder a una aplicación standalone o una ejecutada a través de su stdio.

Es solamente un nombre y puede no tener ninguna relación con el programa que realmente se ejecuta.

Dicho nombre se encuentra dado de alta en una tabla del sistema donde se indica la aplicación a la que se encuentra asociado y la modalidad en que levanta.

3.1.4. servicio

Es un procedimiento que se asocia a un server para que al ser convocado por el cliente el ruteador de transacciones lo dirija al server correspondiente a través de su cola.

Su nombre puede comenzar con un punto . que se reserva para los servicios internos del sistema Gnu-Monitor y puede contener hasta 32 caracteres en una combinación de letras, números y caracteres de puntuación.

En las funciones internas el punto . forma parte del nombre por lo que solamente podrá tener 31 caracteres más.

3.2. Modo Consulta/Respuesta

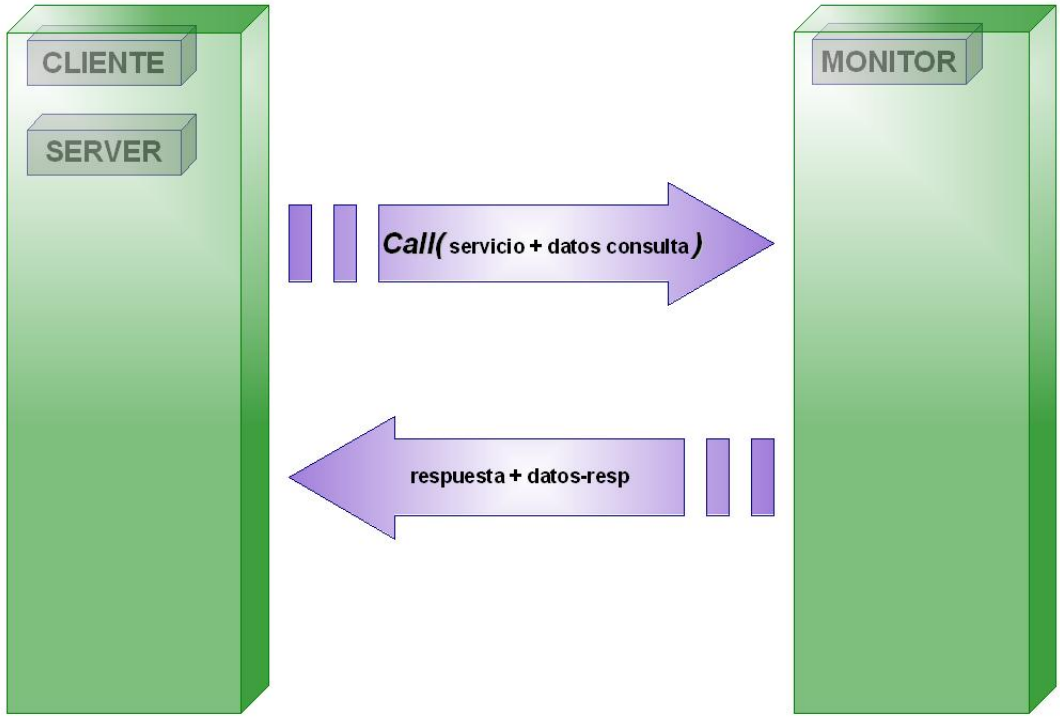
Este tipo de mensaje se utiliza en los casos donde se hace una consulta y se espera la respuesta en un único mensaje. También en los casos en que se envían datos para procesar y se espera el resultado del proceso.

Para procesar el mensaje el ruteador de transacciones arma una lista de todos los servers que responden al servicio solicitado y se la envía al que tenga menos trabajos encolados.

El mensaje devuelto por el server elegido para enviar los datos es devuelto al cliente que solicitó el servicio.

Los mensajes de este tipo se convocan por medio de la API `Call()` del cliente o los servers.

Los servers que atienden los mensajes de este tipo deben suscribirse al servicio con `GM_MSG_TYPE_CR` como tipo de mensaje.



3.3. Modo Evento

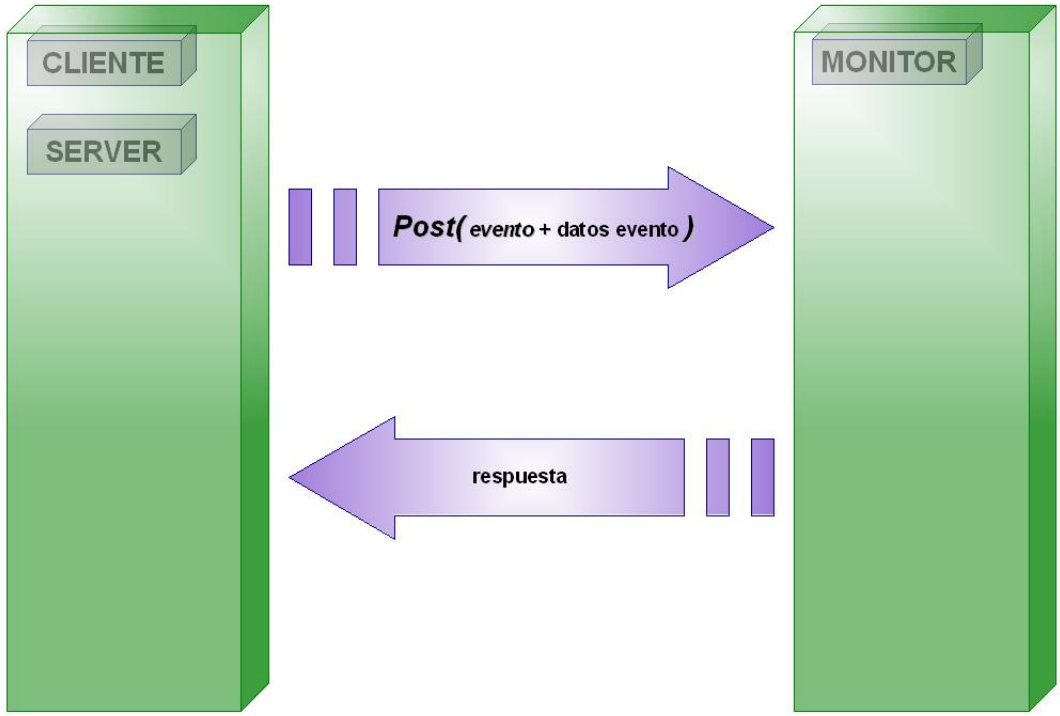
Este tipo de mensaje se utiliza en los casos que se necesita entregar datos al server sin obtener datos de respuesta. Donde la única respuesta es la confirmación de la recepción del mensaje.

Para procesar el mensaje el ruteador de transacciones arma una lista de todos los servers que responden al servicio solicitado y se la envia a todos.

Si al menos un server pudo recibir el mensaje satisfactoriamente se le devuelve una confirmación OK al cliente.

Los mensajes de este tipo se convocan por medio de la API `Post()` del cliente o los servers.

Los servers que atienden los mensajes de este tipo deben suscribirse al servicio con `GM_MSG_TYPE_MSG` como tipo de mensaje.



3.4. Modo Interactivo

Este tipo de mensaje se utiliza cuando la cantidad de datos que el server debe devolver es demasiado grande para utilizar el tipo CONSULTA / RESPUESTA.

Para procesar el mensaje el ruteador de transacciones arma una lista de todos los servers que responden al servicio solicitado y se la envía al que tenga menos trabajos encolados.

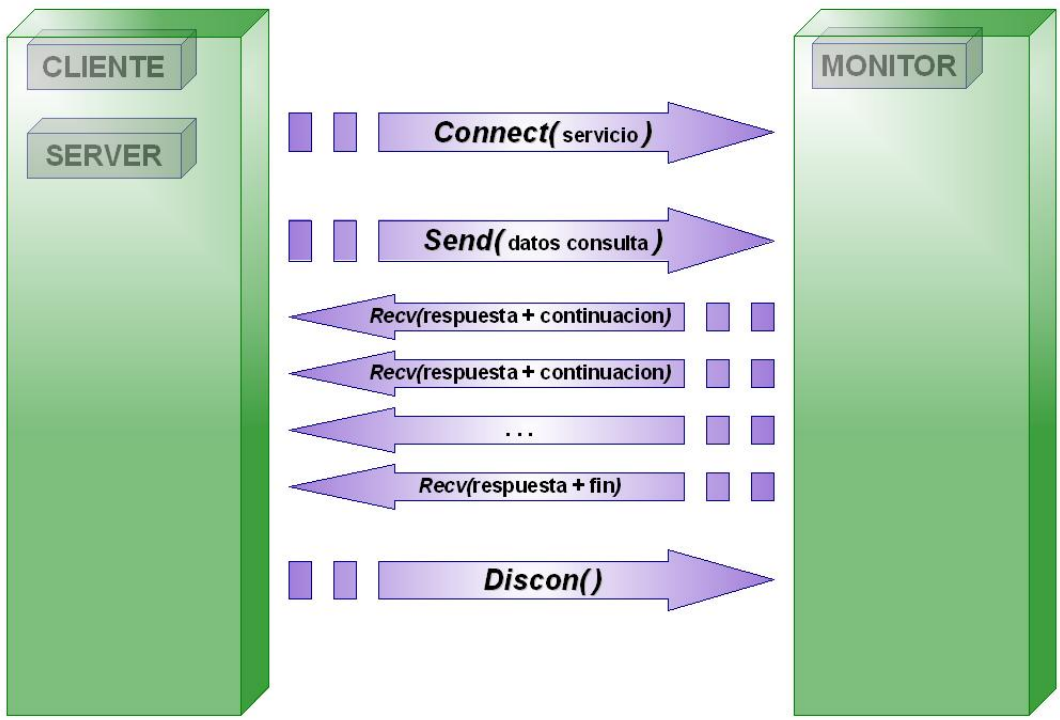
Una vez enviada la consulta el cliente envía tantas peticiones de devolver datos como sean necesarias para traerlos a todos. También se permite cancelar en cualquier momento perdiendo los datos que aún no se hayan transferido.

Los mensajes interactivos que consultan y reciben utilizando las funciones de la API como se detalla a continuación:

```
Connect(...);
Send(...);
```

```
While( Recv(...) >= 0 )
{
 ...
}
Discon();
```

Los servers que atienden los mensajes de este tipo deben suscribirse al servicio con `GM_MSG_TYPE_INT` como tipo de mensaje.

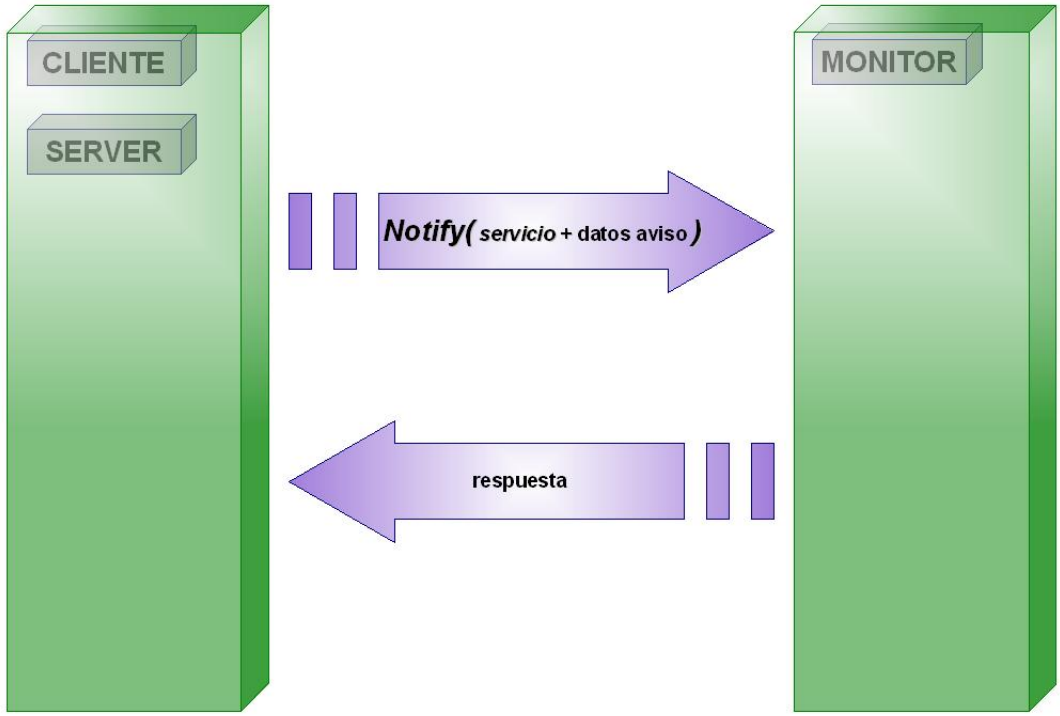


3.5. Modo Aviso

Este tipo de mensaje es muy parecido al `Evento` en el sentido de que no se espera respuesta pero con la diferencia de que el ruteador de transacciones le envía el mensaje a uno solo de los server capaces de procesarlo, el que tenga menos trabajos encolados.

Los mensajes de este tipo se convocan por medio de la API `Notify()` del cliente o los servers.

Los servers que atienden los mensajes de este tipo deben suscribirse al servicio con `GM_MSG_TYPE_NOT` como tipo de mensaje.

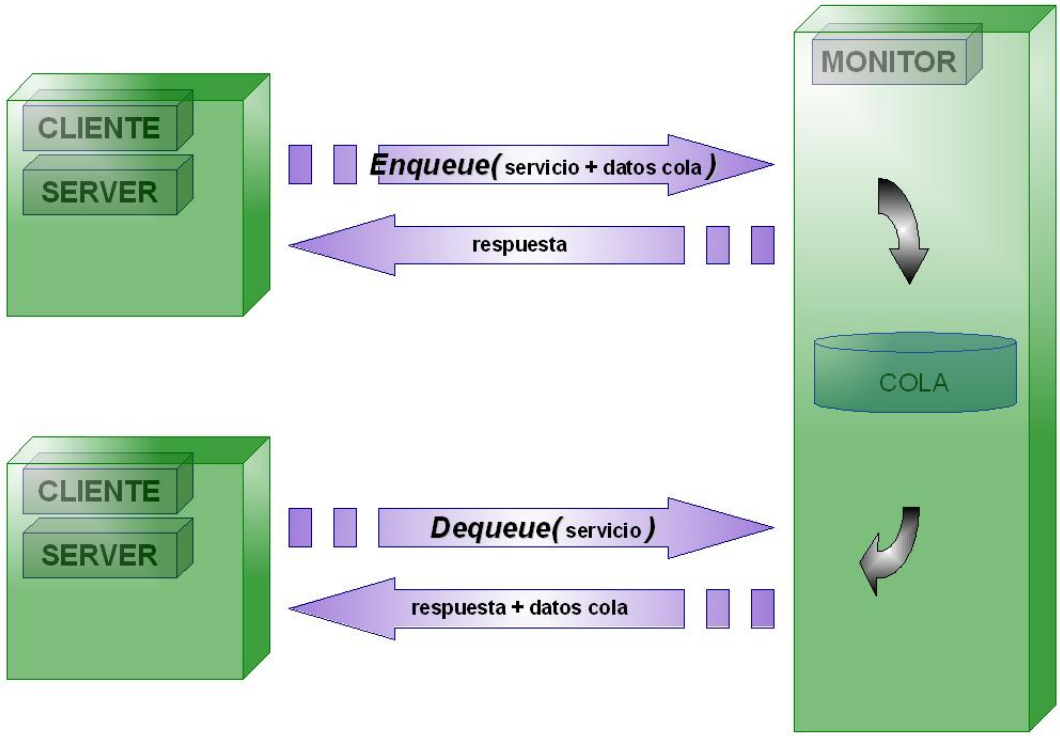


3.6. Modo Encolado

Este tipo de mensaje tiene un parecido al Modo Aviso con la diferencia que los mensajes se encolan en un archivo de Store And Forward para que luego otro cliente o algún server los retiren.

Los mensajes de este tipo se convocan por medio de la API `Enqueue()` del cliente o los servers. Mientras que se utiliza `Dequeue()` para ir retirando mensajes de la cola.

Los servers que atienden los mensajes de este tipo deben suscribirse al servicio con `GM_MSG_TYPE_QUE` como tipo de mensaje.


MONITOR
CLIENTE
SERVER
Enqueue(servicio + datos cola)
respuesta
COLA
CLIENTE
SERVER
Dequeue(servicio)
respuesta + datos cola

Capítulo 4. Protocolo de mensajería

La mensajería entre el cliente y el sistema Gnu-Monitor se realiza a través de un mensaje que contiene un header en texto plano (sin codificar) de 146 bytes.

El header cuenta con suficiente información para determinar el origen del mensaje, varios valores de identificación y autenticación del cliente, tipo de mensaje, servicio solicitado, etc.

A continuación detallaremos el contenido del header y el significado de cada uno de sus componentes.

Tabla 4-1. Componentes del header

Campo	Inicio	Tamaño
TipoMensaje	0	1
VersionHeader	1	3
IdUsuario	4	16
IdCliente	20	16
Key	36	16
IdGrupo	52	16
Funcion	68	32
IdTrans	100	5
IdCola	105	5
IdMoreData	110	5
SecuenciaConsulta	115	5
SecuenciaRespuesta	120	5
OrigenConsulta	125	1
OrigenRespuesta	126	1
IdOrigen	127	5
IdRouter	132	5
IdDestino	137	5
TimeStamp	142	10
CodigoRetorno	152	5
Crc	157	16
TamMensaje	173	10
IndiceMensaje	183	10
TamMaxMensaje	193	10
TamTotMensaje	203	10

4.1. TipoMensaje

Este flag indica el tipo de mensaje al que corresponde entre los comentados anteriormente.

Tipo C: Mensaje de consulta en una comunicación de tipo consulta/respuesta.

Tipo R: Mensaje de respuesta a una consulta de tipo C.

Tipo M: Mensaje tipo evento.

Tipo O: Mensaje de respuesta a una consulta tipo M.

Tipo G: Mensaje de consulta y peticion de mas datos en una comunicación de tipo interactiva.

Tipo L: Mensaje de respuesta a una consulta de tipo G.

Tipo Q: Mensaje de tipo encolado.

Tipo S: Mensaje de respuesta a una consulta de tipo Q.

Tipo N: Mensaje de tipo notificación.

Tipo P: Mensaje de respuesta a uno de tipo N.

Este campo lo llenan tanto el cliente como el servidor y sirve para determinar el procesamiento que se le va a dar a un mensaje del lado del servidor asi como para determinar si una respuesta corresponde con la consulta hecha del lado del cliente.

4.2. VersionHeader

Contiene el número de versión del protocolo de mensajería, se utilizará para lograr compatibilidad hacia atrás en las versiones estables.

Este valor se mantendrá en cero (000) hasta el primer release estable donde pasará a tomar el valor 1 (001).

Cada vez que haya que cambiar el formato del header se completará en este campo el número del release en el que se implementó el cambio en el protocolo.

Los parsers guardarán compatibilidad hacia atrás con respecto al cliente, esto significa que si un cliente intenta conectarse con una versión del protocolo inferior a la del servidor éste le contestará en esa misma versión.

Por el contrario si un cliente intenta conectarse con una versión del protocolo superior a la del server la comunicación será rechazada.

Este campo se llena tanto del lado cliente como del servidor.

4.3. IdUsuario

Corresponde al la identificacion del proceso que utiliza la API del cliente para lograr comunicación con el sistema Gnu-Monitor.

Es útil en el caso de que varias aplicaciones utilicen los servicios desde el mismo cliente y séa necesario individualizarlas a los efectos de enviarles mensajes no solicitados o generar filtros de mensajes especificos, grupos de distribución, etc.

El valor de este campo se inicializa en el momento de iniciar la comunicación y se mantiene constante durante toda la transacción, en caso de variar por algún error del driver no producirá ningún efecto ya que unicamente se registra su valor en el momento de iniciar la comunicación.

4.4. IdCliente

Generalmente el cliente deberá completar este campo con el nombre de host de la terminal donde se encuentra corriendo el proceso, mientras el server lo completará con su propio nombre de host.

Tambien este campo se puede utilizar para definir grupos de distribución de mensajes.

El valor de este campo se inicializa en el momento de iniciar la comunicación y se mantiene constante durante toda la transacción, en caso de variar por algún error del driver no producirá ningún efecto ya que unicamente se registra su valor en el momento de iniciar la comunicación.

4.5. Key

Este campo es para los servicios que requieran una autenticación, en esos casos la clave o el mensaje completo deberán cifrarse

4.6. IdGrupo

Este campo se utiliza para identificar grupo de aplicaciones y es útil si se tiene la necesidad de enviar algún mensaje a todos los clientes de determinado grupo.

El valor de este campo se inicializa en el momento de iniciar la comunicación y se mantiene constante durante toda la transacción, en caso de variar por algún error del driver no producirá ningún efecto ya que unicamente se registra su valor en el momento de iniciar la comunicación.

4.7. Funcion

Este campo lo llena el cliente con el nombre del servicio que deberá ser llamado en el server para procesar los datos del mensaje.

Debe corresponder a un nombre de servicio registrado de lo contrario se devolverá un error.

Consta de hasta 32 caracteres alfanuméricos y puede comenzar con un punto . pero este caracter se reserva para los comando internos del sistema Gnu-Monitor para que no sean confundidos con otros comando de usuario que puedan contener nombres similares. Por este motivo los comando internos están limitados a 31 caracteres (32 con el punto).

4.8. IdTrans

En un mensaje transaccional luego del inicio de la transacción, viaja en este campo el indice de la transacción para que todo los servicios puedan identificar a que transacción deben asignar el mensaje.

4.9. IdCola

.

4.10. IdMoreData

Se utiliza en mensajes interactivos para e indica el indice a la sesión interactiva que permite hacer referencia a los datos guardados cuando se pide continuación.

4.11. SecuenciaConsulta

Este valor es completado por el cliente que lo utiliza como un contador incremental para validar los mensajes de respuesta. A la vuelta del mensaje se controla que la secuencia coincida.

4.12. SecuenciaRespuesta

Este valor lo completa el servicio que responde a un mensaje del tipo interactivo, es para controlar la secuencia en la respuestas múltiples.

4.13. OrigenConsulta

Este campo indica si la consulta es iniciada en un Cliente, un Server o el Router.

4.14. OrigenRespuesta

Este campo indica si la respuesta es resuelta desde un Cliente, un Server o el Router.

4.15. IdOrigen

Este campo es completado por el originante del mensaje utilizando el valor de su propio PID.

4.16. IdRouter

Este campo es completado por el ruteador del mensaje utilizando el valor de su propio PID.

4.17. IdDestino

Este campo es completado por el interprete del mensaje utilizando el valor de su propio PID.

4.18. TimeStamp

Este dato lo completan tanto el cliente como el servidor y es útil al momento de generar logs de mensajes que deban guardar cierta cronología tanto del lado cliente como servidor.

Escaso que se requiera sincronizar ambas partes o se implemente algún mensaje de sincrinización este campo conteiene los datos necesarios.

4.19. CodigoRetorno

Este campo se completa con 0 en las consultas y en las respuestas si contiene un valor distinto a 0 significa que hubo algún tipo de error, ya sea dentro del sistema monitor como destro del server programado por el usuario. Para esto hay una serie de valores definidos.

4.20. Crc

Tanto el cliente como el servidor completan en este campo un codigo de redundancia ciclica del contenido de datos del mensaje.

Ya que su calculo incluye tambien al header, previamente se llena este campo con espacios.

Su propósito es poder verificar la integridad del mensaje.

4.21. TamMensaje

Como su nombre lo indica este campo contiene el valor numerico de la porción de datos del mensaje, sin el header.

4.22. IndiceMensaje

Se utiliza en mensajes del tipo interactivo para indicar en una petición de mas datos el offset a partir del cual se deberá hacer la devolución de la siguiente porción de datos.

En el primer mensaje correspondiente a esta modalidad (interactiva) este dato debe vale cero.

4.23. TamMaxMensaje

Se utiliza en mensajes del tipo interactivo para indicar el tamaño máximo que se devolverá en cada respuesta.

4.24. TamTotMensaje

En mensaje del tipo interactivo representa el tamaño total del bloque de datos que se entregará con los sucesivos pedidos de MAS DATOS.

Capítulo 5. Cliente

Para facilitar la programación todo lo necesario para comunicarse con el sistema GNU-Monitor se encuentra en la librería libgmc.so y el objeto declarado en gmc.h.

5.1. Referencia rápida

A continuación se observan lo miembros de la clase `CGMClient` que implementa la interface del cliente.

Tabla 5-1. Referencia rápida

Miembro	Descripción
`CGMClient`	Constructor de la clase, recibe como parámetro un objeto con los datos del cliente y el host donde se encuentra el sistema GNU-Monitor.
`~CGMClient`	Destructor de la clase, libera toda la memoria alocada por la interface del cliente.
`Free`	Se utiliza para liberar la memoria alocada para el buffer al recibir un mensaje.
`Init`	Asigna los datos del cliente y el host donde se encuentra el sistema Gnu-Monitor, se utiliza cuando se quieren cargar estos datos en algún istante luego de la construcción del objeto.
`Begin`	Marca el inicio de una transacción dentro de una comunicación.
`Commit`	Marca la aceptación de una transacción.
`Abort`	Marca la cancelación de una transacción.
`Notify`	Envía un mensaje de tipo aviso a un servicio, los eventos no contienen datos en su respuesta y se entrega al servidor que resuelva el servicio ue contenga menos mensajes encolados.
`Broadcast`	Envía un mensaje de tipo aviso a múltiples destinatarios identificados por IdUsuario, IdCliente e IdGrupo.
`SetUnsol`	Especifica una función para resolver los mensajes no solicitados en modo sincrónico.
`CheckUnsol`	Realiza una consulta de mensajes no solicitados en modo asincrónico.
`Post`	Envía un mensaje de tipo evento a todos los servers que resuelvan el servicio solicitado.
`Suscribe`	Envía un pedido de suscripcion para mensajes no solicitados.

Miembro	Descripción
`UnSuscribe`	Envía un pedido de baja de suscripcion para mensajes no solicitados.
`Call`	Envía un mensaje de tipo consulta/respuesta al servidor que responda al servicio y tenga menos mensajes encolados.
`ACall`	Envía un mensaje de tipo consulta/respuesta asincrónico al servidor que responda al servicio y tenga menos mensajes encolados.
`GetReply`	Toma la respuesta asincrónica de un mensaje de tipo consulta/respuesta.
`Cancel`	Finaliza de forma anticipada la recepcion del mensaje asincrónico de tipo consulta/respuesta.
`Connect`	Inicia una conversación interactiva.
`Send`	Envía un mensaje de tipo interactivo.
`Recv`	Recibe una respuesta de un mensaje de tipo interactivo un mensaje de tipo interactivo.
`Discon`	Termina una conversación interactiva
`Enqueue`	Envía un mensaje de tipo encolado.
`Dequeue`	Recibe un mensaje de tipo encolado. Con esta funcion se puede implementar en un cliente el tratamiento de mensajes encolados en lugar de hacerlo desde un server.

5.2. Constructor de la clase

5.2.1. CGMClient

Al contruir el objeto de la interface cliente para el sistema Gnu-Monitor uede pasarse como parámetro un puntero a un objeto de la clase `CGMInitData` con los datos necesarios para establecer la comunicación con el sistema.

```
CGMClient(CGMInitData *init_data = NULL);
```

La clase `CGMInitData` está definida de la siguiente forma.

Tabla 5-2. CGMInitData

Miembro	Tipo	Descripción

Miembro	Tipo	Descripción
`m_host`	string	Nombre o dirección del host donde se encuentra corriendo el sistema Gnu-Monitor.
`m_port`	long	Puerto donde escucha el sistema Gnu-Monitor, por defecto este valor es 5533 y no necesita ser asignado salvo que el monitor se configure para otro puerto.
`m_user`	string	Corresponde al nombre de usuario o proceso que se declarará al sistema Gnu-Monitor, se utiliza para diferenciar si es necesario varias instancias del cliente corriendo en el mismo equipo asi como para la utilización de destinatarios y filtros por destinatario en los mensajes no solicitados. Esta información es accesible del lado del monitor por los servers del tipo standalone.
`m_client`	string	Al igual que el anterior es otra forma de identificar al cliente, este dato debería estar relacionado con el equipo donde corre la apliación cliente.
`m_group`	string	Al igual que los anteriores este datos se utiliza para identificar al cliente dentro de un grupo de aplicaciones.
`m_key`	string	Este dato puede ser utilizado para encriptación, eviando a travez de él la clave pública del cliente o utlizarse conpropósitos de identificación enviando una contraseña encriptada.
`m_flags`	long	Aún no se utiliza.

5.2.2. Init

Si no se pueden pasar los datos necesarios en la creación del objeto se puede utilizar este miembro de la clase para inicializar estos valores.

```
voidInit(CGMInitData *init_data = NULL);
```

5.2.3. Free

Se utliliza para liberar el buffer donde se almacenó el mensaje recibido, este es el buffer devuelto por los miembros Call, GetReply y Recv

```
int Free(char *buffer);
```

5.3. Manejo de transacciones

5.3.1. Begin

Inicia un bloque de operaciones comprendida dentro de una misma transacción.

La transacción se termina de forma normal llamando a los miembros `Commit` o `Abort`, también se temina con un `Abort` implícito si finaliza la comunicación con el cliente sin haber convocado a alguno de los miembros de finalización de transacción.

```
int Begin(int to);
```

El parámetro *to* es el tiempo máximo en centésimas de segundo que puede durar la transacción antes que se aborte en forma automática.

5.3.2. Commit

Se utiliza para finalizar una transacción de forma satisfactoria. Provoca que se realice un commit a la base de datos.

La tranzacción debe comenzarse con `Begin`.

```
int Commit(void);
```

5.3.3. Abort

Finaliza una transacción de forma que los datos modificados desde el `Begin` no impactan en la base de datos.

```
int Abort(void);
```

5.4. Mensajes tipo evento

5.4.1. Notify

Envía un mensaje de tipo aviso a un servicio, los eventos no contienen datos en su respuesta y se entrega al servidor que resuelva el servicio ue contenga menos mensajes encolados.

```
int Notify(const char *event, const char *data, unsigned int len);
```

```
int Notify(string &event, CGMBuffer &data);
```

```
int Notify(string &event, CGMBuffer *data);
```

El parámetro *event* indica el nombre del evento notificado.

El parámetro *data* contiene el mensaje asociado al evento.

El parámetro *len* contiene el tamaño del mensaje asociado al evento.

5.4.2. Broadcast

Notifica un evento a todos los grupos, clientes y usuarios que cumplan las condiciones de la máscara pasada como parámetro. Para este miembro se pueden utilizar los comodines * y & para formar los nombres de los destinatarios.

```
int Broadcast(const char *user, const char *client, const char *group, const
char *event, const char *data, unsigned int len);
```

El parámetro *user* contiene la máscara para el usuario de destino del mensaje.

El parámetro *client* contiene la máscara para el cliente de destino del mensaje.

El parámetro *group* contiene la máscara para el grupo de destino del mensaje.

El parámetro *event* indica el nombre del evento notificado.

El parámetro *data* es un puntero al mensaje a enviar.

El parámetro *len* es el tamaño del mensaje.

5.4.3. SetUnsol

Especifica la función que debe ser llamada en caso de recibir mensajes no solicitados y aún no se definió su estructura.

5.4.4. CheckUnsol

Lo utiliza el cliente para consultar la existencia de mensajes no solicitados pendientes para el cliente.

El miembro está sobrecargado con tres definiciones diferentes para adaptarse a la necesidad

```
int CheckUnsol(char *ev_name, char *ev_data, unsigned int *ev_datalen, int
to);
```

```
int CheckUnsol(string &ev_name, CGMBuffer &ev_data, int to);
```

```
int CheckUnsol(string &ev_name, CGMBuffer *ev_data, int to);
```

En el parámetro *ev_name* se devuelve el nombre del evento que se ha detectado.

En el parámetro *ev_data* se devuelve el mensaje asociado con el evento.

En el parámetro *ev_datalen* se devuelve el tamaño del mensaje asociado con el evento.

El parámetro *to* es el tiempo máximo expresado en centésimas de segundo que puede demorar la consulta de mensajes no solicitados luego del cual el procedimiento volverá con error de time-out.

5.4.5. Post

Envía un evento al monitor. Junto con el evento también se envía un bloque de datos que será reenviado a los servers y clientes suscritos al evento.

```
int Post(const char *event, const char *data, unsigned int len);
```

```
int Post(string &event, CGMBuffer &data);
```

```
int Post(string &event, CGMBuffer *data);
```

El parámetro *event* indica el nombre del evento posteado.

El parámetro *data* contiene el mensaje asociado al evento.

El parámetro *len* contiene el tamaño del mensaje asociado al evento.

5.4.6. Suscribe

Informa al monitor que el cliente debe ser notificado ante la ocurrencia del evento solicitado.

```
int Suscribe(const char *event);
```

```
int Suscribe(string &event);
```

El parámetro *event* indica el nombre del evento al que el cliente deséa suscribirse.

5.4.7. UnSuscribe

Solicita al monitor la baja de la suscripción al evento.

```
int UnSuscribe(const char *event);


int UnSuscribe(string &event);
```

El parámetro *event* indica el nombre del evento al que el cliente deséa desuscribirse.

5.5. Mensajes tipo consulta/respuesta

5.5.1. Call

Este miembro permite enviar un mensaje a la vez que espera la respuesta, de esta forma se realiza una consulta sincrónica y se cuenta un parámetro para especificar un tiempo límite para recibir la respuesta.

```
int Call(const char *fn, const char *query, unsigned int qlen, const char
**response, unsigned int *rlen, int to);


int Call(string *fn, CGMBuffer &query, CGMBuffer &response, int to);


int Call(string *fn, CGMBuffer *query, CGMBuffer *response, int to);
```

El parámetro *fn* contiene el nombre del servicio que se convoca.

El parámetro *query* contiene el mensaje que se asocia a la consulta al servicio.

El parámetro *qlen* contiene el tamaño del mensaje de consulta.

En el parámetro *response* se devuelve el mensaje de respuesta de ejecutar el servicio.

En el parámetro *rlen* se devuelve el tamaño de la respuesta.

El parámetro *to* indica el tiempo máximo expresado en centésimas de segundo que se esperará por la respuesta luego del cual el llamado terminará con error de time-out.

Para facilitar la implementación de los buffers para la comunicación se provee la clase `CGMBuffer` que es utilizada por el miembro `Call` entre otros.

5.5.2. ACall

Este miembro trabaja junto con `GetReply` para realizar una consulta asincrónica enviando solamente la consulta.

```
int ACall(const char *fn, const char *query, unsigned int qlen);


int ACall(string *fn, CGMBuffer &query);


int ACall(string *fn, CGMBuffer *query);
```

El parámetro *fn* contiene el nombre del servicio que se convoca.

5.5.3. GetReply

Se utiliza para recibir la respuesta a una consulta hecha con `ACall`.

```
int GetReply(const char **response, unsigned int *rlen, int to);


int GetReply(CGMBuffer &response, int to);


int GetReply(CGMBuffer *response, int to);
```

En el parámetro *response* se devuelve el mensaje de respuesta de ejecutar el servicio.

En el parámetro *rlen* se devuelve el tamaño de la respuesta.

El parámetro *to* indica el tiempo máximo expresado en centésimas de segundo que se esperará por la respuesta luego del cual el llamado terminará con error de time-out.

5.5.4. Cancel

Se utiliza para cancelar una consulta hecha con `ACall` cuando ya no se deséa recibir la respuesta.

```
int Cancell(void);
```

5.6. Mensajes tipo interactico

5.6.1. Connect

Comienza una comunicación para un servicio interactivo.

```
int Connect(const char *fn);
```

```
int Connect(string &fn unsigned long transfer_len);
```

El parámetro *fn* contiene el nombre del servicio que se convoca para la conexión interactiva.

El parámetro *transfer_len* indica el tamaño máximo que deberán tener las respuestas.

5.6.2. Send

Envía la consulta de un servicio interactivo. Se debe llamar primero a `Connect`.

```
int Send(const char *query, unsigned int qlen);
```

```
int Send(CGMBuffer &query);
```

```
int Send(CGMBuffer *query);
```

El parámetro *query* contiene el mensaje que se asocia a la consulta al servicio.

El parámetro *qlen* contiene el tamaño del mensaje de consulta.

5.6.3. Recv

Permite recibir los bloques de respuestas de un servicio interactivo.

En caso que seam necesarios mas mensajes para completar el bloque de datos de la respuesta esta fución devolverá un código de retorno de GME_MORE_DATA mintras falten datos por entregar.

```
int Recv(const char **response, unsigned int *rlen, int to);


int Recv(CGMBuffer &response, int to);


int Recv(CGMBuffer *response, int to);
```

En el parámetro *response* se devuelve el mensaje de respuesta de ejecutar el servicio.

En el parámetro *rlen* se devuelve el tamaño de la respuesta.

El parámetro *to* indica el tiempo máximo expresado en centésimas de segundo que se esperará por la respuesta luego del cual el llamado terminará con error de time-out.

5.6.4. Discon

Finaliza la comunicación de un servicio interactivo.

```
int Discon(void);
```

5.7. Mensajes tipo encolado

5.7.1. Enqueue

Envía un mensaje para que sea encolado y enviado cuando el servicio esté disponible.

```
int Enqueue(const char *cn, const char *data, unsigned int datalen);
```

El parámetro *cn* indica el nombre de la cola donde se encolará el mensaje asociado.

El parámetro *data* corresponde al mensaje.

El parámetro *data* corresponde al tamaño del mensaje.

5.7.2. Dequeue

Permite a un cliente recibir los mensajes encolados de los servicios a los que esté suscripto.

```
int Dequeue(const char *cn, const char **data, unsigned int *datalen);
```

En el parámetro *cn* es de entrada salida, si se le asigna un valor se buscarán mensajes unicamente en la cola correspondiente. Si no se le asigna valor se devuelve el nombre de la cola de la cual se extrajo el mensaje.

En el parámetro *data* se devuelve el mensaje y debe ser liberado con `Free`.

En el parámetro *data* se devuelve el tamaño del mensaje.

5.8. Cliente de ejemplo

Para demostrar lo sencillo que es desarrollar un cliente para el sistema Gnu-Monitor se muestra a continución un ejemplo de uno que realiza una consulta de un servicio del tipo CONSULTA / RESPUESTA básico.

Ejemplo 5-1. Cliente de ejemplo

```
#include <gmonitor/gmc.h>
#include <iostream>

int main(int argc, char** argv)
{
  int rc;
  int i;
  CGMInitData gminit;
  CGMClient *pClient;
  CGMError gmerror;
  string servicio;
  CGMBuffer query;
  CGMBuffer response;

  gminit.m_user = "USER";
  gminit.m_client = "CLIENT";
  gminit.m_key = "KEY";
  gminit.m_group = "GROUP";
  servicio = ".eco";
  query = "la verdad de la milanesa";
  for(i = 1; i < argc; i++)
  {
    if( ! strcmp("-h", argv[i]))
    {
      i++;
      gminit.m_host = argv[i];
    }
    else if( ! strcmp("-s", argv[i]))
    {
      i++;
      servicio = argv[i];
      query.Clear();
    }
    else if( ! strcmp("-d", argv[i]))
    {
      i++;
      query = argv[i];
    }
    else
    {
      cerr << "Use: " << argv[0] << " " << "[-h host] [-s servicio] [-d dato]" << endl;
      exit(1);
    }
  }
  pClient = new CGMClient(&gminit);
  rc = pClient->Call(servicio, query, response, 30000);
  if(rc != 0)
  {
    cout << "<!> Error -> " <<  rc << endl;
    cout << "    ERROR: " << gmerror.Message(rc) << endl;
  }
```

```
  else
  {
    cout << "<i> Ok [" << response.Length() << "] " << response.C_Str() << endl;
  }
  delete pClient;
  return 0;
}
```

Capítulo 6. Server standalone

Este tipo de server es el que está completamente integrado al sistema lo que le permite acceder a todas las facilidades que brinda la interface del server con el monitor.

Su característica principal es que permanece inactivo hasta el arrivo de algún mensaje o evento y vuelve a ese estado de inactividad una vez resuelto el servicio correspondiente.

Para poder realizar tareas programadas con este tipo de server es necesario trabajar junto con el servicio de timers del server gm_timer ya que por su arquitectura este tipo de server solamente responde a eventos externos.

Para ello el server debe estar construido basado en un objeto de la clase `CGMServer` y el programa se deberá linkear con la librería `libgmq.so`.

6.1. Referencia rápida

A continuación se observan los miembros de la clase `CGMServerBase` y `CGMServer` que implementa la interface del server.

En la clase `CGMServerBase` encontramos una serie de miembros que pueden utilizarse para intercambiar datos con el sistema Gnu-Monitor y obtener datos del cliente que convocó al servicio en el server.

En la clase `CGMServer` encontramos una serie de miembros que son llamados por el sistema Gnu-Monitor en distintas instancias durante la vida del server y la recepción de mensajes.

6.1.1. Miembros para comunicarse con el sistema Gnu-Monitor

Son los que pueden ser convocados desde el server para enviar mensajes al monitor.

Tabla 6-1. Miembros para comunicarse con el sistema Gnu-Monitor

Miembro	Descripción
`Notify`	Envía un mensaje de tipo evento a un destinatario identificado por su IdUsuario.
`Broadcast`	Envía un mensaje de tipo evento a múltiples destinatarios identificados por IdUsuario, IdCliente e IdGrupo.

Miembro	Descripción
`Post`	Envía un mensaje al sistema Gnu-Monitor.
`Suscribe`	Informa al sistema Gnu-Monitor que el server resuelve el servicio pasado como parámetro.
`UnSuscribe`	Informa al sistema Gnu-Monitor que el server deja de resolver el servicio pasado como parámetro.
`Call`	Envía un mensaje del tipo CONSULTA/RESPUESTA al sistema Gnu-Monitor.
`m_ClientData`	Al recibir una solicitud de servicio por parte de un cliente este miembro que es un objeto de la clase `CGMInitData` contiene los datos de este.
`m_monitor_path`	Contiene el path completo al binario del monitor.
`m_config_path`	Contiene el path donde se encuentran los archivos de configuración del sistema Gnu-Monitor.
`m_server_name`	Contiene el nombre del server que se pasa como parámetro al arrancarlo.

6.1.2. Miembros llamados por el sistema Gnu-Monitor

Son los que el sistema monitor convoca para pasar mensajes o eventos al server y deben ser resueltos por el programador.

Tabla 6-2. Miembros llamados por el sistema Gnu-Monitor

Miembro	Descripción
`CGMServer`	Constructor de la case, se lo llama cuando se crea el objeto al arrancar el server.
`~CGMServer`	Destructor de la clase, se lo llama cuando se destruye el objeto al terminar el server.
`Init`	Este miembro es llamado al inicializar el server.
`Exit`	Este miembro es llamado al terminar el server, antes de convocar al destructor.
`BeginTrans`	Este miembro es llamado cuando el cliente indica que comienza una transacción.
`CommitTrans`	Este miembro es llamado cuando el cliente indica la finalización satisfactoria de una transacción.
`RollbackTrans`	Este miembro el llamado cuando el cliente aborta una transacción, se corta o cancela la comunicación con el cliente o se produce un time-out por inactividad sin haber terminado la transacción en forma satisfactoria por el cliente.

Miembro	Descripción
`PreMain`	Este miembro es llamado al recibir un mensaje y antes de convocar al miembro `Main` para procesarlo.
`PosMain`	Este miembro es llamado luego de haber sido procesado el mensaje de forma satisfactoria por el miembro `Main`.
`Main`	Este miembro es llamado para procesar el mensaje recibido y debe generar el mensaje de respuesta en caso de ser necesario.

6.2. Miembros para comunicarse con el sistema Gnu-Monitor

Son los que pueden ser convocados desde el server para enviar mensajes al monitor de la misma forma que si se tratara de un cliente.

6.2.1. Notify

Envía una notificación de evento a un grupo, cliente o usuario en particular.

```
int Notify(const char *dst, const char *data, unsigned int len);
```

El parámetro *dst* indica el nombre del grupo de destino para el mensaje.

El parámetro *data* es un puntero al mensaje a enviar.

El parámetro *len* es el tamaño del mensaje.

6.2.2. Broadcast

Notifica un evento a todos los grupos, clientes y usuarios que cumplan las condiciones de la máscara pasada como parámetro. Para este miembro se pueden utilizar los comodines * y & para formar los nombres de los destinatarios.

```
int Broadcast(const char *user, const char *client, const char *group, const
char *data, unsigned int len);
```

El parámetro *user* contiene la máscara para el usuario de destino del mensaje.

El parámetro *client* contiene la máscara para el cliente de destino del mensaje.

El parámetro *group* contiene la máscara para el grupo de destino del mensaje.

El parámetro *data* es un puntero al mensaje a enviar.

El parámetro *len* es el tamaño del mensaje.

6.2.3. Post

Envía un evento al monitor. Junto con el evento también se envía un bloque de datos que será reenviado a los servers y clientes suscritos al evento.

```
int Post(const char *event, const char *data, unsigned int len);
```

El parámetro *event* indica el nombre del evento posteado.

El parámetro *data* contiene el mensaje asociado al evento.

El parámetro *len* contiene el tamaño del mensaje asociado al evento.

6.2.4. Suscribe

Informa al monitor que el server atenderá los mensajes enviados para el evento o servicio y tipo de mensaje que se suscribe.

De esta forma la parametrización del ruteo se realiza de forma dinámica al levantarse cada servidor.

Está permitido que mas de un servidor atiendan el mismo servicio. En caso que el servicio utilice mensajes del tipo consulta/respuesta o interactivos el requerimiento se enviará al que tenga menos trabajos encolados y si el servicio utiliza mensajes de tipo eventos se enviará a todos los registrados.

No se permite que un mismo servicio se atienda con dos tipos de mensaje diferente.

```
int Suscribe(const char *event, char tipo_mensaje);
```

El parámetro *event* indica el nombre del servicio al que se suscribe.

El parámetro *tipo_mensaje* indica el tipo de mensaje para el sevicio (C = Consulta/Respuesta, M = Evento, G = Interactivo y Q = encolado).

6.2.5. UnSuscribe

Informa al monitor que el server deja de atender los requerimientos para el servicio indicado.

Deberá ser llamado al bajar el server para que el monitor deje de rutear los mensajes a este servidor.

```
int UnSuscribe(const char *event);
```

El parámetro *event* indica el nombre del servicio al que se desuscribe.

6.2.6. Call

Este miembro permite enviar un mensaje a la vez que espera la respuesta, de esta forma se realiza una consulta sincrónica y se cuenta un parámetro para especificar un tiempo límite para recibir la respuesta.

```
int Call(const char *fn, const char *query, unsigned int qlen, const char
**response, unsigned int *rlen, int to);
```

```
int Call(string *fn, CGMBuffer &query, CGMBuffer &response, int to);
```

```
int Call(string *fn, CGMBuffer *query, CGMBuffer *response, int to);
```

El parámetro *fn* contiene el nombre del servicio que se convoca.

El parámetro *query* contiene el mensaje que se asocia a la consulta al servicio.

El parámetro *qlen* contiene el tamaño del mensaje de consulta.

En el parámetro *response* se devuelve el mensaje de respuesta de ejecutar el servicio.

En el parámetro *rlen* se devuelve el tamaño de la respuesta.

El parámetro *to* indica el tiempo máximo expresado en centésimas de segundo que se esperará por la respuesta luego del cual el llamado terminará con error de time-out.

Para facilitar la implementación de los buffers para la comunicación se provee la clase `CGMBuffer` que es utilizada por el miembro `Call` entre otros.

6.3. Miembros que mantienen información del entorno

Son variables globales a la clase que contienen distinto tipo de información acerca del server y del sistema Gnu-Monitor.

6.3.1. m_ClientData

Contiene información acerca del cliente que convocó el servicio. Estos datos se encuentran disponibles a partir de que el monitor hace el llamado al miembro `PreMain` de la clase `CGMServer` y contiene lo siguiente.

Tabla 6-3. CGMInitData

Miembro	Tipo	Descripción
`m_host`	string	Nombre o dirección del host del cliente.
`m_port`	long	Puerto desde el cual se realizó la comunicación.

Miembro	Tipo	Descripción
`m_user`	string	Corresponde al nombre de usuario o proceso que se declarará al sistema Gnu-Monitor, se utiliza para diferenciar si es necesario varias instancias del cliente corriendo en el mismo equipo asi como para la utilización de destinatarios y filtros por destinatario en los mensajes no solicitados. Esta información es accesible del lado del monitor por los servers del tipo standalone.
`m_client`	string	Al igual que el anterior es otra forma de identificar al cliente, este dato debería estar relacionado con el equipo donde corre la apliación cliente.
`m_group`	string	Al igual que los anteriores este datos se utiliza para identificar al cliente dentro de un grupo de aplicaciones.
`m_key`	string	Este dato puede ser utilizado para encriptación, eviando a través de él la clave pública del cliente o utlizarse conpropósitos de identificación enviando una contraseña encriptada.
`m_flags`	long	Aún no se utiliza.

6.3.2. m_monitor_path

Indica el path completo de donde se encuentran los binarios del sistema Gnu-Monitor.

string `m_monitor_path`

6.3.3. m_config_path

Indica al path completo de donde se encuentran los archivos de configuración del sistema.

string `m_config_path`

6.3.4. m_server_name

Indica el nombre lógico del server pasado a éste en el momento de levantarlo.

string `m_server_name`

6.4. Miembros llamados por el sistema Gnu-Monitor

Son los que el sistema monitor convoca para pasar mensajes o eventos al server y deben ser resueltos por el programador.

6.4.1. CGMServer

El constructor de la clase es llamado cuando se crea el objeto del server.

```
CGMServer(void);
```

6.4.2. ~CGMServer

El destructor de la clase se llama al eliminar el objeto server.

```
~CGMServer(void);
```

6.4.3. Init

Este miembro es llamado luego de la creación del objeto y de haber seteado los valores de las variables `m_monitor_path`, `m_config_path` y `m_server_name`.

```
int Init(void);
```

6.4.4. Exit

Este miembro es llamado justo antes de destruir el objeto del server.

```
int Exit(void);
```

6.4.5. Main

A este miembro se lo convoca al solicotarse un servicio suscripto por el server y se le pasa por parámetro el nombre del servicio y los datos asociados.

El valor de error devuelto por este miembro será el transferido al cliente en caso que este se distinto de cero.

Si el valor devuelto es cero al cliente se le responderá con los datos cargados al buffer de salida.

```
int Main(const char *funcion, void *in, unsigned long inlen, void ** out,
unsigned long *outlen);
```

El parámetro *funcion* es el nombre del servcio convocado para esta sesión.

El parámetro *in* es un puntero a los datos asociados al servicio.

El parámetro *inlen* es el tamaño del bloque de datos apuntado por *in*.

El parámetro *out* es un puntero al puntero donde se alojarán los datos de salida que serán enviados al cliente.

El parámetro *outlen* es el tamaño de los datos de salida apuntados por *out*.

6.4.6. PreMain

Este miembro es convocado antes de la llamada al `Main` y luego de haber seteado los valores de la variable `m_ClientData`.

```
int PreMain(void);
```

6.4.7. PosMain

Este miembro es llamado luego del retorno de la función `Main`, puede utilizarse para eliminar memoria asignada durante el procesamiento.

```
int PosMain(void);
```

6.4.8. BeginTrans

Este miembro es llamado al inicio de un boqe con control transaccional.

```
int BeginTrans(void);
```

6.4.9. CommitTrans

Este miembro es llamado para finalizar satisfactoriamente un bloque con control transaccional comenzado con `BeginTrans`.

```
int CommitTrans(void);
```

6.4.10. RollbackTrans

Este miembro es llamado para cancelar el bloque de control transaccional.

```
int RollbackTrans(void);
```

6.5. Server standalone de ejemplo

Para demostrar lo sencillo que es desarrollar un server para el sistema Gnu-Monitor se muestra a continución un ejemplo de uno que resuelve la consulta al servicio ".eco", el convocado en el "Cliente de

ejemplo", entre otros.

Ejemplo 6-1. Server standalone de ejemplo

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <sys/wait.h>
#include <syslog.h>
#include <fcntl.h>
#include <errno.h>

#include <gmonitor/gmerror.h>
#include <gmonitor/gmconst.h>
#include <gmonitor/gmbuffer.h>
#include <gmonitor/cmsg.h>
#include <gmonitor/gmisc.h>

#include <gmonitor/gms.h>

/*
  La variable 'void* m_gptr' es un puntero para uso generico por el server
  es el unico puntero miembro de la clase que puede ser utilizado libremente
  al realizar el programa server.
  La variable 'CGMInitData m_ClientData' se completará con los valores del
  cliente que solicitó el servicio antes del llamado a la función PreMain()
  y mantendrá este valos para ser utilizado por el server si es necesario
  hasta el final del servicio.
*/

CGMServer::CGMServer() { }

CGMServer::~CGMServer() { }

/* Colocar en esta funcion lo que se necesite correr al levantar el server */
int CGMServer::Init()
{
  int rc;

  syslog(LOG_INFO, "CGMServer::Init()");

  /* Servicio de prueba de enlace */
  if(Suscribe(".eco", 'C') != GME_OK) return -1;
  /* Servicio para loggear mensajes en el log del sistema */
  if(Suscribe(".log", 'M') != GME_OK) return -1;
  /* Servicio de consulta de eventos */
  if(Suscribe(".check_unsol", 'C') != GME_OK) return -1;
  /* Servicios de suscripcion y dessuscripcion a eventos */
  if(Suscribe(".suscribe_cli", 'C') != GME_OK) return -1;
  if(Suscribe(".unsuscribe_cli", 'C') != GME_OK) return -1;
```

```
  return 0;
}

/* Colocar en esta funcion lo que se necesite correr al bajar el serer */
int CGMServer::Exit()
{
  UnSuscribe(".eco");
  UnSuscribe(".log");
  UnSuscribe(".check_unsol");
  UnSuscribe(".suscribe_cli");
  UnSuscribe(".unsuscribe_cli");

  return 0;
}

/* Estas rutinas son llamadas para el manejo de transaccion se debe colocar en ellas el cód
int CGMServer::BeginTrans()
{
  return 0;
}
int CGMServer::CommitTrans()
{
  return 0;
}
int CGMServer::RollbackTrans()
{
  return 0;
}

/* estas rutinas se llaman antes y después de la de procesamiento de mensaje */
int CGMServer::PreMain()
{
  return 0;
}
int CGMServer::PosMain()
{
  return 0;
}

/* Colocar en esta funcion el proceso que intepreta el mensaje recibido */
int CGMServer::Main(const char *funcion, void* in, unsigned long inlen, void** out, unsigne
{
#ifdef DEBUG
  syslog(LOG_INFO, "CGMServer::Main(%s, 0x%08X, %lu)", funcion, in, inlen);
#endif /* DEBUG */
  if(!strcmp(funcion, ".eco"))
  {
    *outlen = inlen;
    *out = (char*)malloc(*outlen);
    memcpy(*out, in, *outlen);
    return GME_OK;
  }
  else if(!strcmp(funcion, ".log"))
```

```
  {
    syslog( LOG_INFO, "[LOG_SERVER][%s][%s][%s][%s] %s",
    m_ClientData.m_host.c_str(),
    m_ClientData.m_user.c_str(),
    m_ClientData.m_client.c_str(),
    m_ClientData.m_group.c_str(),
    (char*)in);
    return GME_OK;
  }
  else if(!strcmp(funcion, ".check_unsol"))
  {
    return GME_OK;
  }
  else if(!strcmp(funcion, ".suscribe_cli"))
  {
    return GME_OK;
  }
  else if(!strcmp(funcion, ".unsuscribe_cli"))
  {
    return GME_OK;
  }
  else
  {
    return GME_SVC_NOTFOUND;
  }
}
```

Capítulo 7. Server waited

Este tipo de server se integra al sistema a través de los miembros de la clase `CGMServerWait` que implementa la comunicación con el sistema Gnu-Monitor.

A diferencia del Server standalone este se implementa como un programa común con su propio main que por medio de los miembros `Wait` y `Resp` puede recibir y contestar mensajes al sistema.

La facilidad de esta clase se obtiene linkeando la aplicación que va a funcionar como server con la librería `libgmsw.so` que publica la clase `CGMServerWait`.

7.1. Referencia rápida

A continuación se observan los miembros de la clase `CGMServerBase` y `CGMServerWait` que implementan la interface del server waited.

En la clase `CGMServerBase` encontramos una serie de miembros que pueden utilizarse para intercambiar datos con el sistema Gnu-Monitor y obtener datos del cliente que convocó al servicio en el server.

En la clase `CGMServerWait` encontramos una serie de miembros que se utilizan para recibir y contestar mensajes al sistema.

Tabla 7-1. Referencia rápida

Miembro	Descripción
`CGMServerWait`	Constructor de la clase.
`~CGMServerWait`	Destructor de la clase.
`Init`	Inicializa los parametros del server y la conexión con el minitor.
`Wait`	Espera por el arribo de un mensaje proveniente del monitor. La espera puede ser nula, una cantidad determinada de centésimas de segundo o infinita.
`Resp`	Responde al monitor el ultimo mensaje recibido. Solo se puede procesar de un mensaje a la vez.
`Notify`	Envía un mensaje de tipo evento a un destinatario identificado por su IdUsuario.
`Broadcast`	Envía un mensaje de tipo evento a múltiples destinatarios identificados por IdUsuario, IdCliente e IdGrupo.
`Post`	Envía un mensaje al sistema Gnu-Monitor.

Miembro	Descripción
`Suscribe`	Informa al sistema Gnu-Monitor que el server resuelve el servicio pasado como parámetro.
`UnSuscribe`	Informa al sistema Gnu-Monitor que el server deja de resolver el servicio pasado como parámetro.
`Call`	Envía un mensaje del tipo CONSULTA/RESPUESTA al sistema Gnu-Monitor.
`m_ClientData`	Al recibir una solicitud de servicio por parte de un cliente este miembro que es un objeto de la clase `CGMInitData` contiene los datos de este.
`m_monitor_path`	Contiene el path completo al binario del monitor.
`m_config_path`	Contiene el path donde se encuentran los archivos de configuración del sistema Gnu-Monitor.
`m_server_name`	Contiene el nombre del server que se pasa como parámetro al arrancarlo.

7.2. Miembros de la clase `CGMServerWait`

7.2.1. CGMServerWait

Constructor de la clase.

7.2.2. ~CGMServerWait

Destructor de la clase.

7.2.3. Init

Inicializa los parámetros del server y la cola de mensajes utilizada para la comunicación con el monitor.

```
int Init(const char *server_name);
```

```
int Init(string &server_name);
```

El parámetro *server_name* indica el nombre con el cual el server se identifica ante el sistema Gnu-Monitor.

7.2.4. Wait

Se pone a la espera de recibir un mensaje desde el monitor.

```
int Wait(char *fn, void *data, unsigned long maxlen, unsigned long *datalen,
int to);
```

En el parámetro *fn* se devuelve el nombre del servicio o función convocada.

En el parámetro *data* se devuelven los datos asociados al servicio.

El parámetro *maxlen* indica al miembro canto espacio puede alocar como máximo en el buffer *data*.

En el parámetro *datalen* se devuelve el tamaño del mensaje recibido alojado en *data* .

El parámetro *to* indica al miembro el tiempo máximo de espera por el arribo de un mensaje, si este valor es 0 (cero) el llamado volverá inmediatamente aunque no haya mensajes, si es -1 la espera por mensajes será infinita, cualquier otro valor superior a cero indicará el tiempo de espera expresado en centésimas de segundo.

7.2.5. Resp

Responde al monitor el último mensaje recibido.

```
int Resp(const void *data, unsigned long datalen, int rc);
```

El parámetro *data* corresponde al mensaje que se le devolverá al cliente.

El parámetro *datalen* corresponde al tamaño del mensaje que se le devolverá al cliente.

El parámetro *rc* corresponde al código de error que se le devolverá al cliente.

7.2.6. Notify

Envía una notificación de evento a un grupo, cliente o usuario en particular.

```
int Notify(const char *dst, const char *data, unsigned int len);
```

El parámetro *dst* indica el nombre del grupo de destino para el mensaje.

El parámetro *data* es un puntero al mensaje a enviar.

El parámetro *len* es el tamaño del mensaje.

7.2.7. Broadcast

Notifica un evento a todos los grupos, clientes y usuarios que cumplan las condiciones de la máscara pasada como parámetro. Para este miembro se pueden utilizar los comodines * y & para formar los nombres de los destinatarios.

```
int Broadcast(const char *user, const char *client, const char *group, const
char *data, unsigned int len);
```

El parámetro *user* contiene la máscara para el usuario de destino del mensaje.

El parámetro *client* contiene la máscara para el cliente de destino del mensaje.

El parámetro *group* contiene la máscara para el grupo de destino del mensaje.

El parámetro *data* es un puntero al mensaje a enviar.

El parámetro *len* es el tamaño del mensaje.

7.2.8. Post

Envía un evento al monitor. Junto con el evento también se envía un bloque de datos que será reenviado a los servers y clientes suscritos al evento.

```
int Post(const char *event, const char *data, unsigned int len);
```

El parámetro *event* indica el nombre del evento posteado.

El parámetro *data* contiene el mensaje asociado al evento.

El parámetro *len* contiene el tamaño del mensaje asociado al evento.

7.2.9. Suscribe

Informa al monitor que el server atenderá los mensajes enviados para el evento o servicio y tipo de mensaje que se suscribe.

De esta forma la parametrización del ruteo se realiza de forma dinámica al levantarse cada servidor.

Está permitido que mas de un servidor atiendan el mismo servicio. En caso que el servicio utilice mensajes del tipo consulta/respuesta, interactivo o aviso el requerimiento se enviará al que tenga menos trabajos encolados y si el servicio utiliza mensajes de tipo eventos se enviará a todos los registrados.

```
int Suscribe(const char *event, char tipo_mensaje);
```

El parámetro *event* indica el nombre del servicio al que se suscribe.

El parámetro *tipo_mensaje* indica el tipo de mensaje para el sevicio.

7.2.10. UnSuscribe

Informa al monitor que el server deja de atender los requerimientos para el servicio indicado.

Deberá ser llamado al bajar el server para que el monitor deje de rutear los mensajes a este servidor.

```
int UnSuscribe(const char *event);
```

El parámetro *event* indica el nombre del servicio al que se desuscribe.

7.2.11. Call

Este miembro permite enviar un mensaje a la vez que espera la respuesta, de esta forma se realiza una consulta sincrónica y se cuenta un parámetro para especificar un tiempo límite para recibir la respuesta.

```
int Call(const char *fn, const char *query, unsigned int qlen, const char
**response, unsigned int *rlen, int to);


int Call(string *fn, CGMBuffer &query, CGMBuffer &response, int to);


int Call(string *fn, CGMBuffer *query, CGMBuffer *response, int to);
```

El parámetro *fn* contiene el nombre del servicio que se convoca.

El parámetro *query* contiene el mensaje que se asocia a la consulta al servicio.

El parámetro *qlen* contiene el tamaño del mensaje de consulta.

En el parámetro *response* se devuelve el mensaje de respuesta de ejecutar el servicio.

En el parámetro *rlen* se devuelve el tamaño de la respuesta.

El parámetro *to* indica el tiempo máximo expresado en centésimas de segundo que se esperará por la respuesta luego del cual el llamado terminará con error de time-out.

Para facilitar la implementación de los buffers para la comunicación se provee la clase `CGMBuffer` que es utilizada por el miembro `Call` entre otros.

7.3. Server waited de ejemplo

Para demostrar lo sencillo que es desarrollar un server waited para el sistema Gnu-Monitor se muestra a continución un ejemplo de uno que resuelve los timers programados del sistema.

Ejemplo 7-1. Server waited de ejemplo

```
#include <gmonitor/gmerror.h>
#include <gmonitor/gmconst.h>
```

```
#include <gmonitor/gmbuffer.h>
#include <gmonitor/cmsg.h>
#include <gmonitor/gmisc.h>
#include <gmonitor/gmontdb.h>
#include <gmonitor/gmstring.h>

#include "gmswaited.h"

#include <string>
#include <iostream>
using namespace std;

#include <unistd.h>
#include <signal.h>

CGMServerWait *m_pServer;
void OnClose(int sig);
char *m_pInBuffer;
char *m_pOutBuffer;

int main(int argc, char** argv, char** env)
{
  int rc;
  char fn[33];
  unsigned long inlen;
  unsigned long outlen;
  int wait;

  signal(SIGPIPE, SIG_IGN);
  signal(SIGKILL, OnClose);
  signal(SIGTERM, OnClose);
  signal(SIGSTOP, OnClose);
  signal(SIGABRT, OnClose);
  signal(SIGQUIT, OnClose);
  signal(SIGINT, OnClose);
  signal(SIGILL, OnClose);
  signal(SIGFPE, OnClose);
  signal(SIGSEGV, OnClose);
  signal(SIGBUS, OnClose);

  m_pInBuffer = NULL;
  m_pOutBuffer = NULL;
  m_pServer = new CGMServerWait;
  m_pServer->Init("gm_timer");
  m_pServer->m_pLog->Add(1, "Iniciando server de TIMERS");

  if(( rc =  m_pServer->Suscribe(".set_timer", 'M')) != GME_OK)
  {
    m_pServer->m_pLog->Add(1, "ERROR %i al suscribir servicio .set_timer", rc);
    OnClose(0);
  }
  if(( rc = m_pServer->Suscribe(".kill_timer", 'M')) != GME_OK)
  {
```

```
    m_pServer->m_pLog->Add(1, "ERROR %i al suscribir servicio .kill_timer", rc);
    OnClose(0);
  }

  m_pInBuffer = (char*)calloc(1024, sizeof(char));
  m_pOutBuffer = (char*)calloc(1024, sizeof(char));

  wait = 1;
  while((rc = m_pServer->Wait(fn, m_pInBuffer, 1024, &inlen, wait)) > 0)
  {
    if(rc > 0)
    {
      /* proceso el mensaje que llegó */
      sprintf(m_pOutBuffer, "[%s] -> ", fn);
      outlen = inlen;
      memcpy(m_pOutBuffer + strlen(m_pOutBuffer), m_pInBuffer, outlen);

      if(m_pServer->Resp(m_pOutBuffer, outlen, rc) != GME_OK)
      {
        /* error al responder */
      }
    }
    /* cosas que hago siempre */
  }
  OnClose(0);
  return 0;
}

void OnClose(int sig)
{
  m_pServer->m_pLog->Add(1, "Exit on signal %i", sig);
  if(m_pInBuffer) free(m_pInBuffer);
  if(m_pOutBuffer) free(m_pOutBuffer);
  m_pServer->UnSuscribe(".kill_timer");
  m_pServer->UnSuscribe(".set_timer");
  delete m_pServer;
  exit(0);
}
```

Capítulo 8. Clases auxiliares

Estas clases se crearon para facilitar la programación del sistema Gnu-Monitor y se encuentran en la librería estática libgmsh.a con la que están linkeados todos los programas del monitor, incluida la libreria del cliente.

8.1. Clase CGMError

Esta clase encapsula los mensajes correspondientes a los valores de error devueltos por la interface del monitor, ya sean las clases del cliente o las de los servers.

8.1.1. Message

Devuelve la cadena del mensaje correspondiente al error del sistema Gnu-Monitor según el indice que se le pasa por parámetro.

```
string Message(int error);
```

Devuelve la cadena del mensaje correspondiente al último error declarado del sistema Gnu-Monitor.

```
string Message(void);
```

8.1.2. Last

Permite declarar un error para que este sea el utilizado por defecto por `Message` y al convocarlo sin parámetros devuelve el último error declarado.

```
int Last(int error);
```

Devuelve el valor del último error declarado.

```
int Last(void);
```

8.1.3. Constantes de Error

Para facilitar la escritura de los clientes y los servers se declararon constantes con los valores de error utilizados y se reservó un área para continuar agregando.

Tabla 8-1. Constantes de Error

Constante	Valor	Descripción
`GME_OK` y `GME_NO_ERROR`	0	Sin error.
`GME_UNDEFINED`	1	Error aún no definido
`GME_FCN_NOTFOUND`	2	No hay servidores declarados que resuelvan la combinación de servicio y tipo solicitado.
`GME_SVR_NOTFOUND`	3	No hay servidores levantados que resuelvan la combinación de servicio y tipo solicitado. La diferencia con `GME_FCN_NOTFOUND` es que en este caso el servicio y tipo están declarados pero el servidor que debe resolverlo no se encuentra disponible.
`GME_SVC_NOTFOUND`	4	*** COMPLETAR ***
`GME_MSGQ_ERROR`	5	Error en sistema de colas de mensajes.
`GME_HOST_NOTFOUND`	6	Error en la resolución de nombre del sistema monitor.
`GME_INVALID_HOST`	7	El nombre pasado para el monitor no es un nombre de red válido.
`GME_NOT_CONNECTED`	8	Se intentó enviar un mensaje sin estar conectado.
`GME_COMM_ERROR`	9	Error de comunicación (de protocolo).
`GME_SVC_DB_ERROR`	10	Error en la base de datos de servicios.
`GME_INVALID_HEADER`	11	Error de decodificación o integridad del header del mensaje.
`GME_INVALID_MESSAGE`	12	Error de decodificación o intergridad del mensaje.

Constante	Valor	Descripción
GME_TRAN_NOT_ALLOWED	13	Intento de comenzar una segunda transacción sin haber terminado la primera. El sistema no permite anidar transacciones.
GME_UNKNOWN_TRAN	14	Se intentó continuar con un transacción que no estaba iniciada o ya venció.
GME_INVALID_TRAN	15	Error al intentar iniciar una transacción.
GME_TRAN_NOT_INIT	16	Se intenta concluir una transacción que no fué iniciada o ya venció.
GME_ERROR_MAX	100	Indica el espacio reservado para futuros códigos de error.
GME_USER_ERROR	GME_ERROR_MAX	Valor para para comenzar a definir valores de error de usuario.

8.2. Clase CGMBuffer

Esta clase encapsula un buffer de uso genérico utilizado para almacenar los datos enviados y recibidos en distintas secciones del sistema.

8.2.1. CGMBuffer

Al constuir el objeto del buffer se le puede asignar un contenido inicial con alguno de los siguientes modelos.

```
CGMBuffer(const char* str);
```

```
CGMBuffer(const void* ptr, unsigned long len);
```

```
CGMBuffer(string s);
```

El parámetro *str* corresponde a un string terminado en nul.

El parámetro *ptr* corresponde a un puntero y se incorporará al objeto todo el contenido apuntado hasta *len*.

El parámetro *len* corresponde al tamaño del dato en *ptr*.

El parámetro *s* corresponde a un string.

8.2.2. Clear

Permite limpiar el contenido del objeto borrando todo su contenido y liberando la memoria alocada.

```
void Clear(void);
```

8.2.3. Operador =

Asigna el dato al objeto buffer. Devuelve un puntero al buffer interno o NULL en caso de error.

```
const char* operator=(const char* str);
```

```
const char* operator=(CGMBuffer buffer);
```

```
const char* operator=(string s);
```

El parámetro *str* corresponde a un string terminado en nul.

El parámetro *buffer* corresponde a otro objeto CGMBuffer.

El parámetro *s* corresponde a un string.

8.2.4. Operador +=

Agrega el dato al final del contenido actual del objeto buffer. Devuelve un puntero al buffer interno o NULL en caso de error.

```
const char* operator=(const char* str);

const char* operator=(CGMBuffer buffer);

const char* operator=(string s);
```

El parámetro *str* corresponde a un string terminado en nul.

El parámetro *buffer* corresponde a otro objeto CGMBuffer.

El parámetro *s* corresponde a un string.

8.2.5. Set

Asigna el dato al objeto buffer. Devuelve un puntero al buffer interno o NULL en caso de error.

```
const char* Set(const char* str);

const char* Set(const void* ptr, unsigned long len);

const char* Set(CGMBuffer buffer);

const char* Set(string s);
```

El parámetro *str* corresponde a un string terminado en nul.

El parámetro *ptr* corresponde a un puntero y se incorporará al objeto todo el contenido apuntado hasta *len*.

El parámetro *len* corresponde al tamaño del dato en *ptr*.

El parámetro *buffer* corresponde a otro objeto CGMBuffer.

El parámetro *s* corresponde a un string.

8.2.6. Add

Agrega el dato al final del contenido actual del objeto buffer. Devuelve un puntero al buffer interno o NULL en caso de error.

```
const char* Add(const char* str);


const char* Add(const void* ptr, unsigned long len);


const char* Add(CGMBuffer buffer);


const char* Add(string s);
```

El parámetro *str* corresponde a un string terminado en nul.

El parámetro *ptr* corresponde a un puntero y se incorporará al objeto todo el contenido apuntado hasta *len*.

El parámetro *len* corresponde al tamaño del dato en *ptr*.

El parámetro *buffer* corresponde a otro objeto CGMBuffer.

El parámetro *s* corresponde a un string.

8.2.7. Format

Permite setear al buffer con un string formateado al estilo `printf`. Devuelve un puntero al buffer interno o NULL en caso de error.

```
const char* Format(const char* fmt, ... );
```

El parámetro *fmt* es una mascara de formato con las mismas prestaciones que la utilizada en `printf`.

El parámetro `...` es una lista de argumentos variable separados por "," .

8.2.8. AddFormat

Permite agregar al contenido actual del buffer un string formateado al estilo `printf`. Devuelve un puntero al buffer interno o NULL en caso de error.

```
const char* AddFormat(const char* fmt, ... );
```

El parámetro *`fmt`* es una mascara de formato con las mismas prestaciones que la utilizada en `printf`.

El parámetro `...` es una lista de argumentos variable separados por "," .

8.2.9. Length

Devuelve el tamaño del dato alojado en el objeto buffer.

```
unsigned long Length(void);
```

8.2.10. Data

Devuelve un puntero al buffer interno donde se aloja el dato.

```
const char* Data(void);
```

8.2.11. C_Str

Devuelve un puntero al buffer interno donde se aloja el dato pero asegurandose de que corresponda a un string terminado en nul.

```
const char* C_Str(void);
```

8.2.12. String

Devuelve string compuesto por el contenido del buffer.

```
string String(void);
```

8.3. Clase CShMem

Esta clase encapsula el manejo de memoria compartida.

8.3.1. CShMem

Es el contructor de la clase, permite asignarle una clave en el momento de la creación, sino puede hacerse después.

```
CShMem(void);
```

```
CShMem(int key);
```

El parámetro *key* es el valor de la clave.

8.3.2. Key

Permite asignar una clave al objeto luego de su creación, esto es útil cuando se usa esta clase como un objeto estático y se le quiere asignar la clave en el momento de ejecución.

```
int Key(int key);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *key* es el valor de la clave.

8.3.3. Create

Crea un area de memoria compartida del tamaño indicado y la inicializa en blanco (toda con nul).

```
int Create(int length);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *length* indica el tamaño del bloque de memoria a crear.

8.3.4. Open

Se engancha de un área de memoria compartida previamente creada.

```
int Open(int length);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *length* indica el tamaño del bloque de memoria donde se engancha.

8.3.5. Close

Si el objeto corresponde al creador del área de memoria compartida dicha área es destruida, en cambio si no lo es solamente se desengancha.

```
int Close(void);
```

Devuelve 0 para confirmar o -1 en caso de error.

8.3.6. SetAt

Escribe en el área de memoria compartida en una posición determinada. Este miembro solo puede ser utilizado por una clase derivada.

```
int SetAt(int pos, const void* data, int length);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *pos* indica la posición inicial donde escribir los datos.

El parámetro *data* es un puntero al dato a escribir en la memoria compartida.

El parámetro *length* indica el tamaño del dato en *data*.

8.3.7. GetAt

Lee del área de memoria compartida desde una posición determinada. Este miembro solo puede ser utilizado por una clase derivada.

```
int GetAt(int pos, void* data, int length);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *pos* indica la posición inicial de donde comenzar a leer los datos.

El parámetro *data* es un puntero a donde escribir los datos leidos de memoria compartida.

El parámetro *length* indica el tamaño del dato a copiar en *data*.

8.4. Clase CSincro

Esta clase encapsula el manejo de semaforos.

8.4.1. CSincro

Es el contructor de la clase, permite asignarle una clave en el momento de la creación, sino puede hacerse después.

```
CSincro(void);
```

```
CSincro(int key);
```

El parámetro *key* es el valor de la clave.

8.4.2. Key

Permite asignar una clave al objeto luego de su creación, esto es útil cuando se usa esta clase como un objeto estático y se le quiere asignar la clave en el momento de ejecución.

```
int Key(int key);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *key* es el valor de la clave.

8.4.3. Create

Crea un grupo de semaforos seteandole su estado inicial, el valor por defecto es WAIT.

```
int Create(int count, int init_val);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *count* indica la cantidad de semaforos a crear.

El parámetro *init_val* indica el valor inicial a asignarle a los semaforos.

8.4.4. Open

Se engancha a un grupo de semaforos creado.

```
int Open(int count);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *count* indica la cantidad de semaforos del grupo.

8.4.5. Close

Si el objeto corresponde al creador de los semaforos el grupo es destruido, en cambio si no lo es solamente se desengancha.

```
int Close(void);
```

Devuelve 0 para confirmar o -1 en caso de error.

8.4.6. Wait

Realiza un WAIT sobre el semaforo indicado.

```
int Wait(int sem);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *sem* es el índice a un semáforo del grupo.

8.4.7. Signal

Realiza un SIGNAL sobre el semaforo indicado.

```
int Signal(int sem);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *sem* es el índice a un semáforo del grupo.

8.4.8. Set

Permite asignarle un valor a semaforo.

```
int Set(int sem, int val);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *sem* es el índice a un semáforo del grupo.

El parámetro *val* es el valor a asignarle al semaforo.

8.5. Clase CSincMem

Esta clase permite hacer un manejo de memoria compartida sincronizada por medio de semaforos, para esto se deriva de `CShMem` y `CSincro`.

8.5.1. CSincMem

Es el contructor de la clase, permite asignarle una clave en el momento de la creación, sino puede hacerse después.

```
CSincMem(void);
```

```
CSincMem(int key);
```

El parámetro *key* es el valor de la clave.

8.5.2. Key

Permite asignar una clave al objeto luego de su creación, esto es útil cuando se usa esta clase como un objeto estático y se le quiere asignar la clave en el momento de ejecución.

```
int Key(int key);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *key* es el valor de la clave.

8.5.3. Create

Crea un area de memoria compartida del tamaño indicado y la inicializa en blanco (toda con nul).

```
int Create(int length);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *length* indica el tamaño del bloque de memoria a crear.

8.5.4. Open

Se engancha de un área de memoria compartida previamente creada.

```
int Open(int length);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *length* indica el tamaño del bloque de memoria donde se engancha.

8.5.5. Close

Si el objeto corresponde al creador del área de memoria compartida dicha área es destruida, en cambio si no lo es solamente se desengancha.

```
int Close(void);
```

Devuelve 0 para confirmar o -1 en caso de error.

8.5.6. SetAt

Escribe en el área de memoria compartida en una posición determinada. Este miembro solo puede ser utilizado por una clase derivada.

```
int SetAt(int pos, const void* data, int length);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *pos* indica la posición inicial donde escribir los datos.

El parámetro *data* es un puntero al dato a escribir en la memoria compartida.

El parámetro *length* indica el tamaño del dato en *data*.

8.5.7. GetAt

Lee del área de memoria compartida desde una posición determinada. Este miembro solo puede ser utilizado por una clase derivada.

```
int GetAt(int pos, void* data, int length);
```

Devuelve 0 para confirmar o -1 en caso de error.

El parámetro *pos* indica la posición inicial de donde comenzar a leer los datos.

El parámetro *data* es un puntero a donde escribir los datos leidos de memoria compartida.

El parámetro *length* indica el tamaño del dato a copiar en *data*.

8.6. Clase CGLog

Esta clase encapsula las funciones de logeo del sistema monitor, pero puede utilizarse para generar otros logs de usuario.

Los server del monitor ya tienen un log abierto que puede ser utilizado a través del objeto `m_pLog`.

8.6.1. CGLog

Este es el contructor de la clase, a través de sus parámetros se puede definir el nombre del proceso, archivo donde se va a loguear y el nivel se logeo.

```
CGLog(const char* appname, const char* logpath, unsigned int loglevel);
```

El parámetro *appname* permite indicar el nombre de la aplicación, real o de fantasía.

Este nombre aparecerá al principio de la linea del log para permitir la identificación en caso que se utilice un mismo log para varios procesos.

8.6.2. Add

Permite agregar una nueva línea al log utilizando una cadena de formato.

```
void Add(unsigned int level, const char* msg, ... );
```

El parámetro *level* permite indicar el nivel de importancia del mensaje, si este nivel es de igual o mayor importancia que el elegido al crear el objeto entonces se escribe en el log.

El parámetro *msg* es una cadena de formato similar a la utilizada en un `printf`.

El parámetro `...` es una lista de argumentos variable separados por "," .

8.6.3. AddBin

Permite agregar una nueva línea al log con contenido binario.

```
void AddBin(unsigned int level, const char* id, const void* buffer, unsigned
int len, ... );
```

El parámetro *level* permite indicar el nivel de importancia del mensaje, si este nivel es de igual o mayor importancia que el elegido al crear el objeto entonces se escribe en el log.

El parámetro *id* permite ponerle una etiqueta adicional al dato que se graba.

El parámetro *buffer* corresponde al dato a grabar en el log.

El parámetro *len* corresponde al tamaño del dato en `buffer`.

Capítulo 9. Desarrollo de Gnu-Monitor

Este capítulo está orientado a servir de ayuda para los miembros del grupo de desarrollo de Gnu-Monitor.

9.1. Uso de CVS

Para facilitar la taréa de desarrollo cooperativo los fuentes del sistema Gnu-Monitor se encuentran alojados en un server CVS provisto de forma gratuita por Source Forge.

9.1.1. Acceso anónimo

Este tipo de acceso permite crearse en forma local una imagen del CVS pero util solamente a fines de compilar la versión de desarrollo del sistema ya que por este método no será posible realizar actualizaciones a los fuentes en el CVS oficial del proyecto.

Para obtener esta imagen del CVS debe logearse de forma anónima con: **cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/gnu-monitor login** y luego traerse el árbol de fuentes con: **cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/gnu-monitor co `module-name`**, donde `module-name` puede ser `system`, `document` o `samples` según lo que se desee obtener.

9.1.2. Acceso para desarrolladores

Este tipo de acceso es el que permie el desarrollo cooperativo tanto del sistema principal del monitor transaccional como de los distintos manuales que acompañan al sistema y sus programas de ejemplo.

Para utilizar este tipo de acceso al CVS de Gnu-Monitor es necesario contar con un usuario que pertenezca al grupo de desarrollo y cuente con los permisos necesarios otorgados por el administrador.

Previo a la utilización del CVS para desarrolladores debe ser definida la variable de entorno `CVS_RSH` con el valor `ssh` ejecutando **export CVS_RSH=ssh**.

El primer paso es obtener el árbol de fuentes del cvs. Al igual que para el acceso anónimo se deberá ejecutar el comando para cada uno de los módulos que componen el sistema y que se deseen obtener. Esto se logra ejecutando: **cvs -z3 -d:ext:`developper-name`@cvs.sourceforge.net:/cvsroot/gnu-monitor co `module-name`** donde como ya se mencionó anteriormante `module-name` puede ser `system`, `document` o `samples` según lo que se desee obtener.

En adelante para mantener actualizado el árbol de fuentes basta con ejecutar **cvs update** y para actualizar en el CVS los cambios hechos en los fuentes en forma local se ejecuta **cvs commit** ambos comando se deberán ejecutar dentro del directorio que se quiere actualizar y son recursivos.

En caso que sea necesario agregar o quitar archivos al desarrollo, esto se indica al CVS por medio de los comandos **cvs add `filename`** y **cvs remove `filename`**. Luego siempre será necesario ejecutar **cvs commit** para que se impacten los cambios.

Apéndice A. GNU Free Documentation License

A.1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any

mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this

License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

A.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

A.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

A.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

A.10. TERMINATION

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

A.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.12. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.