
# DATATEC API – DAPI

# Guía del programador

## Sistema de Mercados Financieros

## Update 3.008

## Colombia

**2023-May-26**
**ICAP del Ecuador**
**support@datatec-ec.com**

# INDICE

# INDICE DE FIGURAS

# 1. Historia documento

| Fecha | Autor | Descripción |
|---|---|---|
| 18-Sep-2007 | Equipo de desarrollo | Documento inicial. |
| 20-Abr-2009 | Equipo de desarrollo | Actualización con características versión 1.056. |
| 08-Ene-2010 | Equipo de Desarrollo | - Revisión con cambios versión 1.061.<br>- Nuevo parámetro *guibos_feed* para mensajes de Guibos.<br>- Nuevo parámetro *Send_Msg_Delay_Millisec*.<br>- Actualización de valores de configuración por omisión (para pedido de mensajes de log de alta/baja prioridad).<br>- Actualización y explicación de otros valores de configuración.<br>- Cambios en la aplicación de ejemplo de Windows para mensajes Guibos y reenvío al sistema de ciertos mensajes.<br>- Actualización en la sección de compilación en Linux de la librería/aplicación de ejemplo.<br>- Nuevo script **mk**. |
| 28-Jun-2010 | Equipo de Desarrollo | - Revisión general del documento.<br>- Notas aumentadas para la recompilación librería Linux (versiones gcc y soporte sistemas operativos 32-64 bits).<br>- Nota soporte Windows Vista (32bits) y 7 (32bits).<br>- DAPI 1.065. Ajustes generales para usar DAPI en lugar de DGAPI en ambientes que reciben mensajes Guibos.<br>- DAPI 1.066. Dapi envía su versión en la conexión para que sea desplegada en la ventana de versiones en el DMcliente. |
| 20-Sep-2010 | Equipo de Desarrollo | - Sección 8.3 se aumenta Diagrama de flujo ejemplo para recoger información de una transacción para aplicaciones back-office.<br>- Sección 9 se aumenta configuración del DAPI en el DMCliente.<br>- DAPI 1.069 Soporte mensaje nuevo Mantenimiento_Transaccion_Renta_Fija.<br>- Archivo “20100920 DAPI Interface Message Descriptions.xls” |
| 28-Oct-2010 | Alvaro Vela | - Revisión general. |
| 30-Jun-2011 | Equipo de Desarrollo | - Instrucciones de ejecución ejemplo en Windows 64 bits (sección 11.3.1). Regeneración ejemplo Windows (v. 1.077).<br>- Revisión versión DAPI en archivo OCX (sección 11.1.1). |

| | | |
|---|---|---|
| 22-Dec-2017 | Álvaro Vela | - Actualización de documento en:<br>Plataformas de Desarrollo e Instalación.<br>Código de ejemplo del DAPI.<br>Instalación del DAPI en Windows.<br>Instalación del DAPI en Linux.<br>Configuración por omisión en Linux.<br>OpenSSL.<br>Instalación y uso de la librería DAPI en Linux.<br>Compilación de la librería DAPI en Linux.<br>Compilación de la aplicación ejemplo en Linux |
| 29-Oct-2018 | Álvaro Vela | - Revisión general. |
| 21-Nov-2019 | Equipo de Desarrollo | - Implementación de uso de nuevo archivo temporal de configuración, utilizado para comentar con una nueva retransmisión completa de mensajes de un día en el pasado.<br>- Las directivas log_days_high_messages y log_days_low_messages definidas en el archivo dapi.cfg son eliminadas. |
| 25-Jun-2020 | Equipo de Desarrollo | - Implementación de nuevas funciones:<br>GetCompilationType<br>GetVersion<br>- Documentación de la función IsDefined.<br>- Generación del ejemplo Visual Basic con |
| 08-Nov-2022 | Equipo de Desarrollo | - Implementación del mensaje chat (575) |
| 17-Mar-2023 | Equipo de Desarrollo | - Generación del programa drvapitest para Windows x86 y x64. Implementación del mensaje chat (575) |
| 15-May-2023 | Equipo de Desarrollo | - Compatibilidad con Visual Studio 2019. |
| 15-May-2023 | Equipo de Desarrollo | Nuevas funciones para que sean compatibles con 64 bits. Declaración para 32 bits. Ver:<br>- Método *FinishMessageProcess*<br>- *M*étodo *TakeMessage* |
| 26-May-2023 | Álvaro Vela | Revisión general. |

# 2. Objetivo

El objetivo de este documento es brindar una explicación general de la Interfaz para Programas de Aplicación de los Sistemas Datatec, conocido como Datatec API o DAPI. Incluye una Guía de Referencia para Programadores del API y una explicación de las aplicaciones de ejemplo provistas para ambas distribuciones. Adicionalmente, provee los pasos de instalación y configuración del DAPI en las plataformas Windows y Linux.

# 3. Descripción general del Sistema Datatec

El DAPI (Datatec API), como su nombre lo sugiere, es una Interfaz para Programas de Aplicación que permite que el Sistema de Datatec pueda recibir y transmitir información desde y hacia sistemas externos. De esta forma, el DAPI permite establecer un canal de comunicación entre el Sistema de Mercados Financieros (SMF) desarrollado por Datatec y cualquier otro sistema. El SMF es una plataforma distribuida para el negocio electrónico de instrumentos financieros.

Las operaciones de comercio son realizadas a través de mensajes enviados en un ambiente tipo cliente servidor. El SMF utiliza el protocolo TCP/IP para enviar mensajes, por lo tanto, el envío de cada mensaje está garantizado. El siguiente diagrama ilustra cómo se distribuye el sistema a lo largo de varias ubicaciones físicas y como cada componente interactúa con los demás.

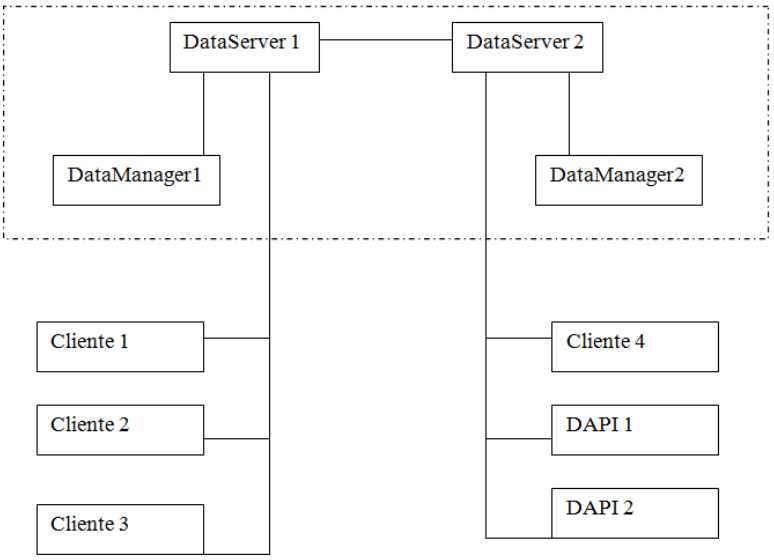



Figura 1: Diagrama general de la arquitectura SMF.

Como se puede notar en la figura anterior, existen varios componentes en la aplicación SMF. Estos pueden ser clasificados como componentes Cliente y componentes Servidor.

Para desarrollar y depurar la interfaz con el DAPI es necesario entender a un nivel general como se integra el DAPI dentro del Sistema SMF y con qué elementos interactúa. La siguiente es una explicación general de la arquitectura del SMF.

## 3.1. Componentes servidor del sistema.

**Data Server:** Actúa como la unidad de intercambio y reenvío de mensajes que integra las conexiones de varios componentes cliente. Se pueden usar múltiples DataServers para distribuir la carga de conexiones de clientes tal como se muestra en la Figura 1. El DataServer puede ser comparado vagamente con un ruteador, ya que mantiene las conexiones de los clientes y reenvía los mensajes del sistema.

**DataManager Server:** Aplicación responsable de validar la autenticación de un usuario, del mantenimiento del perfil del usuario, mantenimiento de los archivos maestros, cálculo de las estadísticas de mercado, y otras funciones relacionadas.

**Motor de Calce:** Encargada de procesar las operaciones y el mantenimiento de cupos de negociación.

## 3.2. Componente cliente

Todos los usuarios finales del Sistema SMF se conectan al DataServer a través de la aplicación cliente.

Esta aplicación tiene una interfaz gráfica que permite a los usuarios ver y participar en los mercados electrónicos en tiempo real. Los operadores de mercado ponen sus órdenes (ofertas y/o demandas) en el mercado y estas son observadas por todos los clientes conectados. Luego, otros usuarios pueden negociar estas posturas.

La aplicación cliente se comunica con el Sistema a través de una variedad de mensajes. Estos mensajes pueden ser enviados del cliente al sistema y viceversa.

## 3.3. Interfaz para programas de aplicación Datatec (DAPI)

El DAPI es un tipo particular de cliente cuyas funciones varían ligeramente de un cliente normal. El DAPI no posee una interfaz gráfica, sin embargo, tiene la capacidad de enviar y recibir los mensajes del Sistema. El DataServer considera al DAPI como cualquier otro cliente.

Para que el DAPI pueda interactuar con el sistema SMF se debe crear una aplicación que encapsule el DAPI y realice las llamadas a los métodos apropiados -los detalles de los métodos y eventos disponibles se describen en una sección posterior de este documento-. Por otra parte, esta aplicación debe interactuar también con el sistema externo tal como se muestra en el siguiente diagrama:

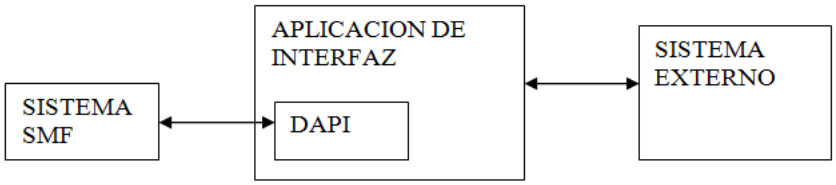



Figura 2: Interfaz del Sistema SMF con un sistema externo a través del DAPI.

Cuando ocurre un cambio en el mercado (por ejemplo: ingreso, modificación o retiro de una postura, negociación, etc.) el Sistema SMF envía un mensaje al DAPI que puede ser leído por la aplicación de interfaz y enviado al sistema externo.

Como en cualquier sistema orientado a conexión, el SMF depende de la existencia de una conexión entre el DataServer y los clientes (incluyendo el DAPI) para operar exitosamente. El DAPI implementa su propio protocolo de Recuperación de Transmisión en el caso de que se produzca una pérdida de conexión. Una vez que el mensaje es recibido por el DAPI, este es puesto en una cola para ser procesado por la aplicación del cliente. El DAPI notifica el arribo de un mensaje a través del disparo de un **Evento**. La aplicación cliente debe ser programada para interpretar estos eventos y recolectar los mensajes que sean de interés usando los **Métodos** provistos por el DAPI y luego

transmitir la información apropiada al sistema externo. Además, el DAPI provee los mecanismos necesarios para que el sistema externo pueda enviar mensajes al Sistema SMF.

# 4. Plataforma donde se puede ejecutar el DAPI

## 4.1. Plataformas de desarrollo

El DAPI está disponible para las siguientes plataformas de desarrollo:

1. ActiveX Control Windows (OCX): Compilado usando Microsoft Visual Studio 2019.
2. Librerías Windows (LIB): Compilado usando Microsoft Visual Studio 2019.
3. Librerías Linux: En este caso, así como en el anterior, el desarrollador debe generar aplicaciones tipo **multithreaded**.

**Observaciones adicionales:**

Las librerías dcapi.lib y crypt.lib son librerías estáticas e incluyen todas sus librerías de dependencia.

El archivo dxapi.ocx, que se incorpora como un objeto COM también incluye todas sus dependencias

El programa drvapitest (C++) de ejemplo, requiere las librerías para 32 y 64 bits dcapi.lib y crypt.lib. Y las librerías del sistema en Windows: kernek32.dll y wsock32.dll

El archivo drvapitest de Linux a más de la librería dcapi.lib, requiere las siguientes librerías del sistema:

- linux-gate.so.1
- libpthread.so.0
- libnsl.so.1
- libresolv.so.2
- librt.so.1
- libdl.so.2
- libm.so.6
- libgcc_s.so.1
- libc.so.6

Existen múltiples librerías que, dependiendo de la aplicación c++ que se genera, serán requeridas por Visual Studio 2019, por ejemplo:

- USER32.dll
- GDI32.dll
- MSIMG32.dll
- WINSPOOL.DRV
- ADVAPI32.dll
- SHELL32.dll
- SHLWAPI.dll
- UxTheme.dll
- ole32.dll
- OLEAUT32.dll
- gdiplus.dll
- OLEACC.dll
- IMM32.dll
- WINMM.dll
- WS2_32.dll

## 4.2. Plataformas de instalación

El DAPI como ActiveX control (OCX) corre en los siguientes sistemas operativos.

- Microsoft Windows 10-11 (32/64 bits)

El DAPI como librería, se puede usar en los siguientes sistemas operativos:

- Linux 7, 8 (32/64 bits)
- Windows 10, 11 (32/64 bits)

## 4.3. Requerimientos mínimos de hardware para uso del DAPI

La aplicación DAPI actúa como un Usuario Primario del sistema Datatec y tiene los mismos requerimientos de hardware. Se recomienda:

- Procesador: Dual core o superior.
- Memoria: 4 GB o superior.
- Disco duro: Al menos 100 GB disponibles.

# 5. Cómo interactuar con el DAPI

La secuencia lógica de métodos a ser invocados por la aplicación cliente para envío/recepción de mensajes del sistema es:

**1.-INGRESO:** El primer método que debe invocarse es *Login(). Este* método requiere el nombre de usuario y la clave como datos de entrada. Este método inicia varios sub-procesos que mantienen una conexión asincrónica con el DataServer. Cualquier error que se produzca durante el proceso de conexión será reportado por el disparo del evento *OnError()*. El evento *OnConnection()* reporta el resultado de cada intento de ingreso. Luego de una conexión exitosa el método *Login()* devolverá el valor **DAPI_INITIALIZED** y el evento *OnConnection()* reportará como resultado un valor 0.

**2.- RECEPCION DE UN MENSAJE:** Cuando el DAPI se encuentra conectado, cada mensaje que el sistema le remita disparará el evento *OnMessageReceived()*.

**3.- ACCEDIENDO A LOS DATOS RECIBIDOS:** Para acceder a la información contenida en el mensaje recibido se debe invocar el método *TakeMessage().* Cada campo dentro del mensaje puede ser leído individualmente usando el método *GetField()*.
**4.- MARCANDO EL MENSAJE COMO PROCESADO:** Una vez que el mensaje ha sido leído exitosamente se debe marcar el mensaje utilizando el método *FinishMessageProcess()* para que no vuelva a ser retransmitido por el DataServer en el caso de que se produzca una pérdida de comunicación y una reconexión al sistema. En el caso de una pérdida inesperada de conexión el sistema no retransmitirá al DAPI los mensajes que han sido confirmados por este.

**5.- ENVIANDO MENSAJES AL DATASERVER:** Para preparar un mensaje que va a ser enviado al DataServer por el DAPI, este mensaje debe ser inicializado con el método *InitMessage().* Luego de una inicialización exitosa los campos deben ser llenados con el método *SetField().* De ser necesario, se especifican los destinatarios del mensaje con el método *SendMessageTo()*. Luego de que el mensaje está listo para ser enviado al sistema se debe utilizar el método *SendMessage()*. Si el método *InitMessage()* es llamado sin ningún parámetro (con el argumento “ ”) se realiza una copia del último mensaje tomado con el método *TakeMessage().*

**6.- SALIDA:** El método *Logout()* produce una desconexión normal del DAPI. En el evento que *Logout()* sea llamado antes que el método *FinishMessageProcess*() termine de procesar un mensaje recibido, éste mensaje pendiente será retransmitido en la siguiente reconexión al sistema.

El DAPI guarda información persistente en los archivos de registro (d*api.log*). Esta información es de utilidad para el proceso de recuperación de transmisión de información en caso de pérdida o reconexión. La longitud de estos archivos no corresponde al tamaño usado en disco debido a que se usa posicionamiento aleatorio en lugar de posicionamiento secuencial. ***En condiciones normales estos archivos NO deben ser borrados.***

# 6. Verificación de transmisión/recepción de mensajes

Antes de enviar cada mensaje al DataServer el DAPI almacena el mensaje en un archivo. En el caso de una reconexión, el DAPI usa este archivo para decidir lo que debe retransmitir. Existen archivos adicionales dentro de la estructura de directorio del DAPI –uno por cada DataServer- que almacenan los mensajes recibidos y procesados por la aplicación que usa el DAPI para interactuar con el Sistema, esta debe tener los privilegios necesarios para poder crear y modificar estos archivos.

## 6.1. Pérdidas de conexión inesperadas

Esta situación se produce cuando el DAPI pierde conexión con el DataServer debido a causas externas. En estas circunstancias, el DAPI intenta reanudar la conexión con el DataServer y cuando esto ocurre recibe del sistema todos los mensajes que no han sido marcados como procesados. En el caso de que un mensaje que ha sido marcado como procesado por el DAPI sea recibido nuevamente, no se disparará el evento *OnMessageReceived()*.

## 6.2. Pérdidas de conexión programadas

Esto sucede cuando la aplicación llama el método del DAPI *Logout()*. En este caso, la memoria ocupada por el DAPI es liberada. Cerrar y volver a reiniciar la aplicación que utiliza el DAPI se considera como un evento de pérdida de conexión programada.

# 7. Métodos y eventos del DAPI

## 7.1. Métodos del DAPI

Métodos son las funciones que el ActiveX control y la clase de la librería disponen para ser llamados desde la aplicación. La aplicación debe llamar a los métodos del control en el caso del ActiveX control, o a los métodos de una clase base en el caso de la Librería para ordenar el proceso que el método indica.

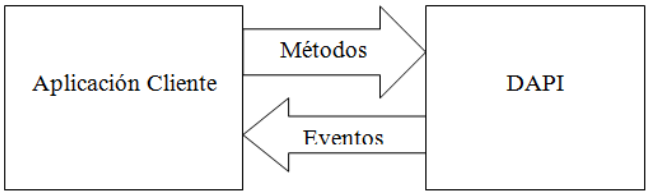



Figura 3: Interacción de la aplicación con el DAPI.

A continuación, se describen los métodos del ActiveX control del DAPI CdxapiCtrl y los métodos de la librería CdcApi. Cada método se presenta primero en formato para el control ActiveX (OCX) y luego en formato C++ para la librería.

### 7.1.1. Método *AddLoginCondition*

VARIANT CdxapiCtrl::**AddLoginCondition** (BSTR *message_type*, BSTR *field_name,* BSTR *cond_operation*, BSTR *eq_value*)
int CdcApi::**AddLoginCondition** (LPCTSTR *message_type*, LPCTSTR *field_name*, LPCTSTR *cond_operation,* LPCTSTR *eq_value*);

**Valor de retorno**
Retorna 0 si no hay errores. En caso contrario retorna alguno de los siguientes errores:
DAPI_ERR_NOT_INITIALIZED_YET
DAPI_ERR_ILLEGAL_RECORD_TYPE_SPECIFIED
DAPI_ERR_ILLEGAL_FIELD_NAME_SPECIFIED

Un error de evaluación de la condición también puede generar este valor:
DAPI_ERR_CONDITION_EVALUATION

**Parámetros**
*message_type:* Indica el tipo de mensaje al que pertenece el campo de nombre *field_name*.

*field_name:* Nombre del campo dentro del mensaje que se va a evaluar con la condición.

Se puede definir una operación lógica entre dos condiciones con los siguientes operadores:

| Operador | Significado |
|---|---|
| "\|" | O LOGICO. |
| "&" | Y LOGICO. |

El resultado puede usarse en una siguiente operación lógica según el formato de ejecución BFN.

*cond_operation:* Define la operación a realizar entre el campo (*field_name*) y *eq_value* para obtener el resultado de la condición. Los operadores condicionales pueden ser:

| Operador | Significado |
|---|---|
| "==" | Verdadero cuando ambos operandos son iguales en contenido |
| "=" | Verdadero cuando ambos operandos son iguales en contenido |
| "LIKE" | Operador de comparación. Permite buscar por patrones de texto dentro de un campo. El patrón de comparación (el segundo operando), puede contener los siguientes caracteres, los mismos que tienen un significado diferente según se indica:<br>%: Significa 0 o más letras, cualesquiera sean<br>*: Significa cualquier letra<br>-: Significa cualquier letra<br>[..]: significa cualquiera de las letras colocadas dentro de los [].<br>[^...]: significa cualquier letra excepto las que se colocan dentro de los []. |
| "NOT LIKE | Inverso lógico del operador de comparación LIKE de SQL. Este operador se describe anteriormente bajo "LIKE" |

*eq_value:* Es el valor que será comparado contra el campo *field_name* para determinar si la condición es verdadera.

**Descripción**
Este método permite seleccionar los mensajes que son recibidos por la aplicación. Los mensajes que no cumplen la condición son desechados por el DAPI y no son entregados a la aplicación.

Ejecutar este método las veces que sean necesarias al inicio de la ejecución; esto es, antes de llamar a el método *Login()*. Se puede ejecutar este método varias veces para indicar diferentes condiciones de aceptabilidad. Usar *field_name* con valores '&' o '|'para indicar operaciones lógicas entre dos comparaciones previamente definidas. Finalmente, se debe considerar la notación RPN (Reverse Polish Notation, o Notación Polaca Invertida) a fin de controlar el resultado final entre todas las condiciones de aceptabilidad. Por ejemplo:

```
L_msType = "Guibos";
m_dapi.AddLoginCondition (L_msType, "instrument",    "==",   "SWAP");
m_dapi.AddLoginCondition (L_msType, "instrument",    "==",   "NDF");
m_dapi.AddLoginCondition (L_msType, "|",             "",     "");
m_dapi.AddLoginCondition (L_msType, "buyer_dealer",  "LIKE", "AAAL%");
m_dapi.AddLoginCondition (L_msType, "seller_dealer", "LIKE", "BBBL%");
m_dapi.AddLoginCondition (L_msType, "|",             "",     "");
m_dapi.AddLoginCondition (L_msType, "&",             "",     "");
```

Las instrucciones anteriores aceptan registros tipo *L_msType* con el campo *instrument* igual a "SWAP" **o** "NDF" **y** con el campo *buyer_dealer* comenzando con "AAAL" **o** con el campo *seller_dealer* comenzando con "BBBL".

Si no se dan condiciones para un determinado tipo de mensaje, todos los mensajes de tal tipo serán entregados a la aplicación.

### 7.1.2. Método *FinishMessageProcess*

VARIANT CdxapiCtrl::**FinishMessageProcess** (BSTR *message_type*, BSTR *message_id*)
int CdcApi::**FinishMessageProcess** (LPCTSTR *message_type*, UINT *message_num*)

Declaración para 32 bits:
int CdcApi::**FinishMessageProcess** (LPCTSTR *message_type*, ULONG *message_num*)

**Valor de retorno**
Retorna 0 si no hay error, en caso contrario retorna el código de error.

**Parámetros**
*message_type:* Indica el tipo de mensaje. Usar el mismo valor recibido en el evento *MessageReceived()*.

*message_num:* Indica el número de mensaje. Úsese el mismo valor recibido en el evento *OnMessageReceived()*. Este número único se usa para identificar el registro y está directamente asociado con el mensaje generado por el equipo DataServer.
*message_id:* Es el identificador del mensaje. Úsese el mismo valor recibido en el evento *OnMessageReceived()*. Este identificador único define el registro y está directamente asociado con el mensaje generado por el equipo DataServer.

**Descripción**
Este método libera el mensaje previamente recibido y almacenado en la cola de proceso y lo marca como procesado por la aplicación. Todos los mensajes que no han sido marcados como procesados por este método serán retransmitidos en caso de que el DAPI pierda/cierre la conexión establecida con el servidor y restablezca una nueva conexión.

### 7.1.3. Método *Login*

VARIANT CdxapiCtrl::**Login** (BSTR *username*, BSTR *password*)
int CdcApi::**Login** (LPCTSTR *username*, LPCTSTR *password*)

**Valor de retorno**
Si el proceso se ha iniciado correctamente retorna 0 cuando *Login* se ejecuta por vez primera, caso contrario retorna DAPI_ERR_ALREADY_INITIALIZED.

Los posibles valores de error son:
DAPI_ERR_NO_DS_DIRECTIVE, DAPI_ERR_CONFIG_NOT_FOUND
DAPI_ERR_CONFIG_BAD_DATA, DAPI_ERR_RC_NOT_AUTHORIZED_USER
DAPI_ERR_RC_USERNAME_DONOT_EXIST, DAPI_ERR_RC_ILLEGAL_PASSWORD
DAPI_ERR_RC_NO_DMACHINES_TRY_LATER
DAPI_ERR_RC_DUPLICATED_MACHINE_CODE
DAPI_ERR_RC_MACHINE_NOT_REGISTERED
DAPI_ERR_RC_ILLEGAL_BRANCH_PASSWORD

**Parámetros**
*Username:* Indica el nombre de usuario que el DAPI usará para conectarse con el servidor.
*Password:* Indica la contraseña que el DAPI usará para conectarse con el servidor.

**Descripción**
El método *Login* inicia una operación asíncrona de conexión del DAPI con el Sistema Datatec. Para esto el DAPI inicia algunas tareas concurrentes para monitorear la conexión y controlar el proceso de los mensajes. Las siguientes llamadas a este método solamente actualizarán el nombre de usuario o contraseña que el DAPI usa para autenticarse con el equipo servidor.

El nombre de usuario y contraseña son inicialmente definidos en el equipo DataManager Cliente de la misma manera que se hace para un Usuario Primario. Además de los procesos normales de configuración que realiza el operador en el equipo Administrador será necesario conceder a este usuario el privilegio para recibir el/los suministro/s de datos requeridos.

La dirección IP del equipo servidor del sistema y los restantes parámetros requeridos para establecer la conexión son tomados del archivo *dapi.cfg*. Este archivo debe configurarse de acuerdo con el esquema de red del sistema y debe ser colocado en el directorio donde se ejecuta la aplicación.

Cuando se completa el proceso de conexión, el DAPI ejecuta el evento *OnConnection()* indicando el resultado del proceso. Si el proceso falla o se pierde la conexión el DAPI reintentará restablecer la conexión cada 3 segundos. Este proceso continúa indefinidamente hasta que se llame el método *Logout()* o hasta que se restablezca finalmente la conexión. Luego de cuatro intentos fallidos al mismo servidor, el DAPI usa el siguiente servidor definido en el archivo de configuración *dapi.cfg*. Una vez completado el intento con todos los servidores, reintenta nuevamente a partir del primero, y así sucesivamente.

### 7.1.4. Método *Logout*

VARIANT CdxapiCtrl::**Logout** (void)
Int CdcApi::**Logout** (void)

**Valor de retorno**
Retorna 0 si el proceso de desconexión finaliza satisfactoriamente. Si el método *Login()* no fue llamado previamente retorna:
DAPI_ERR_NOT_INITIALIZED_YET.

**Parámetros**
Este método no usa ningún parámetro.
**Descripción**
El método *Logout()* finaliza una conexión establecida entre el DAPI y el DataServer por el método *Login().* Este método también termina los intentos de reconexión y todos los *threads* que *Login()* haya iniciado.

Al llamar el método *Logout()* los mensajes recibidos y no terminados de procesar con el método *FinishMessageProcess()* serán retransmitidos al DAPI en la siguiente reconexión.

### 7.1.5. Método *GetNextFieldName (ActiveX)*

VARIANT CdxapiCtrl::**GetNextFieldName** (BSTR *field_name*)

**Valor de retorno**
Retorna el nombre del siguiente campo, si no existe u ocurre algún error retorna “”.

**Parámetros**
*field_name:* Es el nombre del campo anterior al deseado. El método busca el nombre del siguiente campo en el mensaje. Un valor nulo (“”) se puede usar para obtener el nombre del primer campo del mensaje.

**Descripción**
Permite obtener el nombre del siguiente campo en un mensaje. Se puede usar este método dentro de un bucle de repetición para obtener todos los campos que pertenecen al mensaje. En este caso solicitar el primer nombre de campo con una cadena de caracteres vacía (“”).

El mensaje sobre el que este método aplica es el último mensaje obtenido con el método *TakeMessage()*. Por favor notar que por ahora solo se pueden usar las versiones en español de los nombres de los campos.

### 7.1.6. Método *GetNextField (Libreria)*

int CdcApi::**GetNextField** (char * *field_name,*char **data_copy*)

**Valor de retorno**
Retorna el tipo del campo referenciado, en caso de error retorna un valor de 0.

| Code | Field Type |
|---|---|
| 1 | FLD_SHORT |
| 2 | FLD_UNSIGNED_SHORT |
| 3 | FLD_IMSIGNED_LONG |
| 4 | FLD_NUM_ASCII_INT |
| 5 | FLD_NUM_ASCII_DOUBLE |
| 6 | FLD_ALPHA |
| 7 | FLD_INT |

**Parámetros**
*field_name:* Indica el nombre del campo anterior al deseado. Un valor nulo “” se puede usar para obtener el primer campo del mensaje. El nombre del campo encontrado es colocado en *field_name*.

*data_copy:* El contenido del campo referenciado será copiado a esta área.

**Descripción**
Permite obtener el siguiente campo de un mensaje, retorna el nombre, tipo y contenido. Se puede usar este método dentro de una estructura de repetición para obtener todos los campos que pertenecen al mensaje, en este caso iniciar *field_name* con una cadena de caracteres vacía. Asegurarse que tanto *field_name* como *data_copy* tengan capacidad suficiente para almacenar el nuevo valor obtenido.

El mensaje sobre el que este método aplica es el último mensaje obtenido con el método *TakeMessage()*. Por favor notar que por ahora solo se pueden usar las versiones en español de los nombres de los campos.

### 7.1.7. Método *GetField (ActiveX)*

BSTR CdxapiCtrl::**GetField** (BSTR field_name)

**Valor de retorno**
Retorna el valor del campo solicitado como cadena básica. Retorna una cadena nula en caso de error.

**Parámetros**
*field_name:* Indica el nombre del campo cuyo valor se desea obtener.

**Descripción**
Este método permite obtener el valor de un campo expresado como cadena básica. En caso de error el evento *OnError()* será llamado y se retornará una cadena nula.

El mensaje sobre el que este método aplica es el último mensaje obtenido con el método *TakeMessage()*. Por favor notar que por ahora solo se pueden usar las versiones en español de los nombres de los campos.

### 7.1.8. Método *GetField (Libreria)*

int CdcApi::**GetField** (LPCTSTR *field_name,* char **data_copy*)

**Valor de retorno**
Retorna el tipo dato del campo solicitado. En caso de error retorna un valor de 0.

| Code | Field Type |
|---|---|
| 1 | FLD_SHORT |
| 2 | FLD_UNSIGNED_SHORT |
| 3 | FLD_IMSIGNED_LONG |
| 4 | FLD_NUM_ASCII_INT |
| 5 | FLD_NUM_ASCII_DOUBLE |
| 6 | FLD_ALPHA |
| 7 | FLD_INT |

**Parámetros**
*field_name:* Indica el nombre del campo cuyo valor va a ser obtenido.
*data_copy:* Dirección de memoria donde se sacará una copia del valor del campo solicitado.

**Descripción**
Este método permite recoger el contenido del campo. El dato del campo es copiado en bytes a *data_copy*.

El mensaje sobre el que este método aplica es el último mensaje obtenido con el método *TakeMessage()*. Por favor notar que por ahora solo se pueden usar las versiones en español de los nombres de los campos.

### 7.1.9. Método *GetFieldBSTR (Libreria)*

int CdcApi::**GetFieldBSTR** (LPCTSTR *field_name,*char **data_copy*)

**Valor de retorno**
Retorna la longitud del campo solicitado. En caso de error retorna 0.

**Parámetros**
*field_name:* Indica el nombre del campo cuyo valor va a ser obtenido.

*data_copy:* Dirección de memoria donde se pondrá una copia del campo solicitado.

**Descripción**
Este método permite recoger el contenido del campo como cadena de caracteres. El dato del campo es copiado en bytes a *data_copy* y finaliza con un byte nulo.

El mensaje sobre el que este método aplica es el último mensaje obtenido con el método *TakeMessage()*. Por favor notar que por ahora solo se pueden usar las versiones en español de los nombres de los campos.

### *7.1.10. Método GetFieldDATE (Libreria)*

int CdcApi::**GetFieldDATE** (LPCTSTR *field_name*,char **data_copy*)

**Valor de retorno**
La longitud de la cadena resultante, o sea 8. En caso de error retorna 0. Los campos fecha de los mensajes siempre tienen el formato AAAAMMDD.

**Parámetros**
*field_name:* Indica el nombre del campo cuyo valor va a ser obtenido.

*data_copy:* Dirección de memoria donde se pondrá una copia del campo solicitado.

**Descripción**
Este método permite recoger el contenido del campo en formato tipo fecha. El dato del campo es copiado como una secuencia de bytes que contienen la fecha en 8 bytes.

El mensaje sobre el que este método aplica es el último mensaje obtenido con el método *TakeMessage()*. Por favor notar que por ahora solo se pueden usar las versiones en español de los nombres de los campos.

### 7.1.11. Método *GetFieldLONG (Libreria)*

LONG CdcApi :: **GetFieldLONG**(LPCTSTR *field_name*)

**Valor de retorno**
Retorna el valor del campo convertido a *signed long*. Si hay error se llamará a *OnError()* y retorna un valor de 0.

**Parámetros**
*field_name:* Indica el nombre del campo cuyo valor va a ser obtenido.

**Descripción**
Este método permite recoger el contenido del campo como un entero largo con signo (*signed long)*. En caso de error se retorna un valor de 0.

El mensaje sobre el que este método aplica es el último mensaje obtenido con el método *TakeMessage()*. Por favor notar que por ahora solo se pueden usar las versiones en español de los nombres de los campos.

### 7.1.12. Método InitMessage

VARIANT CdxapiCtrl::**InitMessage** (BSTR *message_type*)
int CdcApi::**InitMessage** (LPCTSTR *message_type*)

**Valor de retorno**
Retorna 0 si no hay error. Caso contrario alguno de los siguientes errores puede ser retornado:
DAPI_ERR_NOT_INITIALIZED_YET
DAPI_ERR_ILLEGAL_RECORD_TYPE_SPECIFIED

**Parámetros**
*message_type:* Indica el tipo de mensaje que se va a construir. Si se da un nombre nulo (como “”), el tipo y contenido del último mensaje tomado con *TakeMessage()* será usado como base para el nuevo mensaje.

**Descripción**
Con este método se puede iniciar la construcción de un mensaje que va a ser enviado al DataServer. La aplicación debe llamar este método y luego colocar los valores en todos los campos requeridos. Si se suministra el argumento “” al campo *message_type* solamente será necesario colocar valor en aquellos campos que difieren en contenido al mensaje previamente seleccionado con *TakeMessage()*. Las siguientes llamadas a los métodos *SetField()* aplicarán sobre este nuevo mensaje. De ser necesario, se especifican los destinatarios del mensaje con el método *SendMessageTo()*. Finalmente llámese al método *SendMessage()* para enviar el mensaje al DataServer.

### 7.1.13. Método *ProcessMyEvents (Librería)*

void CdcApi::ProcessMyEvents ()

**Descripción**
Llame a este método para usar el DAPI como librería con protección de multithread. Cuando la aplicación ejecuta este método, el DAPI llama a los métodos que atienden a todos los eventos ocurridos hasta el momento, además suspende la atención de los eventos futuros hasta una siguiente llamada de *ProcessMyEvents()*.

Los eventos de error del DAPI generados por la aplicación (cuando se llama a los métodos de la clase del DAPI) no están restringidos por cuanto estos no tienen conflictos de multithread con la aplicación. Únicamente los eventos del DAPI tales como llegada de mensaje, pérdida de conexión, etc. están restringidos para evitar conflictos. No llamar a esta función si la aplicación usa multithread de forma independiente.

### 7.1.14. Método SendMessage

VARIANT CdxapiCtrl:: **SendMessage** (BSTR *message_type*)
int CdcApi::**SendMessage** (LPCTSTR *message_type*)

**Valor de retorno**
Retorna 0 si no hay error. En caso contrario retorna alguno de los siguientes valores:
DAPI_ERR_NOT_INITIALIZED_YET
DAPI_ERR_ILLEGAL_RECORD_TYPE_SPECIFIED

**Parámetros**
*message_type:* Indica el tipo del mensaje que será enviado. Si no se indica el nombre (“”), el último mensaje tomado con *TakeMessage()* será enviado.

**Descripción**
Este método envía el mensaje previamente construido al sistema. Para construir un mensaje llámese primeramente al método *InitMessage()*, luego colóquese los valores en los campos del mensaje usando los métodos *SetField()*.De ser necesario, se especifican los destinatarios del mensaje con el método *SendMessageTo()*. Finalmente ejecútese este método *SendMessage()*. Si no se indica el nombre del tipo de registro (un valor de “”), el mismo tipo de mensaje tomado con el método *TakeMessage()* será usado.

### 7.1.15. Método *SendMessageTo*

VARIANT CdxapiCtrl::**SendMessageTo** (BSTR *address_type*, BSTR *address_data*)
int CdcApi::**SendMessageTo** (LPCTSTR *address_type*, LPCTSTR *address_data*)

**Valor de retorno**
Retorna 0 si no hay errores, caso contrario retorna uno de los siguientes valores:
DAPI_ERR_NOT_INITIALIZED_YET
DAPI_ERR_ILLEGAL_ADDRESS_TYPE_SPECIFIED.

**Parámetros**
*address_type:* Indica el tipo de dirección al cual se desea enviar el mensaje. Los argumentos válidos que se pueden suministrar en este campo son:

| Valor | Significado |
|---|---|
| “PU” | Usuario Primario |
| “SU” | Sucursal |
| “IN” | Institución |
| “XP” | Excepto a este Usuario Primario |
| “XS” | Excepto a esta Sucursal |
| “GR” | Nombre de un grupo interno |

*address_data:* Indica la dirección a la cual se desea enviar el mensaje. Las direcciones del sistema tienen diferente longitud dependiendo del tipo.; Normalmente corresponden a los códigos asignados a los usuarios, sucursales o instituciones por el equipo Administrador del sistema.

| *Address_type* | Valor |
|---|---|
| “PU” | Nombre del usuario PU (4 caracteres) |
| “SU” | Nombre de la sucursal (8 caracteres) |
| “IN” | Nombre de la institución (4 caracteres) |
| “XP” | Nombre del usuario PU que no es destinatario (4 caracteres) |
| “XS” | Nombre de la sucursal que no es destinataria (8 caracteres) |
| “GR” | Nombre de un grupo interno de destinatarios (8 caracteres) |

**Descripción**
Este método indica la dirección a la cual se enviará el siguiente mensaje. Se puede llamar varias veces este método para indicar varias direcciones de destino. Este método debe ser llamado antes de usar el método *SendMessage()* que se encarga del despacho. Si no se establece los destinatarios del mensaje se entiende que el mensaje será enviado al DataManagerServer. Por ejemplo:

```
m_dapi.TakeMessage (L_MessType, mess_num );  // Indica el mensaje previamente recibido
m_dapi.InitMessage (“”);  // Inicia el mensaje de envío igual a mess_num

m_dapi.SetFieldBSTR(L_MessType, "bandera", "S"); // cambia algún campo

// Indica como destinatarios todos los equipos de
// la sucursal
m_dapi.SendMessageTo (“SU”, Axdxapi1.GetField("sucursal")
// excepto a este usuario
m_dapi.SendMessageTo (“XP”, Axdxapi1.GetField("codigo_primaryuser")

// Envía el mensaje
m_dapi.SendMessage(L_MessType);
```

### 7.1.16. Método *SetField (ActiveX) y SetFieldBSTR (Librería)*

VARIANT CdxapiCtrl::**SetField** (BSTR *message_type*, BSTR *field_name,* BSTR *new_value*)
int CdcApi::**SetFieldBSTR** (LPCTSTR *message_type*, LPCTSTR *field_name*, LPCTSTR *new_value*)

**Valor de retorno**
Retorna 0 si no hay errores. En caso contrario retorna alguno de los siguientes valores:
DAPI_ERR_NOT_INITIALIZED_YET
DAPI_ERR_ILLEGAL_RECORD_TYPE_SPECIFIED
DAPI_ERR_ILLEGAL_FIELD_NAME_SPECIFIED

**Parámetros**
*message_type:* Indica el tipo de mensaje al que pertenece el campo de nombre *field_name*.
*field_name:* Indica el nombre de campo en el que se va a colocar el valor.
*new_value:* Indica el nuevo valor que será asignado al campo.

**Descripción**
Este método asigna el contenido de *new_value* al campo de nombre *field_name* y que pertenece al último mensaje del tipo *message_type* inicializado con el método *InitMessage()*. Si *message_type* es nulo (“”), se usa el último mensaje tomado con el método *TakeMessage()*.

Si ocurre algún error al convertir el valor que se da al tipo del campo destino se ejecuta el evento *OnError()* detallando el error ocurrido.

### 7.1.17. Método *SetFieldDATE (Libreria)*

int CdcApi::**SetFieldDATE** (LPCTSTR *message_type*, LPCTSTR *field_name*, DATE *new_value*)

**Valor de retorno**
Retorna 0 si no hay errores. En caso contrario retorna alguno de los siguientes valores:
DAPI_ERR_NOT_INITIALIZED_YET
DAPI_ERR_ILLEGAL_RECORD_TYPE_SPECIFIED
DAPI_ERR_ILLEGAL_FIELD_NAME_SPECIFIED
**Parámetros**
*message_type:* Indica el tipo de mensaje al que pertenece el campo de nombre *field_name*.
*field_name:* Indica el nombre del campo en el que se va a colocar el valor.
*new_value:* Indica el nuevo valor que será asignado al campo.

**Descripción**
Este método asigna valor de fecha *new_value* (formato AAAAMMDD) al campo *field_name* y que pertenece al último mensaje del tipo *message_type* que se haya inicializado con el método *InitMessage()*. Solo el año, mes y día son tomados en cuenta. Si *message_type* es nulo (“”), se usa el último mensaje tomado con el método *TakeMessage()*.

Si ocurre algún error al convertir el valor que se da al tipo del campo destino se ejecuta el evento *OnError()* detallando el error ocurrido.

### 7.1.18. Método *SetFieldLONG (Libreria)*

int CdcApi::**SetFieldLONG** (LPCTSTR *message_type*, LPCTSTR *field_name*, LONG *new_value*)

**Valor de retorno**
Retorna 0 si no hay errores. En caso contrario retorna alguno de los siguientes valores:
DAPI_ERR_NOT_INITIALIZED_YET
DAPI_ERR_ILLEGAL_RECORD_TYPE_SPECIFIED
DAPI_ERR_ILLEGAL_FIELD_NAME_SPECIFIED

**Parámetros**
*message_type:* Indica el tipo de mensaje al que pertenece el campo de nombre *field_name*.
*field_name:* Indica el nombre de campo en el que se va a colocar el valor.
*new_value:* Indica el nuevo valor que será asignado al campo.

**Descripción**
Este método asigna valor *long* (entero largo con signo) *new_value* al campo cuyo nombre es *field_name* y que pertenece al último mensaje del tipo *message_type* que se haya inicializado con el método *InitMessage()*. Si *message_type* es nulo (“”), se usa el último mensaje tomado con el método *TakeMessage()*. Si ocurre algún error al convertir el valor que se da al tipo del campo destino se ejecuta el evento *OnError()* detallando el error ocurrido.

### 7.1.19. Método TakeMessage

VARIANT CdxapiCtrl::**TakeMessage** (BSTR *message_type*, BSTR *message_id*)
int CdcApi::**TakeMessage** (LPCTSTR *message_type*, ULONG *message_num*);

Declaración para 32 bits:
int CdcApi::**TakeMessage** (LPCTSTR *message_type*, unsigned int *message_num*);

**Valor de retorno**
Retorna 0 si no hay errores. En caso contrario retorna alguno de los siguientes valores:
DAPI_ERR_NOT_INITIALIZED_YET
DAPI_ERR_ILLEGAL_RECORD_TYPE_SPECIFIED
DAPI_ERR_RECORD_NOT_FOUND

**Parámetros**
*message_type:* Indica el tipo de mensaje. Úsese el mismo valor recibido en el evento *OnMessageReceived()*.
*message_num:* Indica el número de mensaje. Úsese el mismo valor recibido en el evento *OnMessageReceived()*. Este número único se usa para identificar el mensaje y está directamente asociado con el mensaje generado por el equipo DataServer.

*message_id:* Es el identificador del mensaje. Úsese el mismo valor recibido en el evento *OnMessageReceived()*. Este identificador único define el registro y está directamente asociado con el mensaje generado por el equipo DataServer.

**Descripción**
Este método inicializa el proceso de obtención de campos de un mensaje recibido. Las siguientes llamadas a los métodos *GetField()* con el mismo valor de *message_type* usarán el mensaje referenciado por este método.

Los mensajes del sistema son enviados al DAPI, éste asigna un número o identificador a cada mensaje y lo pasa a la aplicación por medio del evento *OnMessageReceived()*. El mismo número o identificador debe ser usado por este método para tomar el mensaje que el evento ha anunciado.

### 7.1.20. Método IsDefined

VARIANT CdxapiCtrl:: **IsDefined** (LPCTSTR *field_name*)
Int CdcApi::**IsDefined** (LPCTSTR *field_name*)

**Valor de retorno**
Retorna 0 si no existe el campo. En caso contrario retorna 1.

**Parámetros**
*field_name:* Indica el nombre de campo del cual se requiere confirmar su existencia.

**Descripción**
Este método busca en el mensaje por el nombre del campo *field_name.*

#### *7.1.21.* Método GetCompilationType

VARIANT CdxapiCtrl:: **GetCompilationType** ()
int CdcApi::**GetCompilationType** (LPCTSTR *compilation_type*)

**Valor de retorno**
Retorna siempre 1.

**Parámetros**
*compilation_type:* Se almacena la plataforma de compilación del Dapi.

**Descripción**
Este método asigna una cadena a *compilation_type;* en caso que la plataforma de compilación del Dapi es de 32 bits, asigna la cadena "x86". Caso contrario "x64".

#### 7.1.22. Método GetVersion

VARIANT CdxapiCtrl:: **GetVersion** ()
int CdcApi:: **GetVersion** (LPCTSTR *version*)

**Valor de retorno**
Retorna siempre 1.
**Parámetros**
*version:* Se almacena la versión del dapi.

**Descripción**
Este método asigna una cadena a *version;* con el número de la versión del dapi generado.

## 7.2. Eventos del DAPI

Los eventos son códigos predefinidos a los cuales el DAPI llama cuando ocurren las condiciones requeridas para activarlos. La aplicación debe contener las instrucciones necesarias para realizar el proceso pertinente para cada evento. El DAPI llama a un evento para indicar a la aplicación que algo ha sucedido y la aplicación puede o no realizar un proceso de acuerdo con sus propios requerimientos.

### 7.2.1. Eventos en el DAPI como ActiveX Control (OCX)

En Windows y con el DAPI como ActiveX Control la llamada a un evento se realiza dentro del esquema de proceso de los mensajes de Windows; de esta manera todos los eventos se ejecutan en secuencia y sin interferir unos con otros o con la ejecución del código de la aplicación.

En aplicaciones VB se debe usar el método *Invoke* para ejecutar la función que atiende a un evento del DAPI y que pasa información a los controles de la ventana. Esto es necesario por cuanto en VB el thread que atiende a una ventana de la aplicación es independiente del proceso de los eventos asincrónicos del DAPI.

### 7.2.2. Eventos en el DAPI como librería en C++

Para usar el DAPI como librería, existen dos formas de interactuar con el mismo, la primera es sin control de *multithread* y la segunda es controlando el ambiente *multithread*.

La forma más fácil de interactuar con el DAPI es sin el control de multithread, la aplicación debe dar tiempo de ejecución al DAPI llamando periódicamente (cada cierto intervalo de tiempo) al método *ProcessMyEvents* (Véase como ejemplo el archivo drvapitest.cpp). Esta forma de interactuar implica

cierta latencia o demora en el proceso de los mensajes, esta demora está en relación directa con el intervalo de tiempo entre cada llamada al método *ProcessMyEvents*. La llamada excesiva a este método consume innecesariamente el tiempo de CPU.

Cuando se controla el ambiente multithread se deben resolver mediante *mutex* o semáforos la ejecución de eventos y de procesos de envío de mensajes al sistema. Por ejemplo, si se usa un arreglo de bytes para ensamblar un mensaje que se va a enviar y el mismo arreglo se lo usa para atender a un mensaje recibido, estamos en un caso que tendrá problemas por ejecución concurrente, hay que evitar el conflicto creando otro arreglo, o excluyendo el acceso simultáneo al arreglo mediante un mutex. Recuerde que NUNCA debe ejecutar *ProcessMyEvents()*, porque de hacerlo, el DAPI activa su propio sistema de sincronización y los siguientes eventos que ocurran solo serán ejecutados en una próxima llamada a *ProcessMyEvents().*

### 7.2.3. Evento OnMesageReceived

CdxapiCtrl::**OnMessageReceived** (BSTR *message_type*, BSTR *message_id*)

CdcApi::**OnMessageReceived** (LPCTSTR *message_type*, ULONG *message_num*)

Declaración para 64 bits:
CdcApi::**OnMessageReceived** (LPCTSTR *message_type*, unsigned int *message_num*)

**Parámetros**

*message_type:* Recibe el tipo de mensaje. Este valor debe ser usado para llamar a *TakeMessage()* a fin de recuperar el contenido de los campos.

*message_num:* Es el número asignado por el DAPI a este mensaje. Este valor debe al llamar a *TakeMessage()* para poder recuperar el contenido de los campos.

*message_id:* Es el identificador asignado por el DAPI a este mensaje. Este valor debe al llamar a *TakeMessage()* para poder recuperar el contenido de los campos.

**Descripción**
Este evento ocurre cuando el DAPI recibe un mensaje del servidor del sistema.

La aplicación puede usar el método *TakeMessage()* para ubicar el mensaje, y luego los métodos *GetField()* para tomar el contenido de los campos. El número o identificador también deberán usarse en el método *FinishMessageProcess()*.

El DAPI mantiene los mensajes recibidos hasta que se llame al método *FinishMessageProcess()* o hasta que se pierda la conexión. Si un mensaje ha sido recibido y ocurre una pérdida de conexión sin que se llame al método *FinishMessageProcess()* cuando la conexión se restablezca el mensaje será pasado nuevamente a la aplicación por medio de este evento.

### 7.2.4. Evento OnConnection

CdxapiCtrl::**OnConnection** (int *result_code)*

CdcApi::**OnConnection** (int *result_code)*

**Parámetros**

*result_code:* Recibe el código de conexión resultante. Un valor de 0 significa una conexión establecida satisfactoriamente. En otro caso es el código ASCII de la letra recibida del Data Server.

**Descripción**
El proceso de conexión se mantiene automáticamente por el DAPI una vez que se ha llamado al método *Login()* y al final de cada intento el DAPI ejecuta este evento para indicar el resultado.

### 7.2.5. Evento OnDisconnection

CdxapiCtrl::**OnDisconnection** (int reason_code, BSTR *reason_text*)

CdcApi::**OnDisconnection** (int reason_code, LPCTSTR *reason_text*)

**Parámetros**

reason_code: Recibe el estado actual de la conexión.

| Code | Field Type |
|---|---|
| 1 | DCON_ST_OPENING |
| 2 | DCON_ST_OPENCHK |
| 4 | DCON_ST_NORMCON |
| 5 | DCON_ST_CLOSE |
| 6 | DCON_ST_TERMIN |

*reason_text:* Recibe un texto adjunto detallando “de ser posible” la causa de la desconexión.

**Descripción**
El proceso de conexión es mantenido automática e indefinidamente por el DAPI. Si ocurre alguna pérdida de conexión el DAPI llama a este evento.

En caso de pérdida inesperada de conexión los mensajes recibidos y no terminados de procesar con el método *FinishMessageProcess()* NO son retransmitidos y NO generan el evento *OnMessageReceived()* en la siguiente reconexión, puesto que este tipo de desconexión no borra los mensajes recibidos que no hayan sido procesados.

### 7.2.6. Evento OnError

CdxapiCtrl::**OnError** (int *error_code*, BSTR *error_text*, int *next_action_code*, BSTR *next_action_data*)

CdcApi::**OnError** (int *error_code*, LPCTSTR *error_text*, int *next_action_code*, int *next_action_data*)

**Parámetros**

*error_code:* Recibe el código numérico del error.

*error_text:* Un texto explicativo de la causa del error. Si el caso lo requiere se incluye más detalle dentro de este texto para hacerlo más preciso.

*next_action_code:* Un valor sugestivo del siguiente paso procedente para la aplicación. Los posibles valores se indican a continuación:

| Valor | Definición | Significado |
|---|---|---|
| 0 | DAPI_NEXT_ACTION_CONTINUE | El DAPI puede continuar operando. |
| 1 | DAPI_NEXT_ACTION_RECONNECT | Puede intentar nuevamente la conexión corrigiendo el nombre de usuario o palabra secreta. |
| 999 | DAPI_NEXT_ACTION_EXIT | El DAPI no puede seguir operando debido a este error. |

*next_action_data:* Datos extra de ayuda para la siguiente acción. No se usan en esta versión.

**Descripción**
El DAPI activa este evento cuando algún proceso interno encuentra alguna condición anormal o cuando algún método no se ha logrado completar satisfactoriamente. La aplicación puede usar este evento para registrar y verificar la ejecución adecuada del DAPI.

### 7.2.7. Evento OnStatusChanged

CdxapiCtrl::**OnStatusChanged** (int *status_code*, LPCTSTR *status_text*)

CdcApi::**OnStatusChanged** (int *status_code*, LPCTSTR *status_text*)

**Parámetros**

*status_code:* Recibe el estado actual de la conexión. Los posibles valores son los siguientes

| Valor | Definición | Significado |
|---|---|---|
| 7 | TCP_ERROR_CANT_CONNECT | La conexión TCP/IP ha fallado. Otro intento será realizado, y en su momento, otro servidor será tomado para el intento. |
| 21 | TCP_STATUS_OPEN_CONNECTION | Se ha establecido una conexión TCP/IP. |
| 20 | TCP_STATUS_CLOSED_CONNECTION | Se ha perdido la conexión TCP/IP. |
| 27 | TCP_SHUTTING_DOWN | Se ha requerido que la conexión se cierre. Y se inicia el proceso de cierre de la conexión, se envían últimos datos pendientes. |

*status_text:* Se recibe el texto del estado que puede tener los siguientes valores:

| Definición | Valor | Significado |
|---|---|---|
| TCP_STATUS_OPEN_CONNECTION | "Initiating log translation" | Un archivo de log está siendo retroalimentado y simula una conexión |
| TCP_STATUS_CLOSED_CONNECTION | "Finalizing log translation" | La retroalimentación de un log se ha activado y se está simulando una desconexión |
| TCP_STATUS_OPEN_CONNECTION | "Connected" | Se ha establecido una conexión TCP/IP. |
| TCP_STATUS_CLOSED_CONNECTION | "Disconnected" | Se ha perdido la conexión TCP/IP. |
| TCP_ERROR_CANT_CONNECT | "Can't connect" | La conexión TCP/IP ha fallado. Otro intento será realizado, y en su momento, otro servidor será tomado para el intento. |
| TCP_SHUTTING_DOWN | "Attempts to establish a connection and processing messages have been suspended" | Se ha requerido que la conexión se cierre. Y se inicia el proceso de cierre |

**Descripción**
Este evento se ejecuta cuando la conexión se establece, se cierra, se pierde, o cuando finaliza la ejecución del DAPI con *Logout()*. Este evento está implementado para propósitos de verificación.

#### 7.2.8. Evento OnDataReceived

CdxapiCtrl::**OnDataReceived** (BSTR *source_id*, BSTR *message_name*, BSTR *message_data*)

CdcApi::**OnDataReceived** (LPCTSTR *source_id*, LPCSTR *message_name*, LPCTSTR *message_data*)

**Parámetros**
*source_id:* Recibe una cadena que define la conexión. Este campo se implementó para el caso de conexiones múltiples.

*message_name:* Recibe el tipo de mensaje.

*message_data:* El contenido del mensaje se convierte a una cadena y es pasado al evento. Este evento ha sido creado especialmente para propósitos de verificación.

**Descripción**
El evento *OnDataReceived()* recorta el mensaje recibido hasta el primer byte nulo y lo pasa en *message_data*.

# 8. Códigos de ejemplo del DAPI

Con el kit de desarrollo se proveen dos códigos fuente que se dan como ejemplo del uso del DAPI:

1. **VbdxApiTest**. Aplicación de ejemplo en Visual Basic usando el ActiveX control del DAPI.
2. **Drvapitest**. Aplicación de ejemplo en C++ usando la librería del DAPI para Linux (32/64 bits).

Los ejemplos están codificados para demostrar el uso del DAPI en las siguientes acciones:

- Inicio de conexión del DAPI usando el método *Login().*
- Activación del evento On*MessageReceived()* cuando el DAPI recibe un mensaje del sistema.
- Selección de un mensaje para obtener su contenido usando el método *TakeMessage().*
- Obtención de cualquier campo del mensaje con el método *GetField().*
- Activación del evento *OnConnection()* cuando el DAPI se conecta al sistema.
- Activación del evento *OnDisconnection()* cuando el DAPI se desconecta del sistema.
- Manejo de excepciones cuando se dispara el evento *OnError().*

## 8.1. Ejemplo de cómo procesar un mensaje en Visual Basic

Para procesar un mensaje usualmente debe colocarse el código respectivo en el evento *OnMessageReceived()*, identificar el registro con *TakeMessage()*, extraer los campos con las *GetField()*, hacer conjuntamente el proceso deseado y finalmente llamar al método *FinishMessageProcess()*.

```
Public Sub OnOnMessageReceived(ByVal sender As Object, ByVal e _
        As AxdxapiLib._DdxapiEvents_OnMessageReceivedEvent) _
    Handles Axdxapi1.OnMessageReceived
  If e.message_type = mensaje_tabla_moneda Then
    Axdxapi1.TakeMessage(mensaje_tabla_moneda, e.message_id)
    mercado_postura = Axdxapi1.GetField("mercado")
    If Axdxapi1.GetField("o_d") = "O" Then
      ….
      // obtener un subcampo a partir del código
      nombre_cliente_vendedor = Axdxapi1.GetField("codigo_postura_hoy.nombre")
      ….
    End If
  End If
  Axdxapi1.FinishMessageProcess (e.message_type, e.message_id)
End Sub
```

## 8.2. Ejemplo de cómo enviar un mensaje en C++

Existen dos formas de construir un mensaje y enviarlo, la primera es construyendo todo el mensaje y la segunda es usando un mensaje que ha sido previamente recibido al que se desea cambiar algunos campos. A continuación, se presenta un ejemplo del primer caso programado en C++:

```
  m_dapi.InitMessage (L_MessType);  // Inicia un mensaje en blanco
m_dapi.SetFieldBSTR(L_MessType, "trade_identification", m_trade_value);

m_dapi.SetFieldBSTR(L_MessType, "instance_trade", m_instance_value);
m_dapi.SetFieldBSTR(L_MessType, "bandera", "S"); // coloca valores requeridos
m_dapi.SetFieldBSTR(L_MessType, "flag_side_changed", "A");
  //; B – The BUYER STP STATUS IS specified
  //; S – The SELLER STP STATUS IS specified
  //; A – Both BUYER and SELLER are specified
m_dapi.SetFieldBSTR(L_MessType, "buyer_side_status_error", "reason");
  //; Texto con la razón para el comprador
m_dapi.SetFieldBSTR (L_MessType, "seller_side_status_error", "reason");
  //; Texto con la razón para el vendedor
```

```
m_dapi.SetFieldBSTR(L_MessType, "buy_stp_status", "Status value");
  //; Estado de la transacción del comprador
m_dapi.SetFieldBSTR(L_MessType, "sell_stp_status", "Status value");
  //; Estado de la transacción del vendedor
m_dapi.SendMessage(L_MessType); // Envía el mensaje
```

Un ejemplo del segundo caso es el siguiente (código en C++):

```
m_dapi.TakeMessage (L_MessType, mess_num ); // Selección de un mensaje recibido
m_dapi.InitMessage (“”);  // Inicia un mensaje igual al mensaje referenciado mess_num
m_dapi.SetFieldBSTR(L_MessType, "bandera", "S"); // cambio del campo requerido
m_dapi.SendMessage(L_MessType);
```

## 8.3. Diagrama de flujo ejemplo para recoger información de una transacción para aplicaciones back-office

Es importante identificar cuidadosamente la información que el sistema back-office necesita extraer del sistema. En esta sección se esboza un esquema sencillo para procesar adecuadamente la información recogida por el DAPI y se presenta un diagrama de flujo sugerido.

El mensaje de mayor utilidad que recibe el DAPI del sistema es el *Orden_Transaccion* que es enviado cada vez que se ejecuta una acción que afecta la ventana de mercado. Ejemplos de tales acciones incluyen: ingreso, modificación, retiro o agresión de demandas/ofertas. Para fines prácticos, el diagrama de flujo siguiente se enfoca únicamente en los eventos transaccionales.

1. Cuando llega un mensaje *Orden_Transaccion* leer el campo *clase_mensaje* y dependiendo de su valor proceder por la rama correspondiente.

2. En todos los casos se debe identificar la transacción con un número único que se forma de la unión de los siguientes campos: *mercado*+*fecha*+*hora*+codigo_*postura*+c*odigo_postura_1*. Por ejemplo, el identificador único de una transacción puede ser: 83+20091028+152114+A00CHZF3R9+A00CHZF3RA (36 caracteres).

3. Opcionalmente, si es necesario distinguir entre las transacciones realizadas por tabla y los registros se debe leer el contenido del campo *bandera*. Si este campo está vacío la transacción ha sido realizada por tabla, si contiene una “R” es un registro.

4. Extraer la información necesaria. Refiérase a la hoja Excel que acompaña esta documentación para un listado completo de todos los campos contenidos en el mensaje *Orden_Transaccion*.

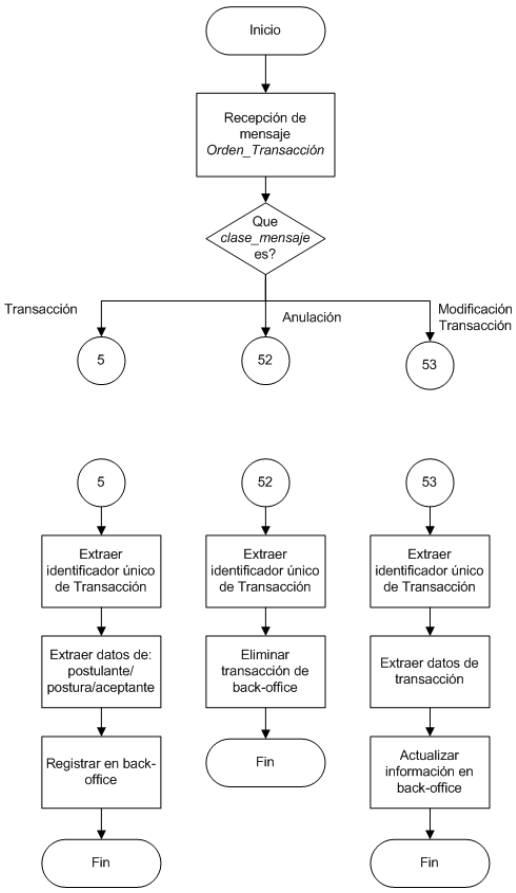



**Figura 4: Driagrama de flujo del proceso de un mensaje ORDEN_TRANSACCION para envío al back-office.**

***Nota:*** *Este diagrama es un ejemplo y puede ser tomado como base para la programación del proceso de transacciones. Sin embargo, puede haber otras variables propias de cada ambiente que deben ser tomadas en cuenta por cada cliente, el momento de diseñar/desarrollar una aplicación que usa el DAPI.*

# 9. Configuración del cliente DAPI (dapi.cfg)

El administrador del sistema debe crear en el DMCliente al DAPI como un usuario independiente. Además, debe proveer al usuario de esta interfaz el archivo DAPI.cfg, que contiene los siguientes datos: nombre de usuario, contraseña, código de sucursal, código de usuario, entre otros datos. A continuación se muestra un archivo DAPI.cfg de ejemplo. El carácter '# 'se utiliza para marcar el inicio de un comentario.

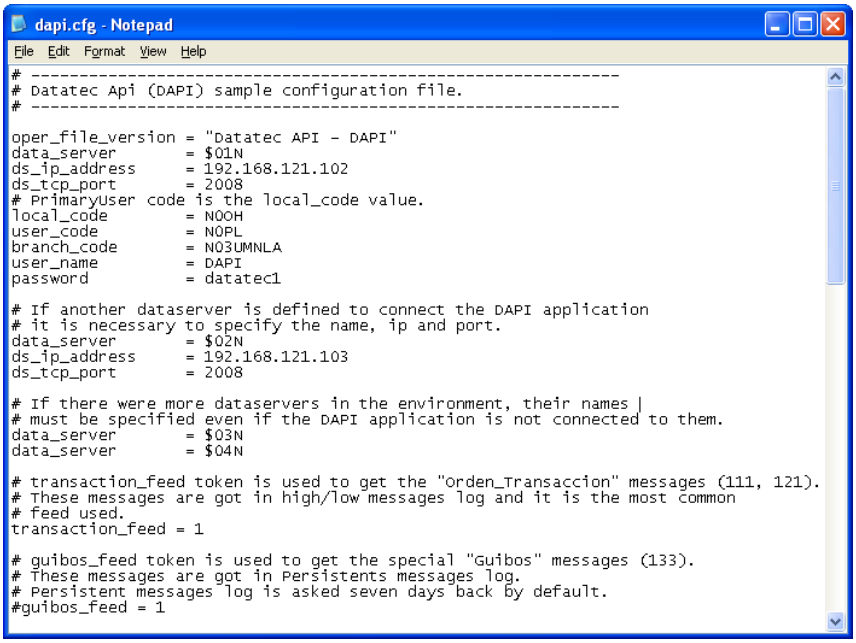



Figura 5: Achivo DAPI.cfg

A continuación, se muestra la lista de declaraciones permitidas en el archivo DAPI.cfg:

| **Declarativa** | **Ejemplo** | **Descripción** |
|---|---|---|
| oper_file_version | Transaction Feed API | Versión |
| local_code | G0OC | Código de PrimaryUser asignado al DAPI por el Administrador del Sistema. |
| user_code | G0IM | Código de usuario asignado por el Administrador del Sistema. Este código es diferente al código de PrimaryUser. |
| send_mode | 2 | Protocolo de comunicación.<br>0 - Cifrado IDEA (Obsoleto)<br>1 - No encriptado, No comprimido.<br>2 - Compresión BSD o RLE, dependiendo del tamaño del mensaje (Omisión)<br>3 - Comprimido y Encriptado con IDEA (Obsoleto). |

| | | |
|---|---|---|
| use_ssl | 0 | 0 - Sin ssl (Omisión).<br>1 - Conexión con SSL.<br><br>Si el valor es 1, que es el valor recomendado para conexiones seguras, los siguientes archivos deben ser copiados al mismo directorio donde se ejecuta la aplicación que usa el DAPI:<br>**ssl.cfg (with –v 3 option)**<br>**ssl/certs/server.pem**<br>**ssl/certs/cuduser.pem**<br>**ssl/private/ca.crt**<br>- Los archivos *.pem con sus nombres cambiados a sus números hash bajo el subdirectorio ssl. Por ejemplo:<br>**ssl/certs/6_users/5b7f9658.0**<br>**ssl/certs/6_users/5f552eb8.0**<br>Adicionalmente, el argumento ***ds_tcp_port*** debe ser configurado al puerto SSL del Dataserver.<br>***Nota:*** *Estos archivos son incluidos en los ejemplos de Linux y Windows.* |
| branch_code | G0JKUIOA | Código de sucursal de usuario. |
| data_server | $02Q | Código del equipo DataServer.<br><br>***Nota:*** *Si otros Dataservers son definidos para que el DAPI se conecte, se debe especificar su nombre, IP y puerto (ds_ip_address* y *ds_tcp_port).*<br><br>***Nota:*** *Se deben definir tantos DataServer como cuantos están corriendo en el Sistema de Datatec, para que el DAPI reciba los mensajes originados en todos los Dataserver.* |
| ds_ip_address | 192.168.112.22 | Dirección ip del equipo DataServer previamente definido con la directiva data_server. |
| ds_tcp_port | 2008 | Puerto tcp del equipo DataServer previamente definido con la directiva data_server.<br>Si el argumento ***use_ssl*** es 1, entonces este puerto debe ser el puerto SSL que el *data_server* está escuchando.<br>Cuando se conecta una aplicación vía Internet, el puerto SSL es usualmente el 443. |
| transaction_feed | 1 | 0 – No recibir mensajes de transacciones realizadas en el sistema<br>1 - Recibir los mensajes de las transacciones realizadas en el sistema.<br><br>***Nota:*** *Los parámetros guibos_feed y transaction_feed no deben ser especificados simultáneamente. Solo uno puede ser definido con valor 1.* |

| | | |
|---|---|---|
| guibos_feed | 1 | 0 – No recibir mensajes de Guibos emitidos por el sistema – esto se usaría para DAPIs configurados para recibir únicamente información de estadística.<br>1 – Recibe mensajes de Guibos.<br><br>***Nota:*** *Los parámetros guibos_feed y transaction_feed no deben ser especificados simultáneamente. Solo uno puede ser definido con valor 1.*<br><br>***Nota****: los mensajes Guibos son* ***persistentes*** *(no tienen mensajes de log alta y baja prioridad) y el DAPI solicita por defecto los mensajes persistentes de log de los últimos 7 días. Si existen mensajes procesados el log de persistentes se solicita desde la fecha del último mensaje marcado como procesado. Por lo tanto, las siguientes opciones no son usadas por guibos_feed:*<br>- *log_days_high_messages,*<br>- *log_days_low_messages y*<br>- *log_last_processed_message* |
| table_code | 6901<br>Posturas_Vigentes | Recibir registros de tablas globales con el código que se indica, y tomarlos como mensajes del tipo adyacente.<br>Formato:<br>table_code <codigo_tabla> <tipo_mensaje><br>donde:<br><codigo_tabla> Es el Código de la tabla a suscribirse. Esta información debe ser suministrada por Icap del Ecuador.<br><tipo_mensaje > Es el tipo de mensaje y este depende directamente del tipo de tabla cuyo código se indica en el argumento anterior.<br><br>Varias tablas pueden tener un mismo tipo de mensaje. Los códigos de tablas y tipos de tabla serán proporcionados por Icap del Ecuador y dependen del mercado y flujo de datos al que se conecta el DAPI.<br><br>***Nota****. Por ejemplo, para los mercados de Renta fija 69,140, 141 las tablas son:*<br>*31 Grupos_Vigentes*<br>*36 Mercados_Renta_Fija_Vigentes*<br>*6906 Resumenes_Vigentes*<br>*14006 Resumenes_Vigentes*<br>*6901 Posturas_Vigentes*<br>*6902 Posturas_Vigentes*<br>*14001 Posturas_Vigentes*<br>*14002 Posturas_Vigentes*<br>*14101 Posturas_Vigentes*<br>*14102 Posturas_Vigentes*<br>*6907 Hechos_Vigentes*<br>*6905 Indices_Vigentes* |

| | | |
|---|---|---|
| testing_file | dapis.log | Indica el nombre del archivo de log que se utilizará para retroalimentar el DAPI. Esta opción permite probar la funcionalidad del DAPI mediante un conjunto de mensajes intercambiados en de una conexión real. De esta forma el DAPI puede recibir mensajes sin tener que estar conectado a un ambiente de pruebas. Mediante esta opción es posible además depurar posibles errores encontrados en una ejecución anterior de la que se tenga el archivo de log. |
| testing_start | 20091215091500 | Especifica la fecha (AAAAMMDD) y hora (hhmmss) de inicio desde la cual el DAPI empieza a leer el archivo testing_file. |
| testing_end | 20091215091500 | Especifica la fecha (AAAAMMDD) y hora (hhmmss) en que el DAPI termina la ejecución de la prueba. |
| testing_output_file | Test_dapi_output.log | Especifica el nombre del archivo de salida de la prueba. |
| language | SPANISH | Indica el idioma que se emplea en los tipos de los mensajes que se pasan a la aplicación.<br>Si esta directiva no es definida se utilizarán los nombres antiguos.<br><br>Si se define language=SPANISH, se utilizará el tipo de mensaje en español.<br>Ejemplo:<br>*mensaje_trans_moneda*<br><br>Si se define *language=ENGLISH*, se utilizará el tipo de mensaje en inglés.<br>Ejemplo:<br>*currency_trade_message* |

| | | |
|---|---|---|
| log_last_processed_message | N, F, o L | Valor de omisión: L.<br>Controla cual es el mensaje a partir del cual se solicita la retransmisión de mensajes de alta y baja prioridad cuando el Dapi se reconecta con el servidor, tomando en cuenta los parámetros log_days_high_messages y log_days_low_messages.<br>El significado aplica según la siguiente tabla:<br>F: Valor de OMISION. El mensaje más antiguo posible y que todavía no está marcado como procesado.<br>L: El mensaje más reciente posible y que la aplicación lo ha marcado como procesado. En este caso cualquier mensaje anterior que la aplicación NO MARCÓ como procesado ya no será solicitado.<br>N: No hay mensaje a considerar, siempre se solicita la retransmisión de todos los mensajes desde el inicio del día.<br>***Nota:*** *Si la aplicación siempre indica como procesados a todos los mensajes recibidos del Dapi, no existe diferencia real en los mensajes que el Dapi entregará a la aplicación puesto que los mensajes marcados como procesados son descartados, se exceptúa claro está el caso en que los archivos de log del Dapi sean previamente borrados.*<br>Ejemplo:<br>*log_days_high_messages = 0*<br>*log_days_low_messages = 0*<br>*log_last_processed_message = N*<br>En caso de reconexión el DAPI siempre solicita al servidor la verificación de todos los mensajes enviados en el día de hoy.<br>Ejemplo:<br>*log_days_high_messages = 0*<br>*log_days_low_messages = 0*<br>*log_last_processed_message = L*<br>En caso de reconexión el DAPI solicita al servidor la verificación de los<br><br>mensajes a partir del último marcado como procesado en el día de hoy. Todo mensaje generado el día de ayer o anteriores no es tomado en cuenta. |
| use_msg_code | 121 in markets 69 140 141 | Indica que el número de mensaje usado para el mensaje Orden_Transaccion.<br><br>En el ejejmplo se especifica que los mercados 69, 140 y 141 usarán el mensaje 121 para Orden_Transaccion en vez del valor por defecto (111). |
| send_msg_delay_millisec | 500 | Retraso en milisegundos usado para el envío de mensajes. |

***NOTA:*** *Todos los DataServers disponibles en el sistema a los cuales el DAPI se conectará deben incluirse en el archivo DAPI.cfg.*

Cada DataServer debe ser identificado por un conjunto de tres parámetros:

* data_server * ds_ip_address * ds_tcp_port

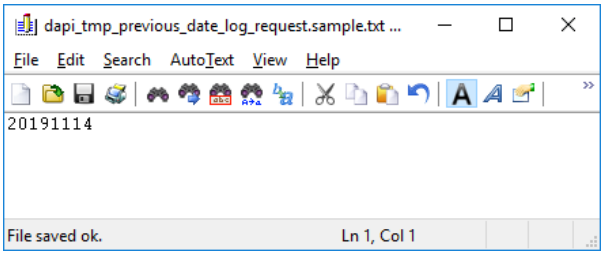
Con todos los DataSevers configurados, se permitirá que la aplicación DAPI pueda conectarse al sistema con otro DataServer en caso de que pierda conexión con el primero. De esta manera, reduce el tiempo fuera de servicio. De la misma manera, si hay DataServers adicionales en el sistema sus nombres deben ser configurados aun cuando el DAPI no se conecte a ellos.

En las aplicaciones de ejemplo distribuidas para plataformas soportadas se incluye un archivo de ejemplo dapi.cfg.sample.

## 9.1. Temporal Configuración Archivo (dapi_tmp_previous_date_log_request.txt)

Extraordinariamente el administrador puede proveer el archivo temporal *dapi_tmp_previous_dat_log_request.txt*, que se utiliza para indicar al dapi que solicite la retransmisión de todos los mensajes de un día definido en el pasado. Este archivo debe crearse en el mismo directorio de dapi.cpf y en el directorio actual de la aplicación. Como precaución para evitar futuras retransmisiones, el archivo se elimina después de ser leído.

La fecha requerida debe definirse en los primeros caracteres de la primera línea y en formato YYYYMMDD, si hay algún error al leer o eliminar el archivo, ó con el reconocimiento de la fecha, se devuelve error y no se continúa con el proceso normal del DAPI.




Los posibles errores son:

| Error | Code | Description |
|---|---|---|
| DAPI_ERR_INVALID_DATE | 27 | Invalidate date in the tmp config file |
| DAPI_ERR_CANNOT_DELETE_TMP_CONFIG | 28 | Can not delete tmp config file |
| DAPI_ERR_READING_TMP_CONFIG | 25 | Reading temporal config file |
| DAPI_ERR_EMPTY_TMP_CONFIG | 26 | Empty temporal config file |

# 10. Instalación del DAPI en Windows

La distribución del DAPI para Windows contiene los siguientes archivos:

- **windows\ActiveX\dxapi.ocx** (ActiveX control).
- **windows\lib\dcapi.lib** (Librería).
- **windows\lib\dxapi.lib** (Librería).
- **sample\windows\x86\vbdxapitest** (app ejemplo en VB usando OCX para 32 bits).
- **sample\windows\x86\drvapitest** (app ejemplo en C usando librerías para 32 bits).
- **windows64\ActiveX\AxdxapiLib.dll.**
- **windows64\ActiveX\dxapi.ocx.**
- **windows64\ActiveX\dxapiLib.dll.**
- **windows64\lib\dcapi.lib.**
- **windows64\lib\dxapi.lib.**
- **windows64\sample\vbdxapitest** (app ejemplo en VB usando OCX para 64 bits).
- **windows64\sample\drvapitest** (app ejemplo en C usando librerías para 64 bits)

## 10.1. Instalación del control ActiveX (OCX)

El DAPI ActiveX control se distribuye con los siguientes archivos:

1. El archivo **dxapi.ocx**
2. Un archivo de configuración **dapi.cfg**
3. Un archivo temporal de configuración **dapi_tmp_previous_date_log_request.txt**

Si ya se ha instalado una versión anterior del DAPI en el equipo, antes de realizar la instalación de la nueva versión, RETIRE la instalación antigua ubicándose en el directorio donde se encuentra el archivo anterior *dxapi.ocx* y ejecute la siguiente línea:

***Regsvr32 /u dxapi.ocx***

Copie el nuevo **dxapi.ocx** al disco duro del equipo de destino y registre el ActiveX control usando la siguiente línea de comando:

***Regsvr32 dxapi.ocx***

Esto instala el ActiveX control en el equipo y puede ser usado por la aplicación del usuario generada con MS Visual Studio.

En Windows de 64bits el ejecutable se debe ubicar en la ruta \windows\sysWOW64.

***Nota:*** *Estos comandos deben ser ejecutados con privilegios de Administrador o abriendo una consola con la opción "Ejecutar como Administrador", especialmente para Windows de 64 bits.*

A continuación, configure los parámetros del archivo **dapi.cfg** de acuerdo con el ambiente y Sistema de Datatec donde se corre la aplicación del usuario.

Finalmente, hay que incluir en la aplicación el código necesario para interactuar con el DAPI. Usualmente se requiere incluir el ActiveX control dentro de una ventana (p.e. dialog application), crear una variable de este control y redefinir los eventos necesarios (principalmente el evento *OnMessageReceived,* esto será discutido más adelante). Si se requiere enviar mensajes al sistema de Datatec se deben llamar a los métodos que la clase del DAPI proporciona para este propósito con la misma variable antes creada.

### 10.1.1. Revisión de la versión DAPI en el archivo OCX

Es posible revisar la versión del DAPI en el archivo OCX. Para esto, **clic-derecho** en el archivo, opción **Propiedades**, escoger la **viñeta Details**.

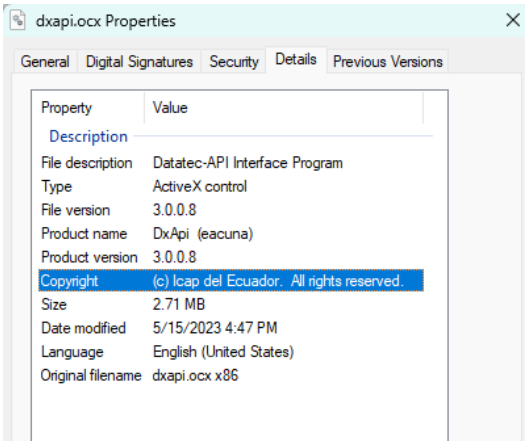



Figura 6: Propiedades de dxapi.ocx

## 10.2. Instalación y uso de la librería DAPI de Windows

En caso de usar la librería del DAPI para Windows, copiar el código del DAPI correspondiente al sistema operativo y compilador deseado en el directorio del proyecto e incluir las librerías en el código que construye el proyecto.

A continuación, ajustar los parámetros del archivo **dapi.cfg** de acuerdo al ambiente y Sistema de Datatec donde se corre la aplicación del usuario.

Finalmente revisar y codificar los métodos definidos en el archivo *drvapi.cpp*, especialmente aquellos métodos que atienden a los eventos que el DAPI genera durante su ejecución. De ser necesario se puede crear una nueva clase que tenga como base la clase CdcApi.

## 10.3. Ejemplo de uso del DAPI en Windows - VbdxApiTest

### 10.3.1. Ejemplo Visual Basic del uso del DAPI

Para ilustrar el uso del DAPI y sus funciones se suministra un fácil y sencillo ejemplo programado en VB. Se encuentra en la carpeta **\sample\windows\vbdxapitest** de la distribución del DAPI. Para usarlo copie el contenido del directorio en el equipo donde desea correr el ejemplo y siga los siguientes pasos:

1. Configure el usuario DAPI usando el DataManager Cliente.
2. Actualice el archivo de configuración dapi.cfg con la información del usuario DAPI y los DataServers a los que puede conectarse.
3. Asegúrese que el archivo *dapi.cfg* se encuentre en el mismo directorio donde se ejecutará la aplicación de ejemplo *vbdxapitest*.
4. Asegúrese que el archivo *dapi_tmp_previous_date_log_request.txt* se encuentre en el mismo directorio donde se ejecutará la aplicación de ejemplo *vbdxapitest*.
5. El directorio desde donde se ejecutará la aplicación de ejemplo debe estar ubicado en un disco local, ya que ésta ***no se ejecuta correctamente en unidades de red.***
6. El control ActiveX *dxapi.ocx* debe estar correctamente registrado en el equipo (véa la sección Instalación del DAPI en Windows).

Ejecute el archivo vbdxapitest.exe, debe presionar el botón ***Login*** y se puede ignorar los datos de usuario y clave que se solicita.

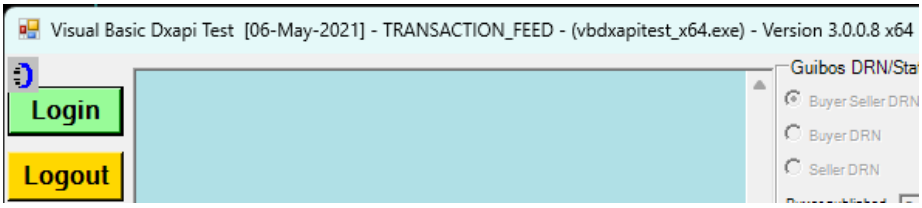



Figura 7: Ventana de ejemplo DAPI windows

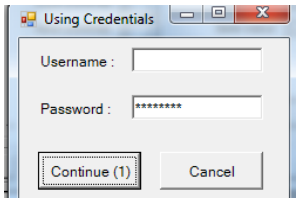



Figura 8: Ventana de credenciales que se puede ignorar.

La siguiente imagen muestra un ejemplo de la ventana luego de un ingreso exitoso al sistema.

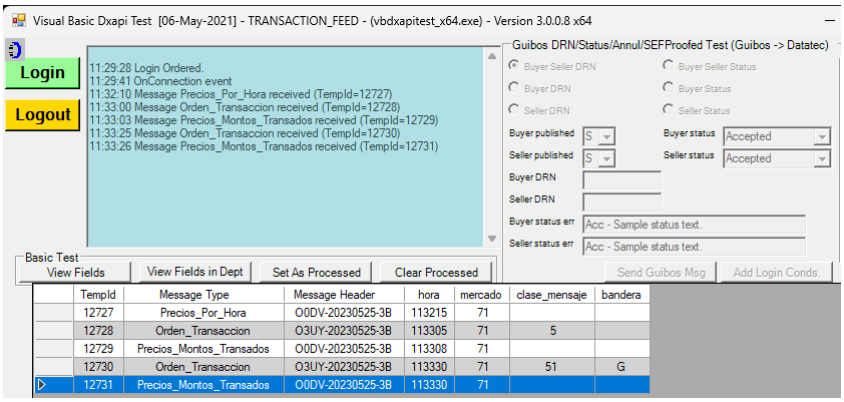



Figura 9: Ventana de ejemplo DAPI windows.

En este ejemplo el control ActiveX (*dxapi.ocx*) se incluye en la plantilla (*Form1*). El nombre que tiene este control en la aplicación de ejemplo es: *Axdxapi1*.

### 10.3.2. Ejemplo Visual Basic del uso del DAPI

El botón ***Login*** llama el método ***Login*** de ***Axdxapi1*** tal como se puede apreciar en las líneas de código de ***Form1.vb***:

```
 Public Sub StartTesting_Click(…) Handles StartTesting.Click…
          …
login_res = Axdxapi1.Login(username, password)
          …
 End Sub
```

De una manera similar, el botón ***Logout*** llama el método ***Logout*** de ***Axdxapi1***.

Para manejar eventos en VB se debe utilizar el método ***invoke***. Este método evita conflictos de ejecución concurrente entre la aplicación VB y el ***dxapi.ocx***. El siguiente extracto de código muestra el proceso:

```
Public Delegate Sub Delegate_OnConnection(ByVal e As
                    AxdxapiLib._DdxapiEvents_OnConnectionEvent)
  ....
  ....
```

```
Public Sub OnOnConnection(ByVal sender As Object, ByVal e As _
  AxdxapiLib._DdxapiEvents_OnConnectionEvent) Handles Axdxapi1.OnConnection

  ' AQUI, No se ejecuta directamente el codigo, sinó que se delega
  ' la ejecución para que lo ejecute el mismo thread de la aplicación
  ' de Visual Basic
  TextBox1.Invoke(New Delegate_OnConnection(AddressOf Dlgt_Onconnection), _
          New Object() {e})
End Sub

Public Sub Dlgt_Onconnection(ByVal e As AxdxapiLib._DdxapiEvents_OnConnectionEvent)

  ' AQUI, SI se ejecuta el código necesario sin problemas de conflicto
  ' con el código de la aplicación
  PrintText("OnConnection Event", TO_SCREEN_AND_LOG)
  PrintText([String].Concat("Result Code:", e.result_code), TO_LOG)
End Sub
```

El problema de ejecución concurrente no sucede por diseño en otros lenguajes de programación tales como Visual C++ o Visual C#.

### 10.3.3. Botones View Fields y View Fields in Depth

El botón ***View Fields*** es un ejemplo de cómo se debe acceder a todos los campos dentro de un mensaje, mientras que el botón***View Fields in Depth*** muestra los campos detallados que dependen de campos anteriores. La parte esencial del código se muestra a continuación:

```
   Dim fieldvalue As Object
   Dim fieldstr As String

   Get the message from OCX
Axdxapi1.TakeMessage (msg_type, msg_id)

   'Get the first field name
   fieldstr = Axdxapi1.GetNextFieldName(fieldstr)

   'Go through all the message fields until the end
   While fieldstr.Length > 0
     If fieldstr.IndexOf(".") = -1 Then
       'Get the field name value
       fieldvalue = Axdxapi1.GetField(fieldstr)

           'Take field name fieldstr and value fieldvalue
           …
           …
     End If
     'Get the next field name of the message
     fieldstr = Axdxapi1.GetNextFieldName(fieldstr)
   End While
```

El resultado de la ejecución de se pasa a una ventana y aparece como sigue:

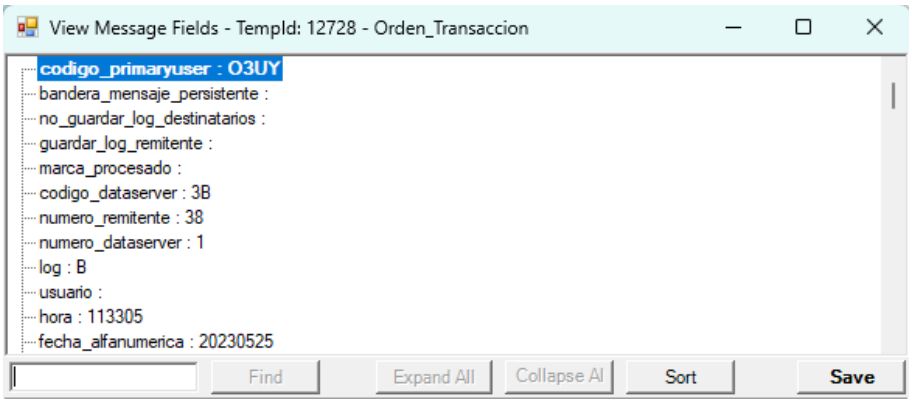



Figura 10: Contenido botón View Fields

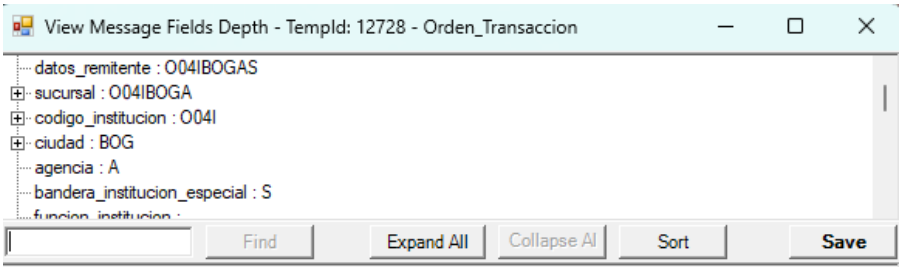



Figura 11: Contenido botón View Fields Depth.

### 10.3.4. Botón Set As Processed

El botón ***Set As Processed*** llama al método ***FinishMessageProcess,*** que es necesario para que el DAPI pueda marcar el mensaje como procesado y libere los recursos asignados a este proceso. De esta manera evita la recepción y reproceso del mensaje en caso de una desconexión.

### 10.3.5. Botón Add Login Conditions

El botón ***Add Login Conditions*** permite configurar las condiciones que debe cumplir un mensaje recibido para ser o no desplegado en la pantalla principal de la aplicación de ejemplo. Este botón puede ser usado sólamente antes de conectar el DAPI, ya que estas condiciones se envían al DataServer en el momento de ingreso al sistema. Cuando se recibe un mensaje, este será revisado contra las condiciones configuradas y si no las cumple entonces es marcado como mensaje procesado de manera similar que usando el método ***FinishMessageProcess***. El ejemplo muestra cómo se debe configurar las condiciones para mensajes Guibos (***guibos_feed***).

Este botón utiliza el método ***AddLoginCondition***.

# 11. Instalación del DAPI en Linux

La distribución para Linux contiene los siguientes archivos:

- linux7\lib\libdapi.a
- sample\linux7\lnxdrvapitest.zip (librería y código de ejemplo desarrollado en C++).

## 11.1. Configuración en Linux

El DAPI para el sistema operativo Linux ha sido generado y probado en un ambiente con las siguientes características:

Para 32 bits en equipo Linux de 32 bits:

- Red Hat Enterprise Linux Server release 8.
- **gcc (GCC) 8.3.1**
- Librería de 32 bits compilada en S.O. de 32 bits.
- Librería de 62 bits compilada en S.O. de 64 bits.

## 11.2. OpenSSL

No se requiere compilar la librería OpenSSL.

## 11.3. Instalación y uso de la librería DAPI en Linux

Luego de seleccionar la plataforma para el Dapi y desempaquetar el archivo lnxdrvapitest.zip; se obtiene el directorio **lnxdrvapitest** y bajo éste, una lista de otros directorios y archivos.

La librería del DAPI **libdapi.a** se distribuye compilada usando el compilador gcc de la versión indicada previamente. Por lo tanto, para usar los métodos del DAPI se debe enlazar la aplicación que se esté desarrollando con esta librería que se encuentra en: **lnxdrvapitest /libdapi/lib/libdapi.a**

El archivo header de la libería DAPI es *dcapi.h* y se encuentra en: **lnxdrvapitest/dcapi/dcapi.h**

## 11.4. Contenido del directorio de la aplicación de ejemplo y de la librería

Luego de seleccionar la plataforma para el Dapi y desempaquetar el archivo lnxdrvapitest.zip; se tiene la siguiente estructura de directorios y archivos:

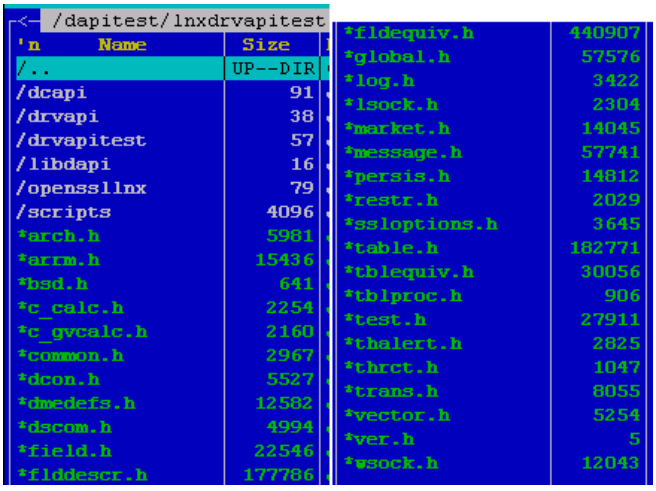



Figura 12: Esttructura de DAPI Linux.

## 11.5. Compilación de la librería DAPI en Linux

**NO se requiere compilar la librería**, porque ya se incluye el compilado de la misma. Los archivos fuentes ya no son distribuidos.

Los archivos libdapi.a y drvapitest están compilados para 32 y 64 bits.

## 11.6. Compilación de la aplicación ejemplo en Linux (drvapitest)

La distribución del DAPI para Linux incluye una aplicación de ejemplo del uso del DAPI (**libdapi.a**) programada en C++.

El programa **drvapitest** se distribuye ya compilado y se encuentra en: **/lnxdrvapitest/drvapitest/**

En caso de que el cliente desee compilar el programa drvapitest, se sugiere instalar:

Para 32 y 64 bits en equipo Linux:

- gcc 8.3.1
- “glibc-devel-2.5-123.i386.rpm”

Para compilar la aplicación de ejemplo de uso del DAPI se debe utilizar los siguientes comandos:

**[myuser@workdir] $ .cd lnxdrvapitest/scripts**
**[myuser@scripts] $ ./mk drvapitest clean**
**[myuser@scripts] $ ./mk drvapitest**

**NOTA**: Este script lo que hace es compilar nuevamente el ejecutable drvapitest, mas no compila la librería libdapi.a, ya que no se distribuyen los archivos fuentes, el ejecutable del ejemplo drvapitest se encuentra en: **lnxdrvapitest/drvapitest/xdebug.**

Cabe recalcar que varios de los procesos para generar la librería del Dapi (***libdapi.a***) y su correspondiente ejemplo se han automatizado mediante el uso de scripts y estos deberán estar en el formato en el que el shell espera encontrarlos para ejecutarlos, es decir, el formato Unix.

Los formatos de los scripts pueden verse afectados al copiarlos desde un sistema operativo basado en la plataforma Windows. Si hay problemas al ejecutar los scripts debido a incompatibilidad de los formatos se sugiere utilizar el comando siguiente:

**[myuser@scripts] $ dos2unix mk**

El comando anterior convierte el script ***mk*** al formato Unix.

## 11.7. Errores al compilar ejemplo DAPI en Linux

Si no se instala las librerías indicadas en el punto anterior se tienen errores como el siguiente:

```
[datatec@support05 scripts]$ ./mk drvapitest
Generating ...
APP NAME:     drvapitest
OS:           linux-gnu
PATH GCC:      /usr/bin/gcc
RHEL VERSION:  Red Hat Enterprise Linux Server release 5.8 (Tikanga)
Kernel \r on an \m
GCC VERSION:   gcc (GCC) 4.1.2 20080704 (Red Hat 4.1.2-52)
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
-> /home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/drvapitest/xrelease/drvapi.o
In file included from /usr/include/features.h:352,
                 from /usr/include/stdio.h:28,
                 from /home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/drvapitest/../drvapi/drvapi.cpp:6:
/usr/include/gnu/stubs.h:7:27: error: gnu/stubs-32.h: No such file or directory
-> /home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/drvapitest/xrelease/drvapitest
g++: /home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/drvapitest/xrelease/drvapi.o: No such file or directory
In file included from /usr/include/features.h:352,
                 from /usr/include/stdio.h:28,
                 from /home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/drvapitest/drvapitest.cpp:6:
/usr/include/gnu/stubs.h:7:27: error: gnu/stubs-32.h: No such file or directory
```

Figura 13: Errores de compilación sin librerías.

Para instalar las librerías de 32 bits se debe utilizar el RPM **glibc-devel-2.5-81.i386.** Con el comando **yum** se puede investigar exactamente qué RPM tiene los archivos necesarios, los RPM están en los ISOs de instalación y se puede montar todo el ISO o copiar uno a uno el RPM según haga falta.

```
[root@support05 Server]# yum whatprovides *stubs-32.h
Loaded plugins: katello, product-id, security, subscription-manager
Updating certificate-based repositories.
Unable to read consumer identity
glibc-devel-2.5-81.i386 : Object files for development using standard C libraries.
Repo        : server
Matched from:
Filename    : /usr/include/gnu/stubs-32.h



[root@support05 Server]# rpm -Uvh glibc-devel-2.5-81.i386.rpm
Preparing...                ########################################### [100%]
   1:glibc-devel            ########################################### [100%]
```

Figura 14: Uso de YUM e instalación de RPM.

Con las librerías de 32 bits instaladas se puede proceder con la compilación del ejemplo, pero, si se utiliza el archivo libdapi.a compilado con GCC 8.3.1 se obtienen errores como estos:

```
[datatec@support05 scripts]$ ./mk drvapitest
Generating ...
APP NAME:     drvapitest
OS:           linux-gnu
PATH GCC:      /usr/bin/gcc
RHEL VERSION:  Red Hat Enterprise Linux Server release 5.8 (Tikanga)
Kernel \r on an \m
GCC VERSION:   gcc (GCC) 4.1.2 20080704 (Red Hat 4.1.2-52)
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
-> /home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/drvapitest/xrelease/drvapi.o
-> /home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/drvapitest/xrelease/drvapitest
/home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/libdapi/xrelease/libdapi.a(dcapi.o): In function `std::__simple_alloc<std::_Rb_tree_node<std::pair<CIdxKey const,
 int> >, std::__default_alloc_template<true, 0> >::deallocate(std::_Rb_tree_node<std::pair<CIdxKey const, int> >*, unsigned int)':
/usr/local/gcc-3.2.3/include/c++/3.2.3/bits/stl_alloc.h:248: undefined reference to `std::__default_alloc_template<true, 0>::deallocate(void*, unsigned int)'
/home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/libdapi/xrelease/libdapi.a(dcapi.o): In function `std::__simple_alloc<std::_Rb_tree_node<std::pair<CFixedIncomeMa
rketsKey const, int> >, std::__default_alloc_template<true, 0> >::deallocate(std::_Rb_tree_node<std::pair<CFixedIncomeMarketsKey const, int> >*, unsigned int)':
/usr/local/gcc-3.2.3/include/c++/3.2.3/bits/stl_alloc.h:248: undefined reference to `std::__default_alloc_template<true, 0>::deallocate(void*, unsigned int)'
/home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/libdapi/xrelease/libdapi.a(dcapi.o): In function `std::__simple_alloc<std::_Rb_tree_node<std::pair<CMarketsKey co
nst, int> >, std::__default_alloc_template<true, 0> >::deallocate(std::_Rb_tree_node<std::pair<CMarketsKey const, int> >*, unsigned int)':
```

Figura 15: Error al compilar con GCC erróneo.

Al cambiar la librería por la que es compilada con GCC 8.3.1 la compilación es exitosa.

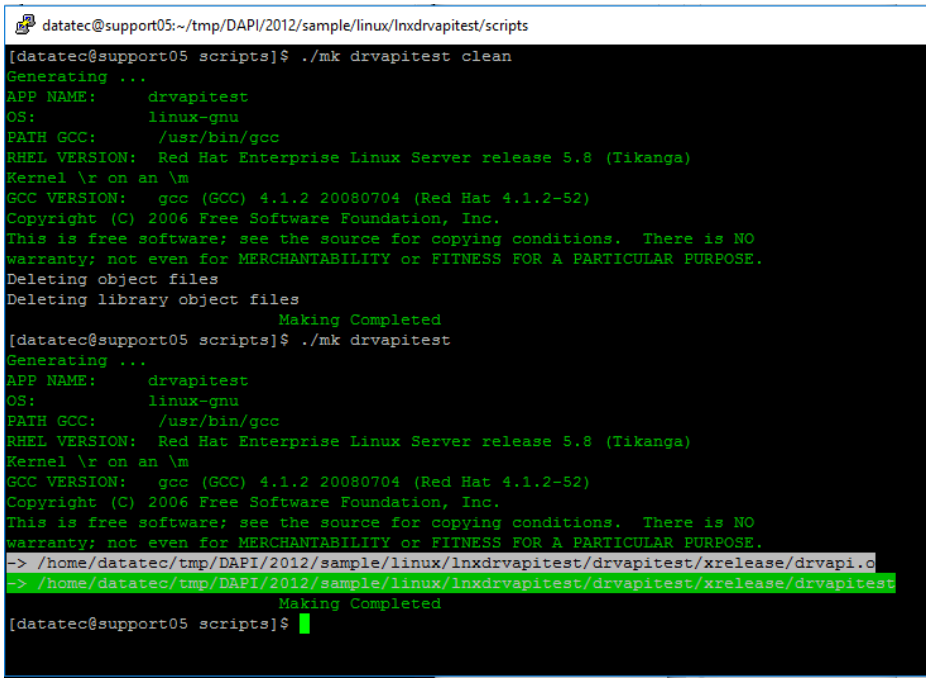



Figura 16: Compilación exitosa.

## 11.8. Ejemplo de uso del DAPI en Linux

La distribución del DAPI para Linux cuenta con un ejemplo de la utilización de la librería ***libdapi.a*** mediante un ejecutable programado en C++ (***drvapitest***). Para que funcione correctamente esta aplicación debe asegurarse de cumplir el siguiente procedimiento:

1. Configure el usuario DAPI usando el DataManager Cliente.
2. Actualice el archivo de configuración dapi.cfg con la información del usuario DAPI y los DataServers a los que puede conectarse.
3. Asegúrese que el archivo *dapi.cfg* se encuentre en el mismo directorio donde se ejecutará la aplicación de ejemplo *drvapitest*.

El ejecutable de ejemplo del Dapi se encuentra bajo el directorio lnxdrvapitest/drvapitest/xrelease. Deberá entonces ubicarse en el directorio y ejecutar el archivo correspondiente de esta manera:

**[myuser@ xdebug]$ ./drvapitest**


Figura 17: Ejecución de ejemplo DAPI Linux.

### 11.9. Programa drvapitest para Windows

El programa drvapitest que se usa en Linux, se lo genera para Windows para las plataformas x86 y x64; el programa es de tipo consola,

El archivo ejecutable se llama drvapitest.exe

## 12. Descripción de los mensajes del DAPI

En la hoja de Excel que acompaña esta documentación se encuentra en detalle la descripción de cada uno de los mensajes (en hojas separadas) que pueden ser recibidos por el DAPI.

Cada mensaje se detalla utilizando las siguientes columnas:

**nombre del campo:** Este es el nombre en español del campo que debe ser utilizado cuando se hace referencia a un campo en el DAPI.

**field name:** Nombre del campo en inglés. Algunos campos están redefinidos como dos o más sub campos. En este caso, el campo maestro se encuentra con caracteres de color azul y los sub campos están indentados con cuatro espacios. El DAPI puede usar el campo maestro para acceder a todos los sub campos como una cadena concatenada de caracteres o el sub campo específico.

**field type:** El tipo de campo, que puede ser:

**alphanumeric:** Cualquier carácter alfanumérico o espacio (“ ”).

**alphabetic:** Cualquier letra A-Z, a-z, o espacio (“ “).

**time:** La hora expresada en formato HHMMSS.

**date:** La fecha expresada en formato AAAAMMDD.

**short:** Entero con signo de dos bytes, desde –32,768 hasta 32,767.

**long:** Entero con signo de cuatro bytes, desde $-2^{31}$ hasta $(2^{31} – 1)$

**numeric ascii:** Un número almacenado en caracteres tipo ASCII, usando un punto decimal si existen decimales.

**numeric ascii signed:** Un número almacenado en caracteres tipo ASCII, usando un punto decimal si existen decimales. Si el número es negativo se encuentra precedido por el símbolo menos (“-“).

**length:** Longitud total del campo. Si el campo es tipo *numeric ascii* y tiene decimales la longitud total es la suma de los dígitos enteros, los dígitos decimales y un carácter adicional para el punto decimal. Si el campo es de tipo *numeric ascii signed* la longitud total es la suma de los dígitos enteros, los decimales más uno (si existen decimales) y un carácter adicional para el signo.

**digits:** El número de dígitos de un campo tipo *numeric ascii*.

**decimals:** El número de decimales de un campo tipo *numeric ascii*.

**descripción:** Descripción del campo en idioma Español.

**description:** Descripción del campo en idioma Inglés.

# 13. Listado de códigos de error

| Código | Identificación de error |
|---|---|
| 1 | DAPI_ERR_INTERNAL_ERROR<br>Se trata de un error que el DAPI recibe desde el sistema operativo. Revisar la última operación que el DAPI ha intentado en el equipo. |
| 2 | DAPI_ERR_CONFIG_NOT_FOUND<br>No se pudo abrir el archivo de configuración cuyo nombre se indica. Verificar la existencia del archivo y que la aplicación tenga permisos de lectura para el mismo. |
| 3 | DAPI_ERR_CONFIG_BAD_DATA<br>Existe un error de sintaxis en el archivo de configuración o alguna directiva está errada. Verifique la línea que se indica. |
| 4 | DAPI_ERR_NOT_INITIALIZED_YET<br>Se ha requerido la ejecución sobre algún método, pero no se ha llamado al método de *login* primeramente. |
| 5 | DAPI_ERR_REJECTED_CONNECTION<br>El servidor de Datatec rechazó el intento de conexión. El motivo no está dentro de los formalmente reconocidos. Contacte a su operador y solicite el soporte respectivo. |
| 6 | DAPI_ERR_RC_NOT_AUTHORIZED_USER (X)*<br>El servidor de Datatec rechazó el intento de conexión por cuanto el usuario no está autorizado para conectarse. |
| 7 | DAPI_ERR_RC_USERNAME_DONOT_EXIST (N)*<br>El servidor de Datatec rechazó el intento de conexión por cuanto el nombre de usuario indicado no está registrado en el equipo Administrador o DataManager. |
| 8 | DAPI_ERR_RC_ILLEGAL_PASSWORD (C)*<br>El servidor de Datatec rechazó el intento de conexión por cuanto la contraseña entregada no está registrada como tal en el equipo Administrador de usuarios o DataManager. |
| 9 | DAPI_ERR_RC_NO_DMACHINES_TRY_LATER (D)*<br>El servidor de Datatec rechazó el intento de conexión por cuanto no existe ningún equipo Administrador de usuarios en línea en este momento que pueda validar este intento de conexión. |
| 10 | DAPI_ERR_RC_DUPLICATED_MACHINE_CODE (R)*<br>El servidor de Datatec rechazó el intento de conexión por cuanto ha detectado que otro usuario con los mismos códigos de máquina ya está conectado al sistema. |
| 11 | DAPI_ERR_RC_MACHINE_NOT_REGISTERED (E)*<br>El servidor de Datatec ha rechazado el intento de conexión porque el código de este equipo no está registrado entre sus equipos permitidos. Verificar que el Administrador de usuarios le actualice la información. |
| 12 | DAPI_ERR_RC_ILLEGAL_BRANCH_PASSWORD (L)*<br>El servidor de Datatec rechazó el intento de conexión porque la contraseña correspondiente a la sucursal es incorrecta. |
| 13 | DAPI_ERR_STATUS_NOT_EXPECTED<br>El estado de la conexión ha cambiado a un estado indefinido. Reporte este resultado a su operador. |
| 14 | DAPI_ERR_IMPROPER_OPERATION<br>La aplicación ha llamado a algún método del DAPI con argumentos con valor inapropiado. El error de detalle adjunto explica mas información con respecto al origen del problema. |
| 15 | DAPI_ERR_ILLEGAL_RECORD_TYPE_SPECIFIED<br>El nombre de registro dado como argumento al método es inapropiado. |
| 16 | DAPI_ERR_ILLEGAL_FIELD_NAME_SPECIFIED<br>El nombre de campo que se dio como argumento al método no existe. |
| 17 | DAPI_ERR_ALREADY_INITIALIZED<br>Ocurre cuando se ejecuta nuevamente el método *Login*. |
| 18 | DAPI_ERR_RECORD_NOT_FOUND<br>El mensaje cuyo número o identificación se han dado no pudo ser localizado. Posiblemente el parámetro dado es incorrecto, o el mensaje ha sido dado de baja al finalizar su proceso con el método *FinishMessageProcess*. |
| 19 | DAPI_ERR_CONDITION_EVALUATION<br>El DAPI no pudo evaluar las condiciones de recepción de mensajes dadas en el método |

| | |
|---|---|
| | *AddLoginCondition*, verifique que el número de operadores y operaciones se correspondan en la forma BNF. |
| 20 | DAPI_ERR_COULDNT_USE_LOG_FILES |
| | El archivo de registro (log) que se indica no ha podido ser abierto, verifique existan los privilegios apropiados en el directorio de ejecución actual y que no exista alguna restricción de apertura sobre el archivo por parte de otro proceso del equipo. |
| 21 | DAPI_ERR_NO_DS_DIRECTIVE |
| | El archivo de configuración no define ningún equipo servidor de Datatec o DataServer, definición necesaria a fin de completar la conexión al Sistema de Datatec. |
| 22 | DAPI_ERR_ILLEGAL_ADDRESS_TYPE_SPECIFIED |
| | No existe el tipo de dirección indicado. Verifique las direcciones. |

***Nota:*** *Los caracteres entre paréntesis (), corresponden al valor recibido del equipo DataServer al final de cada intento de conexión.*