



DATATEC API – DAPI

Guía del programador

Sistema de Mercados Financieros

Update 3.023

Colombia

14-sep-2026
ICAP del Ecuador
support@datatec-ec.com

INDICE

1. Historia documento	5
2. Objetivo	9
3. Descripción general del Sistema Datatec.....	9
3.1. Componentes servidor.....	9
3.2. Componente cliente	10
3.3. Interfaz para programas de aplicación Datatec (DAPI)	10
4. Plataforma donde se puede ejecutar el DAPI.....	10
4.1. Plataformas de desarrollo.....	10
4.2. Plataformas de instalación.....	11
4.3. Requerimientos mínimos de hardware para uso del DAPI	11
5. Cómo interactuar con el DAPI	11
6. Verificación de transmisión/recepción de mensajes	12
6.1. Pérdidas de conexión inesperadas	12
6.2. Pérdidas de conexión programadas.....	12
7. Métodos y eventos del DAPI.....	12
7.1. Métodos del DAPI.....	12
7.1.1. Método <i>AddLoginCondition</i>	13
7.1.2. Método <i>FinishMessageProcess</i>	14
7.1.3. Método <i>Login</i>	15
7.1.4. Método <i>Logout</i>	15
7.1.5. Método <i>GetNextFieldName (ActiveX)</i>	16
7.1.6. Método <i>GetNextField (Librería)</i>	16
7.1.7. Método <i>GetField (ActiveX)</i>	17
7.1.8. Método <i>GetField (Librería)</i>	17
7.1.9. Método <i>GetFieldBSTR (Librería)</i>	18
7.1.10. Método <i>GetFieldDATE (Librería)</i>	18
7.1.11. Método <i>GetFieldLONG (Librería)</i>	18
7.1.12. Método <i>InitMessage</i>	19

7.1.13. Método <i>ProcessMyEvents</i> (Librería)	19
7.1.14. Método <i>SendMessage</i>	20
7.1.15. Método <i>SendMessageTo</i>	20
7.1.16. Método <i>SetField</i> (ActiveX) y <i>SetFieldBSTR</i> (Librería)	21
7.1.17. Método <i>SetFieldDATE</i> (Librería)	21
7.1.18. Método <i>SetFieldLONG</i> (Librería)	22
7.1.19. Método <i>TakeMessage</i>	22
7.1.20. Método <i>IsDefined</i>	23
7.1.21. Método <i>GetCompilationType</i>	23
7.1.22. Método <i>GetVersion</i>	24
7.2. Eventos del DAPI	24
7.2.1. Eventos en el DAPI como ActiveX Control (OCX)	24
7.2.2. Eventos en el DAPI como librería en C++	24
7.2.3. Evento <i>OnMesageReceived</i>	24
7.2.4. Evento <i>OnConnection</i>	25
7.2.5. Evento <i>OnDisconnection</i>	25
7.2.6. Evento <i>OnError</i>	26
7.2.7. Evento <i>OnStatusChanged</i>	26
7.2.8. Evento <i>OnDataReceived</i>	27
8. Códigos de ejemplo del DAPI	28
8.1. Ejemplo de cómo procesar un mensaje en Visual Basic	28
8.2. Ejemplo de cómo enviar un mensaje en C++.....	29
8.3. Diagrama de flujo ejemplo para recoger información de una transacción para aplicaciones back-office	30
9. Configuración del cliente DAPI (dapi.cfg)	31
10. Instalación del DAPI en Windows	33
10.1. Instalación del control ActiveX (OCX).....	34
10.1.1. Revisión de la versión DAPI en el archivo OCX	34
10.2. Ejemplo de uso del DAPI en Windows - <i>VbdxApiTest</i>	35
10.2.1. Ejemplo Visual Basic del uso del DAPI	35
10.2.2. Ejemplo Visual Basic del uso del DAPI	36

10.2.3. Botones View Fields y View Fields in Depth	36
10.2.4. View Fields in Depth para Arreglos	37
10.2.5. Botón Set As Processed	40
10.2.6. Botón Clear Processed	40
10.2.7. Botón Send Guibos Message.....	40
10.2.8. Botón Add Login Conditions	40
10.2.9. Botón Add Login Conditions Generic	41
10.3. Instalación y uso de la librería DAPI de Windows	41
10.4. Ejemplo de uso de la librería DAPI en Windows - Drvapitest.exe.....	41
10.4.1. Ejemplo C++ del uso de la librería DAPI.....	41
11. Instalación del DAPI en Linux	43
11.1. Configuración en Linux	43
11.2. Instalación y uso de la librería DAPI en Linux	44
11.3. Contenido del directorio de la aplicación de ejemplo y de la librería.....	44
11.4. Compilación de la librería DAPI en Linux	44
11.5. Compilación de la aplicación ejemplo en Linux (drvapitest)	45
11.6. Errores al compilar ejemplo DAPI en Linux	45
11.7. Ejemplo de uso del DAPI en Linux	47
12. Listado de códigos de error.....	48

1. Historia documento

Fecha (aaaa-mm-dd)	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		- Revisión con cambios versión 1.061. - Nuevo parámetro <i>guibos_feed</i> para mensajes de Guibos. - Nuevo parámetro <i>Send_Msg_Delay_Millisec</i>. - Actualización de valores de configuración por omisión (para pedido de mensajes de log de alta/baja prioridad). - Actualización y explicación de otros valores de configuración. - Cambios en la aplicación de ejemplo de Windows para mensajes Guibos y reenvío al sistema de ciertos mensajes. - Actualización en la sección de compilación en Linux de la librería/aplicación de ejemplo. - Nuevo script mk
28-Jun-2010		- Revisión general del documento. - Notas aumentadas para la recompilación librería Linux (versiones gcc y soporte sistemas operativos 32-64 bits). - Nota soporte Windows Vista (32bits) y 7 (32bits). - DAPI 1.065. Ajustes generales para usar DAPI en lugar de DGAPI en ambientes que reciben mensajes Guibos. - DAPI 1.066. Dapi envía su versión en la conexión para que sea desplegada en la ventana de versiones en el DMcliente. - DAPI 1.067. Uso de OpenSSL 0.9.8j en lugar de 0.9.7a.
20-Sep-2010		- Sección 8.3 se aumenta Diagrama de flujo ejemplo para recoger información de una transacción para aplicaciones back-office. - Sección 9 se aumenta configuración del DAPI en el DMCliente. - DAPI 1.069 Soporte mensaje nuevo Mantenimiento_Transaccion_Renta_Fija. - Archivo "20100920 DAPI Interface Message Descriptions.xls"
28-Oct-2010		- Revisión general.

30-Jun-2011		- Instrucciones de ejecución ejemplo en Windows 64 bits (sección 11.3.1). Regeneración ejemplo Windows (v. 1.077). - Revisión versión DAPI en archivo OCX (sección 11.1.1).
22-Dec-2017		- Actualización de documento en: Plataformas de Desarrollo e Instalación. Código de ejemplo del DAPI. Instalación del DAPI en Windows. Instalación del DAPI en Linux. Configuración por omisión en Linux. OpenSSL. Instalación y uso de la librería DAPI en Linux. Compilación de la librería DAPI en Linux. Compilación de la aplicación ejemplo en Linux (drvapitest). Ejemplo de uso del DAPI en Linux.
29-Oct-2018		- 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. - 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: GetCompilationType GetVersion - Documentación de la función IsDefined. - Generación del ejemplo Visual Basic con VisualStudio 2013 para las plataformas x86 y x64.
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.
15-May-2023	Equipo de Desarrollo	- Nuevas funciones para que sean compatibles con 64 bits. - Se modifica el tipo de dato en el código fuente del dapi de long a int para compatibilidad entre 32 y 64 bits.

9-Jun-2023	Equipo de Desarrollo	- LINUX: Se incluye el argumento -std=c++11, para compilar con 32 bits las librerías del dapi con gcc 8.3.1 - WINDOWS: Se requiere distribuir la versión debug de la librería Windows para el dapi. Se incluye las librerías debug en el directorio del programa drvapitest para windows, los directorios son lib32d y lib64d, para indicar que son debug. Y también se incluye los archivos .h necesarios para la compilación del programa drvapitest. - Se actualiza drvapitest windows para las versiones 3.008 y se incluye en la nueva versión 3.009.
4-Jul-2023	Equipo de Desarrollo	- Las versiones de dapi para linux7, son generadas utilizando el gcc 8.3.1 tanto para x86 como para x64. El drvapitest es generado con la versión 8.3.1
1-Ago-2023	Equipo de Desarrollo	- Ya no se genera el dapi para linux5. - Cuando se realizaba la consulta para visualizar en profundidad los campos de un mensaje; si el campo está vacío ya no se mostraba los campos en profundidad para el dapi 3.010. Los clientes solicitan que se continúe mostrando vacío. - Los métodos GetNexField() y GetNextFieldName() continúan retornando vacío cuando se hacía referencia un registro inexistente.
6-Dic-2023	Equipo de Desarrollo	- Se actualiza las versiones del compilador gcc en el documento. Se utiliza gcc 8.3.1 para generar las versiones de 32 y 64 bits para el dapi y el drvapitest.
23-Feb-2024	Equipo de Desarrollo	- Se actualiza documento con el uso de los directorios lib32 y lib64 para las versiones generadas en Linux.
23-Abr-2024	Equipo de Desarrollo	- Se retira comentarios sobre ssl (el dapi ya no usa ssl). Se retira la directiva ssl en el archivo dapi.cfg.sample y de la documentación
10-Jun-2024	Equipo de Desarrollo	- Se implementa el mensaje Comentario Transacción (104) - Revisar la documentación de la estructura del mensaje en el archivo 20240613 - DAPI Interface Message Descriptions.xlsx

16-Oct-2024	Equipo de Desarrollo	- Se implementa los mensajes: Mensaje protos_amortizaciones 330 Mensaje pagos_adicionales 329 - Revisar la documentación de la estructura de los mensajes en el archivo 20241016 - DAPI Interface Message Descriptions.xlsx
29-04-2025	Equipo de Desarrollo	- Se crea el archivo dapi.messages.sample. Se define los nombres de los mensajes que no tienen campos correspondientes a una postura o transacción. El propósito es no preguntar por los nombres de los campos que están en la cabecera del programa ejemplo visual basic.
30-05-2025	Equipo de Desarrollo	- Se aumenta un argumento a la función IsDefined. - Se retira el uso del archivo dapi.messages.sample - En el ejemplo visual basic se aumenta un nuevo comportamiento, ahora se permite visualizar el contenido de los elementos de un arreglo.
25-02-2026	Equipo de Desarrollo	- En el método OnDisconnect se aumenta una explicación para los números de los estados.
23-03-2026	Equipo de Desarrollo	- Se realiza cambios a la función AddLoginConditions porque no estaba procesando las condiciones. - Se implementa un nuevo botón "AddLoginCondGen", en el programa ejemplo Visual Basic, que sirve para leer condiciones desde un archivo de texto plano, llamado dapi_login_cond.cfg, estas condiciones son enviadas al dapi mediante AddLoginConditions
11-09-2026	Álvaro Vela	- Se retira apartados de configuración general que aplican solo para administrador Set-FX

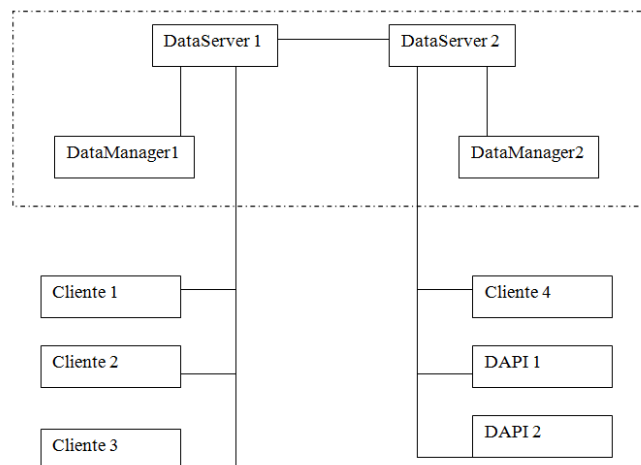
2. Objetivo

Brindar una breve explicación de la Interfaz para Programas de Aplicación de los Sistemas Datatec, conocido como Datatec API o DAPI.

3. Descripción general del Sistema Datatec

El DAPI (Datatec API), 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.



Como se puede notar en la figura anterior, existen varios componentes en la aplicación SMF. Estos pueden ser clasificados a groso modo 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

Data Server: Unidad de intercambio y reenvío de mensajes que integra las conexiones de varios componentes cliente. Se puede usar múltiples DataServers para distribuir la carga de conexiones de clientes.

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.

Estos componentes son usualmente instalados en una locación central.

3.2. Componente cliente

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

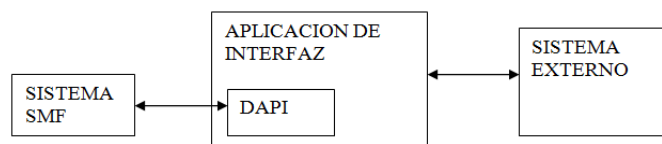
Esta aplicación tiene una interfaz gráfica que permite a los usuarios ver y participar en los mercados electrónicos en tiempo real. Los operadores de mercado ponen sus órdenes (ofertas y/o demandas) en el mercado y estas son observadas por todos los clientes conectados. Luego, otros usuarios pueden negociar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



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 de Windows (OCX): Control compatible con aplicaciones desarrolladas en Microsoft Visual Studio, por ejemplo, Visual Basic, C#, C++, J#, etc.
2. Librería de Windows (LIB): Librería compatible con aplicaciones desarrolladas usando Microsoft Visual Studio usando lenguaje c++. En este caso, el desarrollador de la aplicación que usa esta interfaz debe generar una aplicación tipo **multithread**. Una aplicación multithread permite ejecutar varios procesos simultáneamente.
3. Librería estática de Linux (.a) y dinámica (.so): Se debe generar aplicaciones tipo **multithreaded** en c++.

NOTA: No se utiliza ssl para el dapi. Se recomienda habilitar una conexión privada, vpn, o dssl para garantizar la privacidad de la comunicación.

4.2. Plataformas de instalación

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

- Microsoft Windows 10 - 11 (64 bits)

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

- Linux 8 – 9 (64 bits)
- Windows 11 (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 (*dapi.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()*.

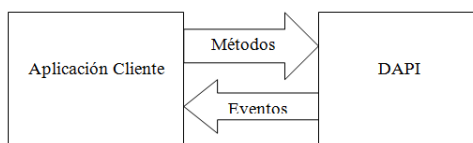
6.2. Pérdidas de conexión programadas

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 o 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



A continuación, se describen los métodos del ActiveX control del DAPI **CdxapiCtrl** y también los métodos de la librería **CdcApi**. Cada método se presenta primero en el 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.

Opcionalmente, se puede definir una operación lógica entre dos condiciones previamente definidas con los siguientes operadores:

Operador	Significado
" "	O LOGICO, operación de O lógico entre dos resultados previamente establecidos por llamar a este método. El resultado de esta operación lógica puede usarse en una siguiente operación lógica según el formato de ejecución BFN.
"&"	Y LOGICO, operación de Y lógico entre dos resultados previamente establecidos por llamar a este método. El resultado de esta operación lógica 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 LIKE de SQL. 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: %: Significa 0 o más letras, cualesquiera sean *: Significa cualquier letra -: Significa cualquier letra [..]: significa cualquiera de las letras colocadas dentro de los []. [^...]: 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. Úsele 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. Úsele 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 administrador del sistema. La dirección IP del equipo servidor del sistema y los restantes parámetros requeridos para establecer la conexión son tomados del archivo *dapi.cfg*.

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, reintentará 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.

Tomar en cuenta que 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, puesto que este tipo de desconexión borra los mensajes recibidos que no hayan sido procesados.

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* (Librería)

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* (Librería)

`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* (Librería)

`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* (Librería)

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 ejecútase 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* (Librería)

`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* (Librería)

`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 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. Úsele el mismo valor recibido en el evento *OnMessageReceived()*.

message_num: Indica el número de mensaje. Úsele 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. Úsele 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 *report_error*)
 Int CdcApi::IsDefined (LPCTSTR *field_name*, int *report_error*)

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.

report_error: Posible valor 0 o 1, por omisión su valor es 1, es decir se reporta error en caso de que el campo no haya sido definido. Si es 0, indica que no se reporta error, se usa únicamente para validar la existencia del campo, ya que desea realizar otra acción.

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 a 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 asíncronos 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 OnMessageReceived

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	Description
1	DCON_ST_OPENING	Opening status. Try to open the connection
2	DCON_ST_OPENCHK	Check opening state. Check if the connection was achieved
4	DCON_ST_NORMCON	Normal connection
5	DCON_ST_CLOSE	Close state. Close connection
6	DCON_ST_TERMIN	Terminate state

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.

Tomar en cuenta que en caso de pérdida inesperada de conexión los mensajes recibidos y no terminados de procesar con el método *FinishMessageProcess()* NO son retransmitidos desde el DAPI a la aplicación de usuario 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, no pudo establecerse satisfactoriamente. 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 con el servidor
20	TCP_STATUS_CLOSED_CONNECTION	Se ha perdido la conexión TCP/IP con el servidor.
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 translation" log	Un archivo de log está siendo retroalimentado y simula una conexión
TCP_STATUS_CLOSED_CONNECTION	"Finalizing translation" log	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 con el servidor
TCP_STATUS_CLOSED_CONNECTION	"Disconnected"	Se ha perdido la conexión TCP/IP con el servidor.
TCP_ERROR_CANT_CONNECT	"Can't connect"	La conexión TCP/IP ha fallado, no pudo establecerse satisfactoriamente. 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 tipo consola de ejemplo en C++ usando la librería del DAPI para Linux y Windows.

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 *OnMessageReceived()* 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()*.
- Construcción de un mensaje para ser enviado al sistema usando los métodos *InitMessage("message_name")* y *SetField()*.
- Construcción de un mensaje para ser enviado al sistema a partir de un mensaje previamente recibido y cambiando algunos campos con los métodos *InitMessage()* y *SetField()*.
- 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);

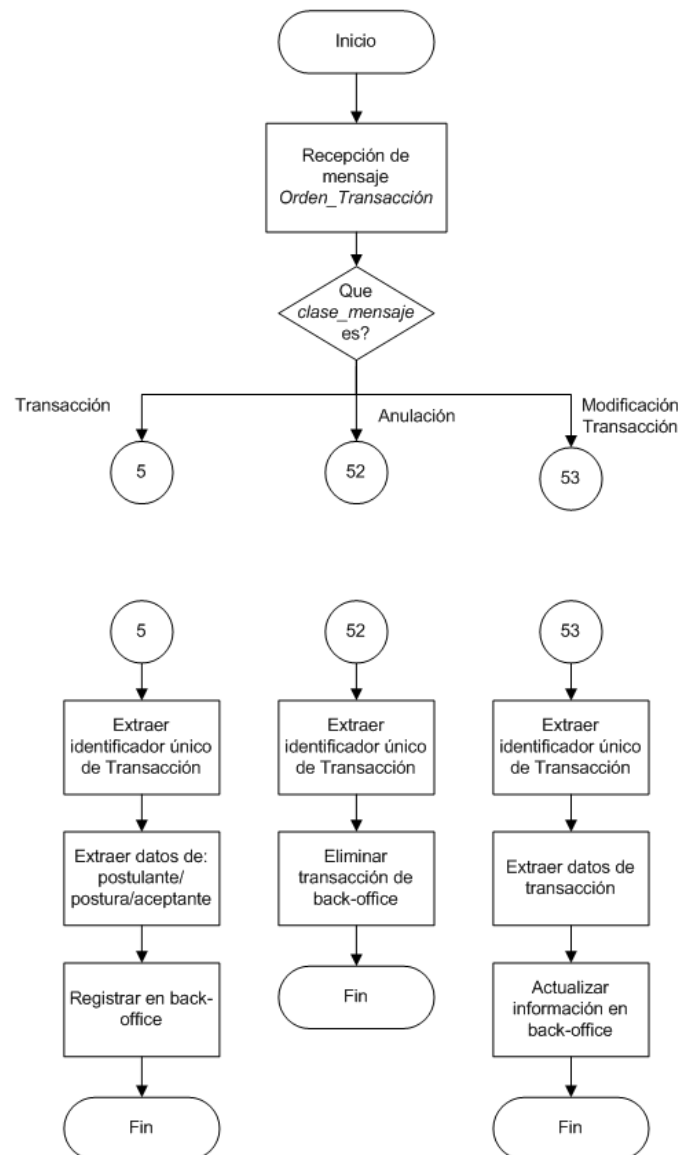
```

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+codigo_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.



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/developar una aplicación que usa el DAPI.

9. Configuración del cliente DAPI (dapi.cfg)

El administrador del sistema debe crear al DAPI como un usuario independiente. Además, debe proveer 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.

```

* dapi.cfg @ D:\CO\DAPI\3022\Bolsa_DAPIWEBB_vb\vbdxapitest\bin64\
File Edit Search AutoText View Help
-----
#
# Datatec Api (DAPI) sample configuration file.
# -----

oper_file_version = "DAPI WEB DAPIWEBB"
data_server       = $03B
ds_ip_address     = 192.168.135.12
ds_tcp_port      = 2016
# PrimaryUser code is the local_code value.
local_code       = 0247
user_code        = 037N
branch_code      = WEBBBOGA
user_name        = DAPIWEBB
password         = datatecl

# If another dataserver is defined to connect the DAPI application
# it is necessary to specify the name, ip and port.
#data_server     = $02N
#ds_ip_address   = 192.168.121.103
#ds_tcp_port     = 2008

# If there were more dataservers in the environment, their names
# must be specified even if the DAPI application is not connected to them.
#data_server     = $03N
#data_server     = $04N

# transaction_feed token is used to get the "Orden_Transaccion" messages (111, 121).
# These messages are got in high/low messages log and it is the most common
# feed used.
transaction_feed = 1

# language options. SPANISH, ENGLISH
#language        = SPANISH

# Delay in milliseconds when DAPI sends messages to Dataserver to prevent
# an involuntary feed that causes problems in the environment.
Send_Msg_Delay_Millisec = 500

# The next directives can be used to read the data from a testing file,
# indicating a range of date/time for the messages to be read, and also the
# the name of the output log file for this test reading.
#testing_file    = dapi_bbv.log
#testing_start   = 20090710091500
#testing_end     = 20090710092000
#testing_output_file = test_dapi_output.log

```

A continuación, se muestra una lista de los campos requeridos en el archivo DAPI.cfg:

Declarativa	Descripción
oper_file_version	Nombre referencial de DAPI.
local_code	Código de PrimaryUser asignado.
user_code	Código de usuario asignado.
branch_code	Código de sucursal.
data_server	Código del equipo DataServer. 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). 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	Dirección ip del equipo DataServer.
ds_tcp_port	Puerto tcp del equipo DataServer.
transaction_feed	0 – No recibir mensajes de transacciones realizadas en el sistema 1 - Recibir los mensajes de las transacciones realizadas en el sistema.
testing_file	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	Especifica la fecha (AAAAMMDD) y hora (hhmmss) de inicio desde la cual el DAPI empieza a leer el archivo testing_file.
testing_end	Especifica la fecha (AAAAMMDD) y hora (hhmmss) en que el DAPI termina la ejecución de la prueba.
testing_output_file	Especifica el nombre del archivo de salida de la prueba.
language	Indica el idioma que se emplea en los tipos de los mensajes que se pasan a la aplicación.

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

- data_server
- ds_ip_address
- ds_tcp_port

Con todos los DataServers 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.

10. Instalación del DAPI en Windows

La distribución del DAPI para Windows contiene los siguientes archivos:

- windows_64\ActiveX\AxdxapiLib.dll.
- windows_64\ActiveX\dxapi.ocx.
- windows_64\ActiveX\Interop.dxapiLib.dll.
- windows_64\lib\dcapi.lib.
- windows_64\lib\dxapi.lib.
- sample_windows\vbdxapitest\ (app ejemplo en VB usando el control OCX para 32 y 64 bits).
- sample_windows\vbdxapitest\bin64 (están los dll, ejecutable y archivos de configuración de ejemplo para 64 bits).

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

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**

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 desde \windows\sysWOW64.

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:

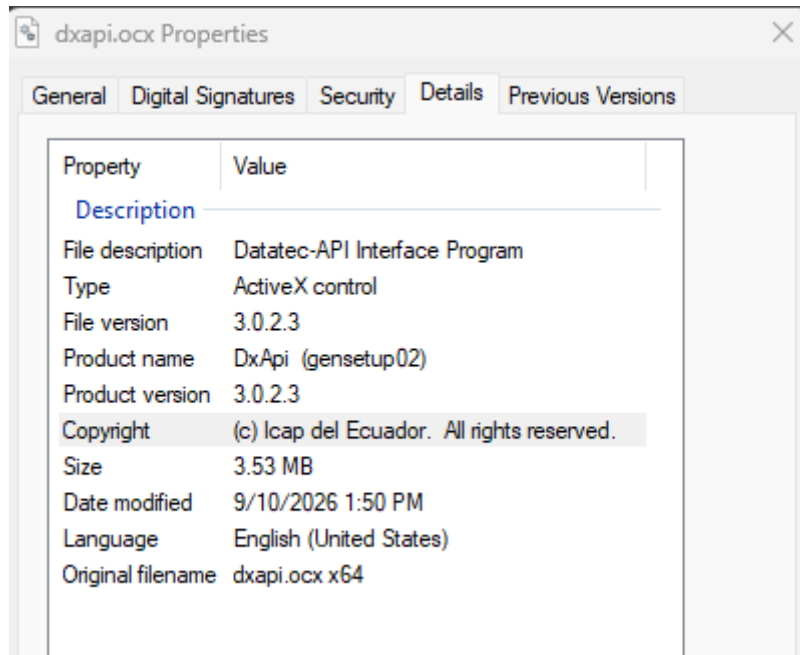
Regsvr32 dxapi.ocx

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

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 (por ejemplo: 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, **clíc-derecho** en el archivo, opción **Propiedades**, escoger la **viñeta Details** en Windows.



10.2. Ejemplo de uso del DAPI en Windows - VbDxApiTest

10.2.1. 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 (véase la sección Instalación del DAPI en Windows).
7. Los ejecutables `vbDxapitest.exe` y `vbDxapitest_64.exe`, se encuentran bajo `vbDxapitest\bin32` y `vbDxapitest\bin64` para 32 y 64 bits respectivamente

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

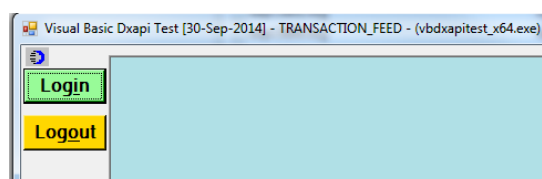
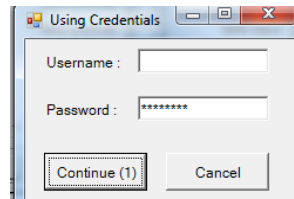
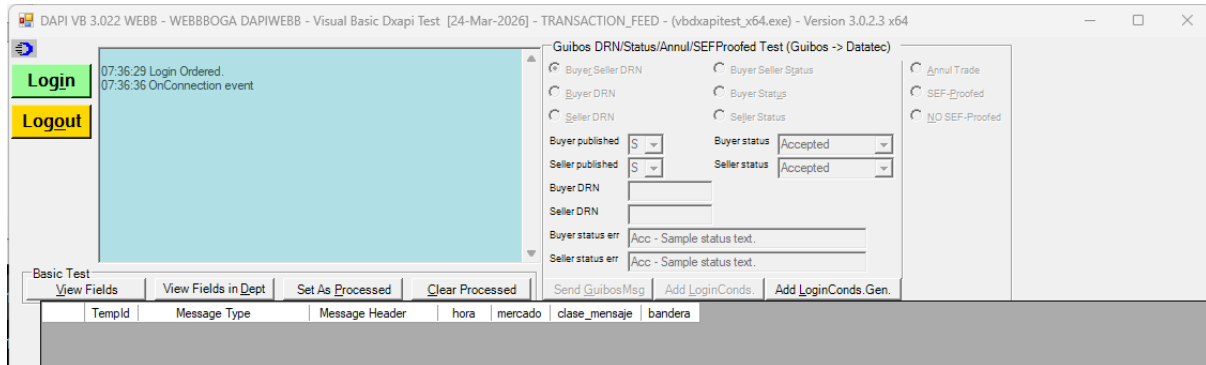


Figura 1: Ventana de ejemplo DAPI windows



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



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*.

10.2.2. 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 código, sino que se delega
    'la ejecución para que lo ejecute el mismo thread de la aplicación
    'de Visual Basic
    TextBox1.Invoke(New Delegate_OnConnection(AddressOf Dlg_Onconnection), _
        New Object() {e})
End Sub

Public Sub Dlg_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#.

10.2.3. 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:

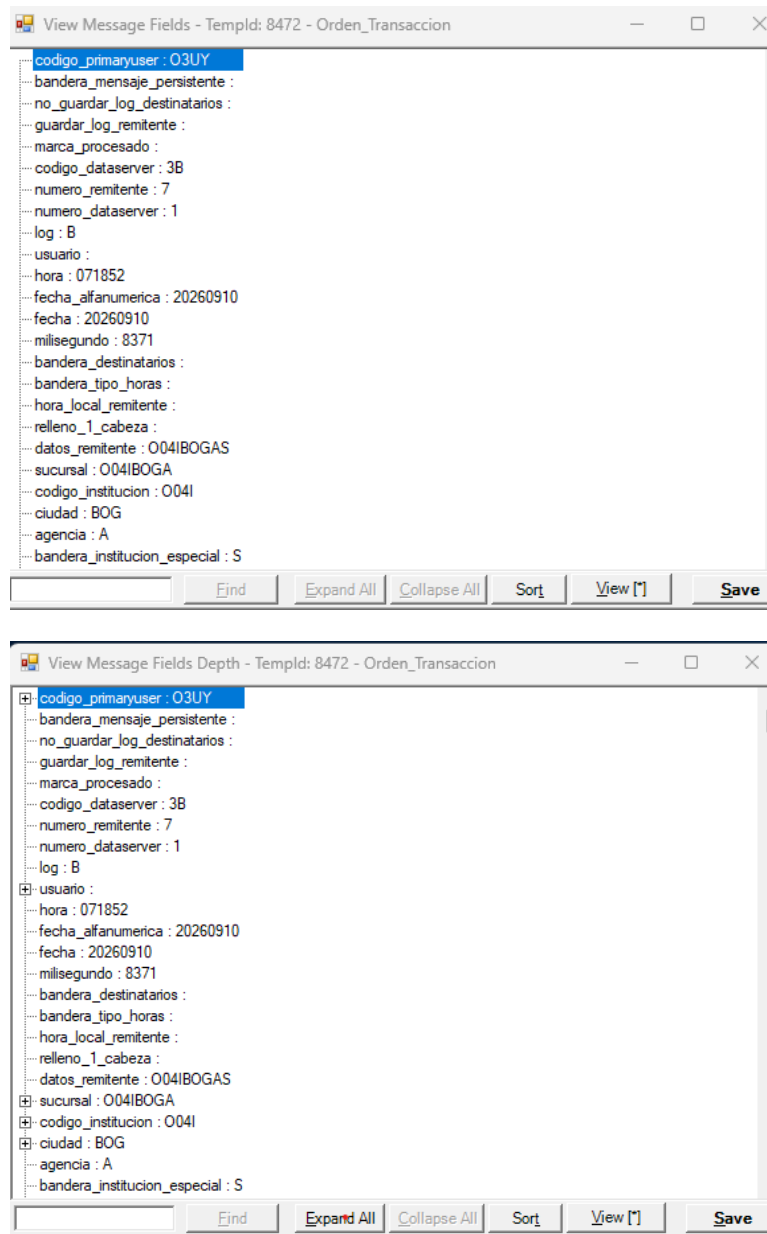
```
Dim fieldvalue As Object
Dim fieldstr As String

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 se pasa a una ventana y aparece como sigue:



10.2.4. View Fields in Depth para Arreglos

En la Ventana "View Fields in Depth", se ha implementado un nuevo comportamiento para mensajes que incluyen **arreglos**. Es decir, se puede mostrar el contenido de un elemento del arreglo, en otra ventana. Es necesario ubicarse con el clic del mouse en el elemento del arreglo que se desea mirar el contenido de sus campos. A continuación, un ejemplo del código a usar:

```
Dim FieldName As String = ""
Dim FieldValue As String = ""

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

'Get the name of element of array
```

```

Dim CurrentNode As TreeNode = Tree.SelectedNode
Dim strTemp As String = Mid(CurrentNode.Text, 1, CurrentNode.Text.IndexOf(":") - 1) & "," &
Mid(CurrentNode.Text, CurrentNode.Text.IndexOf(":") + 3)
str_val = str_val & Mid(strTemp, 1, strTemp.IndexOf(","))

'Verify it the field is an array
If str_val.Contains("[") = False Then Return

'The point is concatenated to the name, to obtain the first field of the array element and its value
FieldName = [String].Concat(str_val, ".")
FieldName = m_frm1.Axdxapi1.GetNextFieldName(FieldName)

Do
'Get FieldValue
FieldValue = m_frm1.Axdxapi1.GetField(FieldName)

'Call subroutine to manage FieldName and FieldValue strings
...

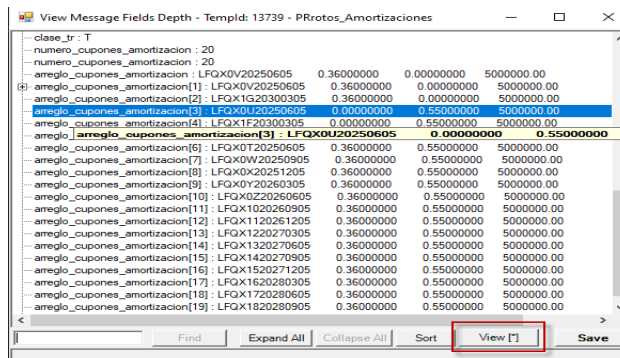
'Retrieve Next FieldName
FieldName = m_frm1.Axdxapi1.GetNextFieldName(FieldName)

Loop Until (FieldName.Length = 0 Or FieldName.Contains(str_val) = False)

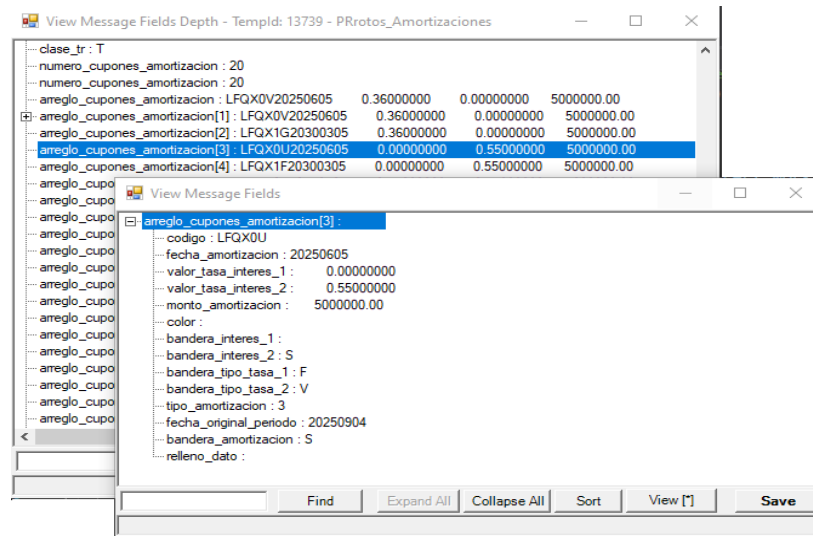
```

Se puede acceder al contenido del elemento del arreglo de 3 maneras:

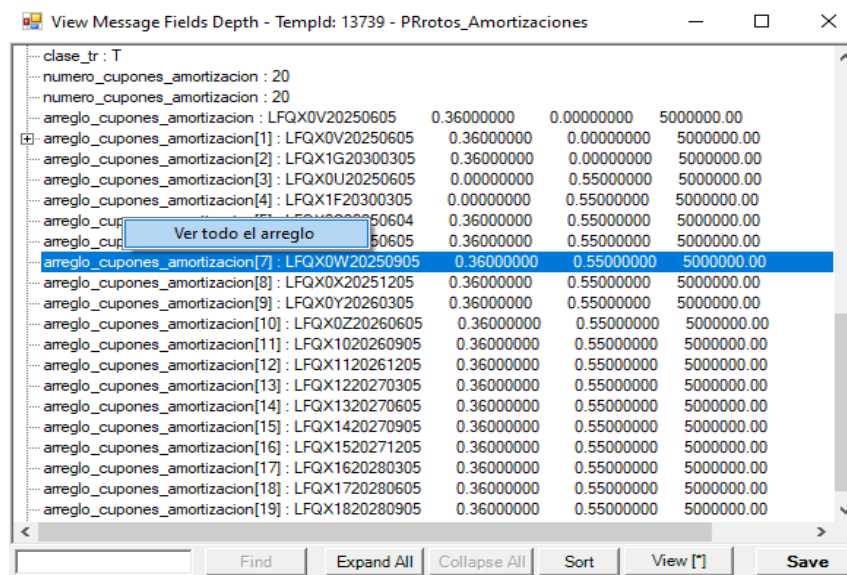
- 1) Se ha aumentado el botón "View[*]", al presionar el clic del mouse, se despliega una ventana con el contenido del elemento del arreglo seleccionado.



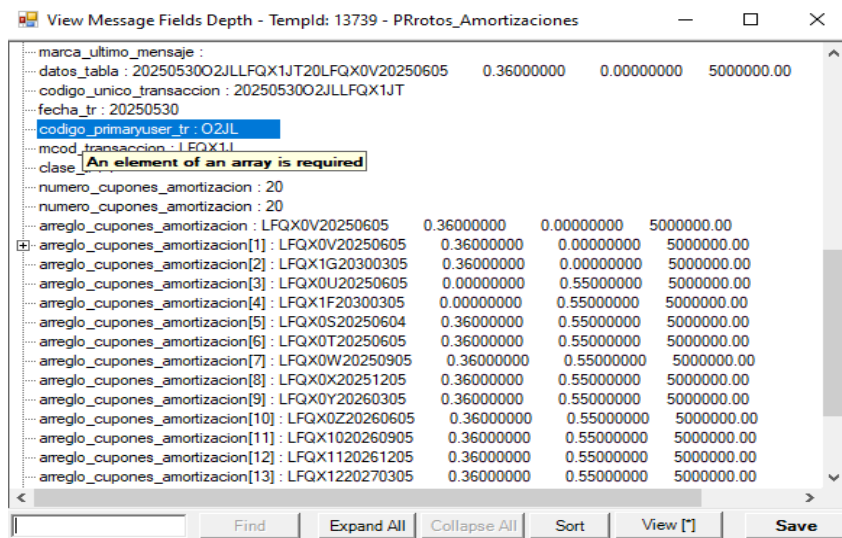
- 2) También se permite mostrar la información con doble clic del mouse



- 3) Al usar clic derecho sobre el elemento del arreglo se despliega un menú. Luego se muestra una ventana como en la figura anterior.



En caso de presionar doble clic o clic derecho sobre un campo que no es arreglo, se despliega un tooltip informativo.



10.2.5. 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.

10.2.6. Botón Clear Processed

El botón **Clear Processed** elimina o borra el registro marcado previamente con el botón “Set As Processed”. Es decir, se borra el registro de la ventana del ejemplo; si se baja y sube el programa ejemplo, el registro ya no se visualiza.

10.2.7. Botón Send Guibos Message

El botón Send Back Guibos Msg, cambió de nombre a **Send Guibos Msg**, utiliza un mensaje recibido de tipo Guibos y modifica una de las opciones seleccionadas (botones de tipo radio) para luego enviarlo de vuelta al sistema con datos de comprador/vendedor/comprador y vendedor, campo DRN o estado para simular lo que se realiza en las aplicaciones de backoffice para mensajes de tipo Guibos.

Esta opción usa los métodos **TakeMessage**, **InitMessage**, **SetField** y **SendMessage**.

Nota: Use esta característica del DAPI con precaución, ya que si los campos del mensaje son llenados incorrectamente al ser enviados al sistema pueden causar efectos indeseables.

10.2.8. 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 solamente 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**.

10.2.9. Botón Add Login Conditions Generic

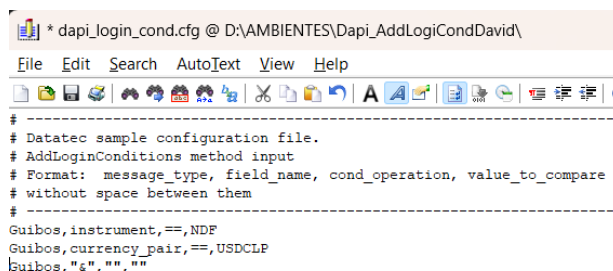
El botón **Add Login Conditions Generic** permite configurar las condiciones que debe cumplir un mensaje recibido para ser o no desplegado en la pantalla principal de la aplicación de ejemplo a través de un archivo plano, llamado **dapi_login_cond.cfg**. Este botón puede ser usado solamente antes de 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**.

Ejemplo del contenido del archivo **dapi_login_cond.cfg**, en una fila se define una condición compuesta por el nombre del mensaje, el nombre del campo, el operador y el valor a comparar; el separador a usar es la coma y no debe existir espacios en blanco entre los campos.

En la imagen se define tres condiciones:

- 1) Guibos,instrument,==,NDF
Se desea desplegar en el ejemplo, todos los mensajes Guibos del instrumento NDF
- 2) Guibos,currency_pair,==,USDCLP
Se desea desplegar en el ejemplo, todos los mensajes Guibos con el par de monedas USDCLP
- 3) Guibos,& ,"" ,""
Condición lógica &, para el ejemplo se va a desplegar los mensajes Guibos que cumplan la primera y segunda condición. Los mensajes que no cumplan las condiciones no son mostrados.



```
*dapi_login_cond.cfg @ D:\AMBIENTES\Dapi_AddLogiCondDavid\
File Edit Search AutoText View Help
# -----
# Datatec sample configuration file.
# AddLoginConditions method input
# Format: message_type, field_name, cond_operation, value_to_compare
# without space between them
# -----
Guibos,instrument,==,NDF
Guibos,currency_pair,==,USDCLP
Guibos,"&","",""
```

10.3. Instalación y uso de la librería DAPI de Windows

En caso de usar la librería del DAPI para Windows, copiar el directorio **sample_windows\drvapitest** al proyecto deseado. En nuevas versiones actualizar las librerías correspondientes al sistema operativo (Windows) y compilador (32/64 bits) desde los directorios **windows*lib** a los subdirectorios lib correspondientes en **drvapitest**

Finalmente ajustar el proceso de mensajes emulando **drvapitest.cpp** de acuerdo con el requerimiento particular. 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**.

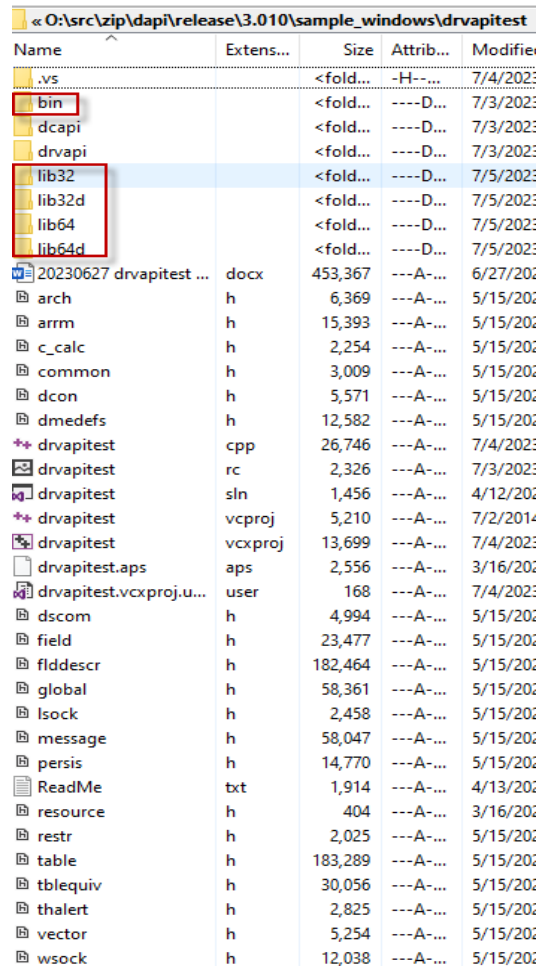
10.4. Ejemplo de uso de la librería DAPI en Windows - Drvapitest.exe

10.4.1. Ejemplo C++ del uso de la librería DAPI

Un programa **drvapitest** que normalmente se usa en Linux, también se lo ha generado para Windows, el programa ejemplo es de tipo consola. El archivo ejecutable se llama **drvapitest.exe**.

Se distribuye en la carpeta `\sample_windows\drvapitest\` de del DAPI. Bajo este directorio se incluyen las librerías para compilar y ejecutar las versiones *debug* y *release* en 32 o 64 bits. (lib32d, lib64d, lib32 y lib64). También se incluye el directorio `\sample_windows\drvapitest\bin` que contiene los archivos de configuración.

El contenido del directorio `drvapitest` es:



Name	Extens...	Size	Attrib...	Modifie
.vs		<fold...	-H-...	7/4/202:
bin		<fold...	----	7/3/202:
dcapi		<fold...	----	7/3/202:
drvapi		<fold...	----	7/3/202:
lib32		<fold...	----	7/5/202:
lib32d		<fold...	----	7/5/202:
lib64		<fold...	----	7/5/202:
lib64d		<fold...	----	7/5/202:
20230627 drvapitest ...	docx	453,367	---A-...	6/27/20:
arch	h	6,369	---A-...	5/15/20:
arrm	h	15,393	---A-...	5/15/20:
c_calc	h	2,254	---A-...	5/15/20:
common	h	3,009	---A-...	5/15/20:
dcon	h	5,571	---A-...	5/15/20:
dmedefs	h	12,582	---A-...	5/15/20:
drvapitest	cpp	26,746	---A-...	7/4/202:
drvapitest	rc	2,326	---A-...	7/3/202:
drvapitest	sln	1,456	---A-...	4/12/20:
drvapitest	vcproj	5,210	---A-...	7/2/201:
drvapitest	vcxproj	13,699	---A-...	7/4/202:
drvapitest.aps	aps	2,556	---A-...	3/16/20:
drvapitest.vcxproj.u...	user	168	---A-...	7/4/202:
dscom	h	4,994	---A-...	5/15/20:
field	h	23,477	---A-...	5/15/20:
flddescr	h	182,464	---A-...	5/15/20:
global	h	58,361	---A-...	5/15/20:
lsock	h	2,458	---A-...	5/15/20:
message	h	58,047	---A-...	5/15/20:
persis	h	14,770	---A-...	5/15/20:
ReadMe	txt	1,914	---A-...	4/13/20:
resource	h	404	---A-...	3/16/20:
restr	h	2,025	---A-...	5/15/20:
table	h	183,289	---A-...	5/15/20:
tblequiv	h	30,056	---A-...	5/15/20:
thalert	h	2,825	---A-...	5/15/20:
vector	h	5,254	---A-...	5/15/20:
wsock	h	12,038	---A-...	5/15/20:

En los directorios `lib32` y `lib32d`, están las librerías `dcapi.lib` y `crypt.lib` *release* y *debug* respectivamente, compiladas para 32 bits.

En los directorios `lib64` y `lib64d`, están las librerías `dcapi.lib` y `crypt.lib` *release* y *debug* respectivamente, compiladas para 64 bits.

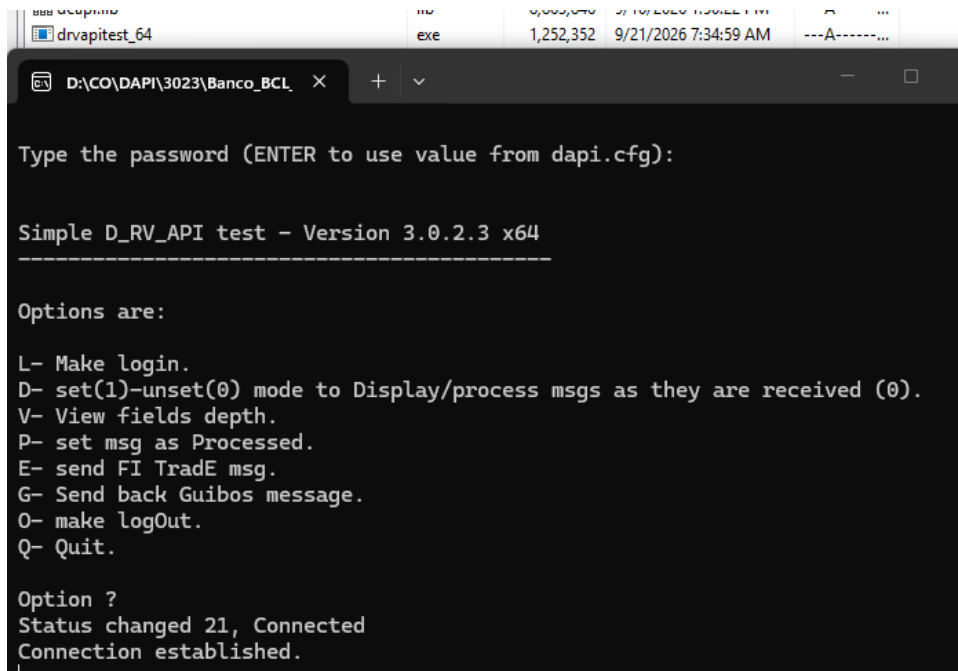
Con nuevas versiones, las librerías `dcapi.lib` y `crypt.lib` deben copiarse en `lib*`, según la plataforma requerida. Si desea regenerar el ejecutable abrir el archivo solución (.sln) usando Visual Studio (2019) y realice el proceso para la generación del ejecutable en configuración y plataforma requeridos. Si usa nuevas versiones de Visual Studio (por ejemplo 2022), al abrir el archivo .sln se preguntará si desea migrar la plataforma a la del VS, seleccione la que corresponda a fin de que pueda compilar el proyecto.

Para una prueba inicial copie el contenido del directorio `drvapitest` en el equipo donde desea correr el ejemplo y siga estos 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 `drvapitest/lib32`, `drvapitest/lib64`, para ejecutar versiones `release` ó `drvapitest/lib32d`, `drvapitest/lib64d`, para versiones `debug`.
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 `drvapitest/lib*`.
5. Ejecute el programa.

Para ejecutar el `drvapitest.exe` o `drvapitest_x64.exe`, es suficiente con presionar doble clic del mouse, la ventana resultado es:



```

Type the password (ENTER to use value from dapi.cfg):

Simple D_RV_API test - Version 3.0.2.3 x64
-----

Options are:

L- Make login.
D- set(1)-unset(0) mode to Display/process msgs as they are received (0).
V- View fields depth.
P- set msg as Processed.
E- send FI Trade msg.
G- Send back Guibos message.
O- make logOut.
Q- Quit.

Option ?
Status changed 21, Connected
Connection established.

```

Las opciones listadas son las mismas que el programa `drvapitest Linux`.

11. Instalación del DAPI en Linux

La distribución para Linux contiene los siguientes archivos:

- `linux_64\lib64\libdapi.a`
- `linux_64\lib64\libdapi.so`
- `libstdc++.so.6`
- `sample_linux\linux7_64\lnxdrvapitest.zip` (librería estática `.a` y código de ejemplo desarrollado en C++).
- `sample_linux\linux7_64\lnxdrvapitest_so.zip` (librería dinámica `.so` 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 64 bits en equipo Linux de 64 bits:

- Red Hat Enterprise Linux Server release 8 y 9 para 64 bits.
- **gcc (GCC) 8.5.0 en Linux 8**
- **gcc (GCC) 10.5.0 en Linux 9**

11.2. Instalación y uso de la librería DAPI en Linux

Luego de seleccionar la plataforma para el Dapi y desempaquetar el archivo `Inxdrvapitest.zip`; se obtiene el directorio **Inxdrvapitest** 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: **Inxdrvapitest /libdapi/lib/libdapi.a**

El archivo header de la librería DAPI es `dcapi.h` y se encuentra en: **Inxdrvapitest/dcapi/dcapi.h**

Los mismos pasos descritos anteriormente se aplica para el archivo **Inxdrvapitest_so**, en este caso se utiliza la librería dinámica **libdapi.so**

11.3. Contenido del directorio de la aplicación de ejemplo y de la librería

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

... xdebug64/Inxdrvapitest			
*n	Name	Size	Modif
	/..	UP--DIR	Jul
	/bin	4096	Jul
	/dcapi	4096	Jul
	/drvapi	4096	Jul
	/drvapitest	4096	Jul
	/libdapi	4096	Jul
	/scripts	4096	Jul
	*arch.h	6369	Jul
	*arm.h	15393	Jul
	*c_calc.h	2254	Jul
	*common.h	3009	Jul
	*dcon.h	5571	Jul
	*dmedefs.h	12582	Jul
	*dcom.h	4994	Jul
	*field.h	23477	Jul
	*flldescr.h	182464	Jul
	*global.h	58361	Jul
	*lsock.h	2458	Jul
	*message.h	58047	Jul
	*persis.h	14770	Jul
	*restr.h	2025	Jul
	*table.h	183289	Jul
	*tblequiv.h	30056	Jul
	*thalert.h	2825	Jul
	*vector.h	5254	Jul
	*wsock.h	12038	Jul

11.4. Compilación de la librería DAPI en Linux

NO se requiere compilar la librería, ya se incluye el compilado de la misma. Los archivos fuentes ya no son distribuidos, solo los archivos .h necesarios (en *sample_linux*)

Los archivos `libdapi.a`, `libdapi.so` y `drvapitest` están compilados para 32 bits en equipo de 64 bits.
Los archivos `libdapi.a`, `libdapi.so` y `drvapitest` están compilados para 64 bits en equipo de 64 bits.

11.5. Compilación de la aplicación ejemplo en Linux (drvapitest)

La distribución del DAPI para Linux incluye una aplicación de ejemplo del uso del DAPI y la librería estática **libdapi.a** y la librería dinámica **libdapi.so** programadas en C++, e incluyen todas sus librerías de dependencias.

El programa **drvapitest** que usa la librería estática **.a**, se distribuye ya compilado y se encuentra en: **/Inxdrvapitest/drvapitest/**

El programa **drvapitest** que usa la librería dinámica **.so**, se distribuye ya compilado y se encuentra en: **/Inxdrvapitest_so/drvapitest/**

Usar la librería dinámica tiene la ventaja de que, para las siguientes versiones, no se requiere recompilar su aplicativo (excepto para el uso de nuevas funcionalidades), solo se necesita copiar la librería **libdapi.so** al directorio de ejecución.

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

```
[myuser@workdir] $ .cd Inxdrvapitest/scripts
[myuser@scripts] $ ./mkrel 64 drvapitest clean
[myuser@scripts] $ ./mkrel 64 drvapitest

[myuser@workdir] $ .cd Inxdrvapitest_so/scripts
[myuser@scripts] $ ./mkrel drvapitest_so clean
[myuser@scripts] $ ./mkrel drvapitest_so
```

NOTA: Este script compila nuevamente el ejecutable **drvapitest**, mas no compila la librería **libdapi.a/libdapi.so**, ya que no se distribuyen los archivos fuentes, el ejecutable del ejemplo **drvapitest** se generan en: **Inxdrvapitest/drvapitest/xrelease**, para la librería estática y para la dinámica se generan bajo **Inxdrvapitest_so/drvapitest/xrelease**.

Cabe recalcar que varios de los procesos para generar las librerías del Dapi (**libdapi.a**, **libdapi.so**) 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 mkrel
```

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

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

- | | | |
|-------------------|-----------------|-------------|
| • linux-vdso.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 |
| • libstdc++.so.6 | | |

11.6. 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.o
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
```

Para instalar las librerías de 32/64 bits se debe utilizar el RPM glibc-devel-2.17-78.el7.i686.rpm. 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.

```
[datatec@wdf12 ~]$ rpm -Uvh glibc-devel-2.17-78.el7.i686.rpm
```

Con las librerías instaladas se puede proceder con la compilación del ejemplo, pero, si se utiliza el archivo libdapi.a compilado con GCC anterior 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.o
/home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/libdapi/xrelease/libdapi.a(dcap.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(dcap.o): In function 'std::simple_alloc<std::Rb_tree_node<std::pair<CFixedIncomeMarketsKey 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(dcap.o): In function 'std::simple_alloc<std::Rb_tree_node<std::pair<CMarketsKey const, int>>, std::default_alloc_template<true, 0>>::deallocate(std::Rb_tree_node<std::pair<CMarketsKey const, int>>*, unsigned int)':
```

Si no se tiene instalado el g++ a más del gcc, se muestra el siguiente error:

```
[datatec@support70 scripts]$ rpm -q g++
package g++ is not installed
[datatec@support70 scripts]$
```

```
[datatec@support70 scripts]$ ./mk 64 drvapitest
Generating ...
APP NAME:      drvapitest
OS:            linux-gnu
RHEL VERSION:  Red Hat Enterprise Linux Server release 7.5 (Maipo)
GCC VERSION:   gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-28)
Copyright (C) 2015 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/sitest/d_3009_test/lnxdrvapitest/drvapitest/xdebug64/drvapi.o
-> /home/sitest/d_3009_test/lnxdrvapitest/drvapitest/xdebug64/drvapitest
Making Completed
[datatec@support70 scripts]$
```

Al cambiar la librería por la que es compilada con GCC 8.3 la compilación es exitosa. A continuación, el resultado de la compilación.

```
[datatec@wdfl2 scripts]$ ./mkrel 64 drvapitest
Generating ...
-----
CONFIGURATION: Release 64 bits
APP NAME:      drvapitest
OS:            linux-gnu
-----
RHEL VERSION:  Red Hat Enterprise Linux Server release 7.7 (Maipo)
GCC VERSION:   gcc (GCC) 8.3.1 20190311 (Red Hat 8.3.1-3)
Copyright (C) 2018 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.
-----
PLEASE, USE THIS FILE TO COMPILE...
-rwxrwxr-x+  1 datatec datatec   4190 Oct  5 16:51 Makedrvapitest
-rwxrwxr-x+  1 datatec datatec   4454 Oct  4 16:47 Makedrvapitest_so
-> /home/eacuna/dme/drvapitest/xrelease64/drvapi.o
-> /home/eacuna/dme/drvapitest/xrelease64/drvapitest
Making Completed
```

11.7. Ejemplo de uso del DAPI en Linux

La distribución del DAPI para Linux (de 32/64 bits) cuenta con un ejemplo de la utilización de la librería **libdapi.a/libdapi.so** 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
Inxdrvapitest/drvapitest/xrelease64

Inxdrvapitest_so/drvapitest/xrelease64

Deberá entonces ubicarse en el directorio y ejecutar el archivo correspondiente de esta manera:

```
[myuser@ xrelease32]$ ./drvapitest
```

```
Simple D_RV_API test
Options are:

L- Make login.
D- set(1)-unset(0) mode to Display/process msgs as they are received (0).
V- View fields depth.
P- set msg as Processed.
E- send FI Trade msg.
G- Send back Guibos message.
O- make logOut.
Q- Quit.

Option ?
Status changed 21, Connected
Connection established.
```

12. Listado de códigos de error

Código	Identificación de error
1	DAPI_ERR_INTERNAL_ERROR 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 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 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 Se ha requerido la ejecución sobre algún método, pero no se ha llamado al método de <i>login</i> primeramente.
5	DAPI_ERR_REJECTED_CONNECTION 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)* 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)* 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)* 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)* 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)* 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)* 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)* El servidor de Datatec rechazó el intento de conexión porque la contraseña correspondiente a la sucursal es incorrecta.
13	DAPI_ERR_STATUS_NOT_EXPECTED El estado de la conexión ha cambiado a un estado indefinido. Reporte este resultado a su operador.
14	DAPI_ERR_IMPROPER_OPERATION 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 El nombre de registro dado como argumento al método es inapropiado.
16	DAPI_ERR_ILLEGAL_FIELD_NAME_SPECIFIED El nombre de campo que se dio como argumento al método no existe.
17	DAPI_ERR_ALREADY_INITIALIZED Ocurre cuando se ejecuta nuevamente el método <i>Login</i> .
18	DAPI_ERR_RECORD_NOT_FOUND 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 <i>FinishMessageProcess</i> .
19	DAPI_ERR_CONDITION_EVALUATION

	El DAPI no pudo evaluar las condiciones de recepción de mensajes dadas en el método <i>AddLoginCondition</i> , 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.