



DATATEC API – DAPI Developer's Guide

Datatec Financial Market System

Update 2.034

Primary audience:

Designers and developers of an interface to the Datatec API

20-Nov-2020
ICAP del Ecuador

Table of Contents

1. Document history	5
2. Objective	7
3. General description of the Datatec system	7
3.1 Server components.....	8
3.2 Client components	8
3.3 Datatec Application Programming Interface (DAPI)	8
4. DAPI platforms	9
4.1 Development platforms.....	9
4.2 Installation platforms	9
4.3 Minimum DAPI hardware requirements	9
5. How to interact with the DAPI	9
6. Verification of transmission / reception of messages	10
6.1 Unexpected connection loss.....	10
6.2 Expected connection loss	11
7 Methods and events in the DAPI	11
7.1 Methods in the DAPI	11
7.1.1 AddLoginCondition method	11
7.1.2 FinishMessageProcess method	13
7.1.3 Login method	13
7.1.4 Logout method	14
7.1.5 GetNextFieldName method (ActiveX)	14
7.1.6 GetNextField method (Library)	15
7.1.7 GetField method (ActiveX)	16
7.1.8 GetField method (Library)	16
7.1.9 GetFieldBSTR method (Library)	16
7.1.10 GetFieldDATE method (Library)	17
7.1.11 GetFieldLONG method (Library)	17
7.1.12 GetFieldULONG method (Library)	17
7.1.13 InitMessage method	18
7.1.14 ProcessMyEvents method (Library)	18
7.1.15 SendMessage method.....	18
7.1.16 SendMessageTo method	19
7.1.17 SetField(ActiveX) and SetFieldBSTR(Library) methods.....	20
7.1.18 SetFieldDATE method (Library)	20
7.1.19 SetFieldLONG method (Library)	21
7.1.20 SetFieldULONG method (Library).....	21
7.1.21 TakeMessage method.....	21
7.1.22 Método IsDefined	22
7.1.23 Método GetCompilationType	22
7.1.24 Método GetVersion.....	23
7.2 Events available in the DAPI.....	23
7.2.1 Events in the DAPI ActiveX control (OCX)	23
7.2.2 Events in the DAPI(Library) library	23
7.2.3 OnMessageReceived event	23

7.2.4	OnConnection event.....	24
7.2.5	OnDisconnection event.....	24
7.2.6	OnError event.....	25
7.2.7	OnStatusChanged event.....	25
7.2.8	OnDataReceived event.....	26
8	DAPI code examples	27
8.1	Example of how to process a message in Visual Basic	27
8.2	Example of how to send a message using C++	28
8.3	Sample workflow process for gathering transaction information for back-office applications	28
9	Configuration of the DAPI in the DMclient	30
10	Configuration of the DAPI in the client (dapi.cfg)	31
10.1	Temporal Configuration File (dapi_tmp_previous_date_log_request.txt)	36
11	Installation of the DAPI in Windows	36
11.1	Installation and use of the Windows ActiveX control (OCX).....	36
11.1.1	Check the DAPI version in the OCX File.....	37
11.2	Installation and use of the Windows library	38
11.3	VbdxApiTest - Windows example of the use of DAPI	38
11.3.1	Note for 64-bit Windows machines running Visual Basic example.....	38
11.3.2	Running Visual Basic example.....	39
11.3.3	Login and Logout buttons	40
11.3.4	View Fields and View Fields in Depth buttons	40
11.3.5	Set As Processed button.....	42
12	DAPI Linux installation	42
12.1	Default configuration under Linux	42
12.2	OpenSSL.....	42
12.3	Installation and use of the Linux version of DAPI	43
12.4	Building the library in Linux	43
12.5	Building the Linux sample (drvapitest)	43
12.6	Possible error messages during the compilation process	44
12.7	drvapitest - Linux example using the DAPI	45
13	Message descriptions	46
14	Error Code List	47

Table of Figures

Figure 1: General overview of the architecture	7
Figure 2 DAPI interface with in-house system.....	8
Figure 3 Interaction scheme between the wrapper application and the DAPI.....	11
Figure 4 Flowchart depicting the recommended processing of an <i>Orden_Transaccion</i> message for back-office logging.....	29
Figure 5 Select the branch and right-click to view the menu	30
Figure 6 Addition of a new PU assigned to the branch of the DAPI user	30
Figure 7 Properties of the new DAPI user.....	30
Figure 8 User passwords option in the DataManager Client	31
Figure 9 DAPI.CFG File.....	31
Figure 10 Dapi_tmp_previous_date_log.....	36
Figure 11 Dxapi.ocx properties.....	37
Figure 12 Microsoft Visual C++ redistributable package installed on the client machine.....	38
Figure 13 VbdxApi Test Application Main Window prior to login	39
Figure 14 VbdxApi Test Credentials window (can be ignored).	39
Figure 15 VbdxApi Test Application Main Window	39
Figure 16 View Fields Button Window	41
Figure 17 View Fields in Depth Button Window	42
Figure 18 Dapi Linux structure.....	43
Figure 19 Sample error message when gcc libraries are not found	44
Figure 20 Using yum to install compatibility libraries	44
Figure 21 Sample error message when wrong libdapi.a version is used	45
Figure 22 Successful compilation of the sample program.....	45
Figure 23 Execution of the sample program.....	46

1. Document history

Date	Description
18-Sep-2007	Initial documentation
20-Apr-2009	Review for version 1.056
08-Jan-2010	<ul style="list-style-type: none">- Review for changes in version 1.061- New guibos_feed config parameter for Guibos Messages.- New Send_Msg_Delay_Millisec config parameter.- Update of configuration default values (high/low msg log asking).- Update of other config parameters explanation.- Windows VBasic sample changes for Guibos messages and to send back to Datatec System certain messages.- Linux compilation library/sample notes updated.- New mk script.
28-Jun-2010	<ul style="list-style-type: none">- General review of the document and update of English/Spanish versions of it, to use the same format.- Notes added for recompilation (gcc versions and support for 32-64bit operating systems)- Note support Windows Vista (32bits) y 7 (32bits)- DAPI 1.065. General adjustments to use DAPI instead of DGAPI in environments that receive Guibos messages.- DAPI 1.066. It sends its version at connection time, so DAPI version is displayed in versions window of DMClient.- DAPI 1.067. Use of OpenSSL 0.98j instead of 0.9.7a.
20-Sep-2010	<ul style="list-style-type: none">- Section 8.3 added for sample workflow process for gathering transaction information for back-office- Section 9 added for configuration of DAPI in the DMClient- DAPI 1.069. Support new message Mantenimiento_Transaccion_Renta_Fija. File "20100920 DAPI Interface Message Descriptions.xls"
28-Oct-2010	<ul style="list-style-type: none">- General review.
30-Jun-2011	<ul style="list-style-type: none">- Instructions to run example in Windows 64 bits platforms (section 11.3.1). Rebuild of windows sample (v. 1.077).- DAPI version checking in OCX file (section 11.1.1).

22-Dec-2017	<ul style="list-style-type: none"> - Document update in the following sections: - Host requirements for Development and run-time platforms. - Windows DAPI installation. - Linux DAPI installation. - Linux OpenSSL configuration. - DAPI library installation in Linux. - Compilation of the DAPI library in Linux. - Compilation of the DAPI example in Linux. - Sample execution of the DAPI in Linux.
14-Nov-2019	<ul style="list-style-type: none"> - English version update.
21-Nov-2019	<ul style="list-style-type: none"> - Implementation of using a new temporary configuration file, used to start with a complete retransmissions of messages of one day in the past. Directives log_days_high_messages and log_days_low_messages in the dapi.cfg file are removed.
25-06-2020	<ul style="list-style-type: none"> - Implementation new functions: - GetCompilationType - GetVersion - Function documentation IsDefined. - Generation of VisualBasic sample with VisualStudio 2013 for platforms x86 and x64.
22-09-2020	<ul style="list-style-type: none"> - General review. OpenSSL 1.1.1b.
24-09-2020	<ul style="list-style-type: none"> - In the login message 201, a lowercase “d” is assigned in the bandera_tipo_primaryuser field, to indicate that it is a Dapi client.
29-09-2020	<ul style="list-style-type: none"> - For the Spot or Next Day Colombia markets, the codigo_externo field is not set to zero.
12-10-2020	<ul style="list-style-type: none"> - 2.033 Changes. New static (.a) and dynamic (.so) libraries. - Linux 5 (6.8) and Linux 7 examples available.
20-11-2020	<ul style="list-style-type: none"> - General review.

2. Objective

The objective of this document is to give a general overview of the Datatec API (DAPI), its installation, configuration and files distributed. It is also included a Reference Guide for API programmers and the description for samples provided in the supported platforms (Linux/Windows). Additionally, it shows the installation and configuration steps of DAPI applications for Windows and Linux platforms.

3. General description of the Datatec system

The DAPI (Datatec API), as the name suggests, is an Application Programming Interface that allows the Datatec application platform to send and receive data to and from external systems. The application that the DAPI allows external systems to interface with is the Electronic Financial Markets system (known as SMF for its initials in Spanish) developed by Datatec. The SMF is a distributed platform for trading of financial instruments.

The normal trading operations are performed via messaging in a client-server environment. SMF uses the TCP/IP protocol for sending messages, and therefore the delivery of each message to its destination is assured. The following diagram illustrates how the system is spread over several physical locations and how each of the components interacts with each other.

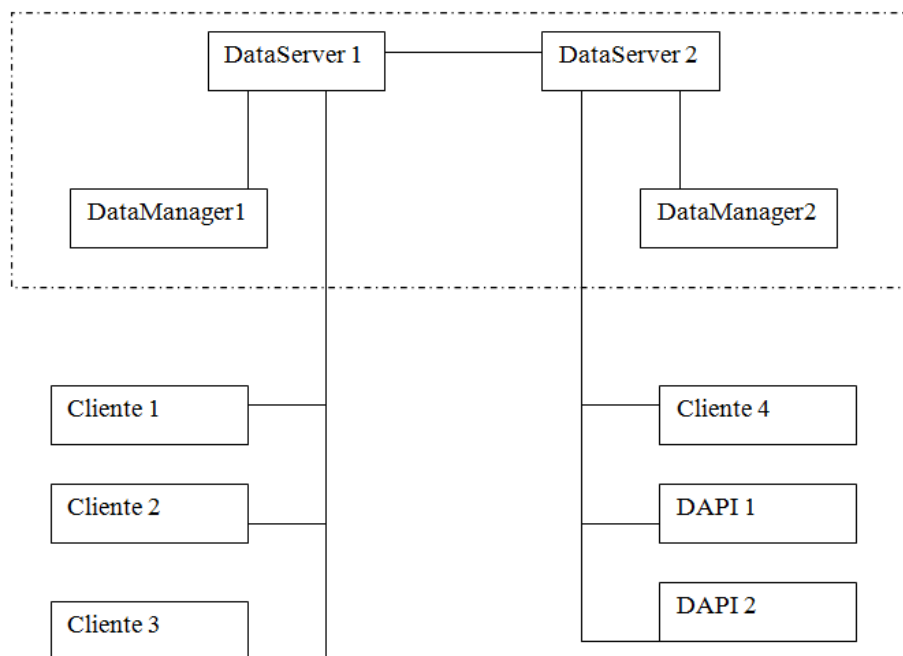


Figure 1: General overview of the architecture

As you will note in the above figure, there are several components in the SMF application. They are broadly classified as Client and Server components.

To develop and test the interface with the DAPI, it is essential to understand on a general level where the DAPI fits into the SMF application and with whom it interacts. The following is a general explanation of the architecture of SMF.

3.1 Server components

DataServer. It is one of the server components. It acts as the message exchange and forwarding unit that integrates connections from various client components. Multiple DataServers can be used to distribute the load of client connections as shown in Figure 1. Loosely, DataServers can be compared to routers, because they maintain connections and forward messages.

DataManager Server. It is the other server component. This application is responsible for user logon validation, user profile maintenance, master file maintenance, calculation of market statistics, and other related functions.

Matching Engine. It is a other server component that is in charge of maintaining trade credits limits and process orders, trades and trade statistics.

These components are usually setup in a central location.

3.2 Client components

All end users of the SMF system connect to a DataServer via a client application.

This client application has a graphical user interface that allows users to view and participate in the market in real time. Market operators place orders (offers or bids) into the markets that are then viewed by the entire market. Other users can then trade these bids and offers.

The client application communicates with the system via a number of different messages. These messages can be sent from the client to the system, or can be received by the client from the system.

3.3 Datatec Application Programming Interface (DAPI)

The DAPI is a particular kind of client whose function slightly varies from a normal client. The DAPI does not have a GUI, but nonetheless can send and receive the messages that a normal client would use. It is important to remember that the DataServer considers the DAPI as just another client.

Anyone that intends to use the functionality of the DAPI to interact with the SMF system should create an application that acts as a wrapper of the DAPI and makes appropriate function calls to use the DAPI. The details of these functions are explained further down in this document. On the other side, this wrapper application should also interact with the appropriate in-house system as shown in the following diagram:

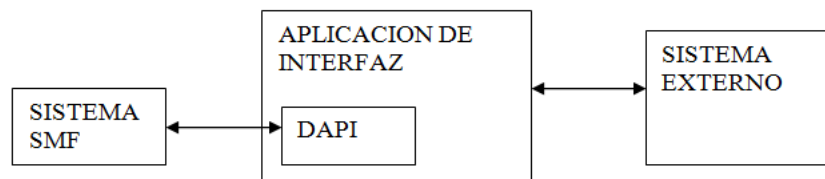


Figure 2 DAPI interface with in-house system.

Whenever a change occurs in the market (e.g.: an order input, a withdrawal of an order, modification of an order, etc), the SMF system sends a message to the DAPI, which can be read by the wrapper application and provided to the in-house application.

As in any connection-oriented system, SMF relies on the existence of connection between the DataServer and the client components including the DAPI. The DAPI uses its own form of Transmission Recovery Protocol in the event that the connection is lost. Once a message is received by the DAPI, it is placed in a message queue to be processed by the wrapper application. The DAPI announces the arrival of a message via the triggering of an **Event**. The wrapper application must be programmed to interpret these events, collect messages of interest using **Methods** provided by the DAPI, and then transmit the appropriate information on to the in-house system. There are also methods available in the DAPI which the in-house system can use to send messages out to the SMF system.

4. DAPI platforms

4.1 Development platforms

The DAPI is available for the following platforms:

1. Windows ActiveX Control (OCX).
2. Windows Library (LIB). In this case, the developer of the interface application has to generate a **multithreaded** application.
3. Linux 5 (6.8) and Linux 7 static and dynamic libraries. Libdapi.a and Libdapi.so.

4.2 Installation platforms

The DAPI, in its OCX form and/or windows library, can be run on the following operating systems:

- Windows 10/Server (32/64 bits)

As a static and/or dynamic library, it can be used in the following operating systems:

- Linux 5, 6.8 and 7

4.3 Minimum DAPI hardware requirements

- Processor : Dual core or higher.
- Memory : 4GB or more.
- Hard Disk : At least 100 GB available space.

5. How to interact with the DAPI

The following is the logical sequence of DAPI methods to be called by the in-house application:

LOGIN: The first method that needs to be called is the *Login()*. The login method requires the username and password to be sent as parameters. This method initiates various sub-processes that maintain an asynchronous connection with the DataServer. Any error during the connection process will be reported by the *OnError()* event handler. Please refer to the error list further below in this document. The *OnConnection()* event will report the result of each login attempt. On successful connection, the *login()* will return **DAPI_INITIALIZED** and the *OnConnection()* will return 0.

MESSAGE RECEPTION: On reception of each message by the DAPI from the SMF system, the *OnMessageReceived()* event will be triggered.

ACCESSING THE MESSAGE RECEIVED: To access the message received, the *TakeMessage()* method must be used. Individual fields of the message can be accessed using the *GetField()* method.

MARKING THE MESSAGE AS PROCESSED: Once the message has been read successfully, to ensure that the DataServer does not retransmit the same message on reconnection after a connection loss, the *FinishMessageProcess()* should be called. It is important to note that in case of an unexpected loss of connection, the SMF does NOT retransmit a message that it already transmitted and whose reception has been confirmed by the DAPI.

SENDING MESSAGES TO THE DATASERVER: To prepare a message to be sent to the DataServer by the DAPI, the message has to be initialized by calling *InitMessage()*. On successful initialization, the fields can be set by using *SetField()*. If it were necessary, specify the message targets with method *SendMessageTo()*. Once the message has been populated, *SendMessage()* will send the message to the DataServer. If the *InitMessage()* method is called without any parameters, it will initialize the last message handled by *TakeMessage()*.

LOGOUT: *Logout()* will produce a normal disconnection of the DAPI from the DataServer. In the event that *Logout()* has been called before the *FinishMessageProcess()* is completed for a message received from the DataServer, on subsequent reconnection of the DAPI to the DataServer, this pending message will be retransmitted.

The DAPI records certain information in its own data files. An example is the DAPI.log. The information recorded in these files is useful for the DAPI to determine what to send or receive from the DataServer when a connection is made. Therefore, ***under normal conditions, these files should not be deleted or modified.***

6. Verification of transmission / reception of messages

Before sending each message to the DataServer the DAPI stores the message in a file. On reconnection, the DAPI uses this file to decide what to retransmit. There are certain other files within the DAPI directory structure, one for each DataServer, which store the messages that have been received and processed by the wrapper application. It is essential that the wrapper application has sufficient privileges to create and modify these files.

6.1 Unexpected connection loss

This is when the DAPI loses connection with the DataServer due to external causes. In such cases, the DAPI starts attempting to reopen the connection with the DataServer, and on successful connection it receives the messages that are not marked as having been processed. If

a message that had been marked as already processed were to be received by the DAPI, the *OnMessageReceived()* event would not be triggered.

6.2 Expected connection loss

This is when the wrapper application calls the *Logout()* DAPI method. The DAPI memory is released. Shutting down and restarting the wrapper application or the DAPI would be considered as an expected loss of connection.

7 Methods and events in the DAPI

7.1 Methods in the DAPI

Methods are functions that the OCX and the C++ library make available to the wrapper application so that it can communicate with the SMF system.

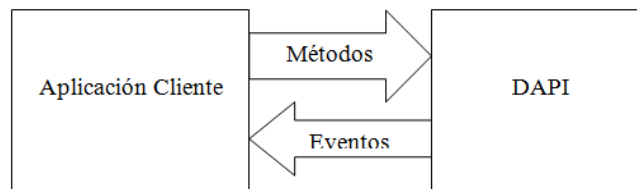


Figure 3 Interaction scheme between the wrapper application and the DAPI

Below are described the methods of *CdxapiCtrl* ActiveX control and the ones for *CdcApi* library. Each method is presented in the ActiveX (OCX) control format and then in C++ format for C++ library.

For more details see section [Error Code List and Message descriptions](#).

7.1.1 AddLoginCondition method

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*)

Returns

Returns 0 if there are no errors. If there are errors, one of the following errors will be returned:

DAPI_ERR_NOT_INITIALIZED_YET
DAPI_ERR_ILLEGAL_RECORD_TYPE_SPECIFIED
DAPI_ERR_ILLEGAL_FIELD_NAME_SPECIFIED

An error in the condition evaluation can also generate the following error:

DAPI_ERR_CONDITION_EVALUATION.

Parameters

message_type The message type to be handled.

field_name The field of the message to be checked.

Optionally a logical operation can be defined between two previously defined conditions using the following operators:

Operator	Meaning
" "	OR. A logical OR operation between the two previously defined results of calling this method. The result of this logical operation can be used in a subsequent logical operation using the BNF execution format.
"&"	AND. A logical AND operation between the two previously defined results of calling this method. The result of this logical operation can be used in a subsequent logical operation using the BNF execution format.

cond_operation The logical operation to be performed between the *field_name* and *eq_value* parameters. The conditional operators can be:

Operator	Meaning
"=="	True when both operands have equal contents
"!="	True when both operands have equal contents
"LIKE"	The SQL comparison operator LIKE. This operator permits searching for patterns of text within a field. The comparison pattern (the second operand) can contain the following characters: %: Stands for 0 or more characters (any character) *: Stands for any character -: Stands for any character [...]: Stands for any of the characters placed inside the brackets. [^...]: Stands for any character except those placed inside the brackets.
"NOT LIKE"	Logical inverse of the SQL comparison operator LIKE.

eq_value The value against which the contents of the parameter *field_name* will have to be compared against.

Description

This method allows the selection of the messages to be received by the application. The messages which do not comply with the condition are discarded by the DAPI and are not delivered to the application.

Before using *Login()*, call this method as many times as required and using different conditions if necessary. The parameter *field_name* may contain '&' or '|' to indicate logical operations between two conditions previously defined. BNF format must be used to control the final result between the selection conditions as shown in the following examples:

```

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, "&", "", "");

```

The above examples accept record types of *L_msType* with the field named *instrument* having values 'SWAP' or 'NDF' and with the field *buyer_dealer* starting with 'AAAL' or with the field *seller_dealer* starting with "BBBL"

If no condition is specified for a particular message type, all the messages of that type will be accepted and delivered to the application.

7.1.2 FinishMessageProcess method

VARIANT CdxapiCtrl :: **FinishMessageProcess** (BSTR *message_type*, BSTR *message_id*)

int CdcApi :: **FinishMessageProcess** (LPCTSTR *message_type*, ULONG *message_id*)

Returns

Returns 0 if there are no errors. If there are errors, the number corresponding to the error will be returned.

Parameters

message_type: Indicates the message type. Use the value received by the event *OnMessageReceived*.

message_num: Indicates the message number. Use the value received by the event *OnMessageReceived*. This is a unique identifier of the record and is directly associated with the message generated by the DataServer.

message_id: This is the message identifier. Use the value received by the event *OnMessageReceived*. This is a unique identifier of the record and is directly associated with the message generated by the DataServer.

Description

This method frees a particular message from the message queue managed by the DAPI and marks the message as processed. All those messages which have not been marked as processed using this method will be retransmitted on reconnection of the DAPI.

7.1.3 Login method

VARIANT CdxapiCtrl :: **Login** (BSTR *username*, BSTR *password*)

int CdcApi:: **Login** (LPCTSTR *username*, LPCTSTR *password*)

Returns

Returns 0 if there are no errors. If there are errors, the number corresponding to the error will be returned.

Parameters

Username The username that the DAPI will use to connect to the DataServer. This would be provided by the administrator of the system.

Password The password that the DAPI will use to connect to the DataServer. This would be provided by the administrator of the system.

Description

Login() initiates an asynchronous connection attempt on behalf of the DAPI with the DataServer. It also starts a few other processes which monitor the connection and control the message flow with the DataServer. Subsequent calls of this method within the same session of the DAPI application only updates the username and password which the DAPI uses to connect to the DataServer.

Apart from providing the username and password, the system administrator will also have to provide sufficient privileges to the DAPI user to receive the desired market information.

The IP address of the DataServer and other relevant information required to establish the connection with the system, should be updated in *dapi.cfg*. This file should reside in the same directory from where the DAPI interface application is being executed. The *OnConnection* event indicates the result of the Login process. If the Login attempt fails, or the connection to the DataServer is lost, the DAPI will initiate connection attempts every 3 seconds until reconnection is achieved, or *Logout()* is called.

After four consecutive unsuccessful attempts to connect to a particular DataServer, the DAPI will try to connect to the next DataServer in the list as indicated in the *dapi.cfg* file. On reaching the end of the list, it will start again with the first DataServer configured.

7.1.4 Logout method

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

Returns

Returns 0 if there are no errors. If there are errors, the number corresponding to the error will be returned.

Parameters

This method does not accept any parameters.

Description

This method is used to cause a normal termination of the connection of the DAPI with the DataServer. This function can also be used to end reconnection attempts to the DataServer by *Login()*, in which case all the threads launched by *Login()* are also terminated. It is important to note that messages which have not yet been marked as processed by using *FinishMessageProcess()* at the moment of calling *Logout()*, will be retransmitted upon reconnection of the DAPI to the DataServer.

7.1.5 GetNextFieldName method (ActiveX)

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

Return value

Returns the name of the next field in the message. If there is no next field in the message, or if an error occurs, nothing is returned.

Parameters

field_name The name of the field whose subsequent field's name is to be fetched. Please refer to the exact field name as provided in the Message List. If nothing (") is configured for this parameter, the name of the first field in the message is returned.

Description

This method is used to get the name of the next field in a message. It can be used within a loop to browse all fields of a message. In this case, the first field name must be asked with an empty string (").

This method allows parsing through the field names of the last message processed by *TakeMessage()*. Please note that for now only the Spanish version of the field name may be used.

7.1.6 GetNextField method (Library)

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

Returns

Returns the data type of the next field in the message. See table below. In the event of an error, returns 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

Parameters

field_name The name of the field whose subsequent field data type and content is to be obtained. If nothing (") is configured for this parameter, it returns the name of the first field in the message. The name of the field queried (i.e. the next field) will be copied to *field_name*.

data_copy The contents of the field queried (next field) will be copied to this parameter.

Description

This method permits browsing through the fields of the last message processed by *TakeMessage()* and obtaining their contents. It returns the field name, field type and the value stored in the field. This method can be used within a loop to browse through the message. It is important to ensure that the parameters *field_name* and *data_copy* have sufficient space allocated to store the field name and its contents. Only the Spanish version of the field names may be used at the moment.

7.1.7 GetField method (ActiveX)

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

Returns

Returns the contents of the field in the form of a string. Returns an empty string in the event of an error.

Parameters

field_name The name of the field whose content is required.

Description

Returns the value contained in the field mentioned in the parameter *field_name* as a string. In the event of an error, the *OnError* event handler is triggered and it returns an empty string. This method allows parsing through the fields of the last message processed by *TakeMessage()*. Please note that for now only the Spanish version of the field names may be used.

7.1.8 GetField method (Library)

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

Returns

Returns the data type of the field queried. See table below. In the event of an error, returns 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

Parameters

field_name The name of the field whose content is to be obtained.

data_copy The contents of the field queried will be copied to this parameter.

Description

This method permits the contents of a field in a message to be obtained. The method allows parsing through the fields of the last message processed by *TakeMessage()*. For now only the Spanish version of the field names may be used.

7.1.9 GetFieldBSTR method (Library)

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

Returns

Returns the length of the field queried. In the event of an error, it returns 0.

Parameters

field_name The name of the field whose length and content is to be obtained

data_copy The contents of the field queried will be copied to this parameter as a null terminated string.

Description

This method obtains the contents of a message field in string format. The method allows parsing through the fields of the last message processed by *TakeMessage()*. For now, only the Spanish version of the field names may be used.

7.1.10 GetFieldDATE method (Library)

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

Returns

Returns the length of the field returned, which is 8 characters. In the event of an error, returns 0.

Parameters

field_name Name of the field whose value is to be obtained.

data_copy The contents of the field queried will be copied to this parameter, in the format YYYYMMDD.

Description

This method obtains the contents of a field whose data type is date. The field is copied as a sequence of 8 bytes which contains the date. The method allows parsing through the fields of the last message processed by *TakeMessage()*. For now, only the Spanish version of the field names may be used.

7.1.11 GetFieldLONG method (Library)

LONG CdcApi :: GetFieldLONG(LPCTSTR *field_name*)

Returns

Returns the value of the field converted to a *signed long* data type. If an error is encountered, the *OnError* event handler is triggered and returns a value of 0.

Parameters

field_name Indicates the name of the field whose value is to be obtained.

Description

This method obtains the content of a field as a *signed long* number. The method allows parsing through the fields of the last message processed by the *TakeMessage()*. Please note that for now only the Spanish version of the field names may be used.

7.1.12 GetFieldULONG method (Library)

ULONG CdcApi :: GetFieldULONG (LPCTSTR *field_name*)

Returns

Returns the value of the field converted to an *unsigned long* data type. If an error is encountered, the *OnError* event handler is triggered and returns a value of 0.

Parameters

field_name Indicates the name of the field whose value is to be obtained.

Description

This method obtains the content of the field as an *unsigned long* number. The method allows parsing through the fields of the last message processed by *TakeMessage()*. Please note that for now only the Spanish version of the field names may be used.

7.1.13 InitMessage method

VARIANT CdxapiCtrl :: **InitMessage** (BSTR *message_type*)

int CdcApi::InitMessage (LPCTSTR *message_type*)

Returns

Returns 0 if there are no errors. If an error occurs, it returns the appropriate error code.

Parameter

message_type Indicates the type of the message to be constructed. If the parameter is empty (""), the type and contents of the last message handled by *TakeMessage()* will be used as the base for the new message.

Description

Using this method, the construction of a new message to be sent to the system can be initiated. The DAPI wrapper application must first call this method and then populate the fields of the message with appropriate values. If the *message_type* parameter is blank, then only those fields which are different in the new message as compared to the previous one handled by *TakeMessage()* need to be populated. Use *SetField()* to populate the message fields. If it were necessary, specify the message targets with method *SendMessageTo()*. Finally call *SendMessage()* to send the message to the system.

7.1.14 ProcessMyEvents method (Library)

Void CdcApi :: **ProcessMyEvents**(void)

Description

Call this method to use the multithread protection of the DAPI. On calling this method, the DAPI will call certain subroutines which will attend to all the events that have occurred up to that moment. Once processed, it suspends attention to all events until the next call of *ProcessMyEvents()*. Note that error events are thread-safe whereas normal events are not.

Do not call this method if the application solves multithreading independently.

7.1.15 SendMessage method

VARIANT CdxapiCtrl :: **SendMessage** (BSTR *message_type*)

int CdcApi :: **SendMessage** (LPCTSTR *message_type*)

Returns

Returns 0 if there are no errors. If an error occurs, it returns the appropriate error code

Parameters

message_type Indicates the type of the message that needs to be sent. If the parameter is empty, the last message handled by *TakeMessage()* will be used.

Description

This method is used to send a previously constructed message to the system. Use *InitMessage()* to initialize the message, then populate the fields of the message using *SetField()*. If it were necessary, specify the message targets with method *SendMessageTo()*. Finally execute *SendMessage()* to deliver the message to the system.

7.1.16 SendMessageTo method

VARIANT CdxapiCtrl :: **SendMessageTo** (BSTR *address_type*, BSTR *address_data*)

int CdcApi :: **SendMessageTo** (LPCTSTR *address_type*, LPCTSTR *address_data*)

Returns

Returns 0 if there are no errors. If an error occurs, it returns one of the following error codes:

DAPI_ERR_NOT_INITIALIZED_YET

DAPI_ERR_ILLEGAL_ADDRESS_TYPE_SPECIFIED.

Parameters

address_type

Indicates the type of address where the message has to be sent to. This can be:

Value	Meaning
"PU"	Primary user
"SU"	Branch
"IN"	Institution
"XP"	Except this primary user
"XS"	Except this branch
"GR"	Name of an internal group

address_data

Indicates the addresses where the message has to be sent. It is recommended that the validity of this address is confirmed with the system administrator. This can be:

<i>address_type</i>	Value
"PU"	Name of a user PU (4 characters)
"SU"	Name of a branch(8 characters)
"IN"	Name of an institution (4 characters)
"XP"	Name of a user PU that is not a receiver (4 characters)
"XS"	Name of a branch that is not a receiver (8 characters)
"GR"	Name of an internal group or receivers (8 letters)

Description

This method sets the address to which the message needs to be sent. This method should be called before using *SendMessage()*. If the message needs to be sent to various destinations, this method should be called multiple times. The destination address (*address_data*) can vary in size depending on the type of address. Normally it is the code assigned to the users, branches or institutions by the administrator.

By default, that is to say if this method is not called, the message will be sent to the Datamanager.

Example:

```

m_dapi.TakeMessage (L_MessType, mess_num ); // Assigns the last message received
m_dapi.InitMessage (""); // Initializes the message to be sent equal to
mess_num

m_dapi.SetFieldBSTR(L_MessType, "bandera", "S"); // Changes some field

// Assigns as receivers all the users of the branch
m_dapi.SendMessageTo ("SU", m_dapi.GetField("sucursal")
// except this user
m_dapi.SendMessageTo ("XP", m_dapi.GetField("codigo_primaryuser")

// Sends the message
m_dapi.SendMessage(L_MessType);

```

7.1.17 SetField(ActiveX) and SetFieldBSTR(Library) methods

VARIANT CdxapiCtrl :: **SetField** (BSTR *message_type*, BSTR *field_name*, BSTR *new_value*)

int CdcApi::SetFieldBSTR (LPCTSTR *message_type*, LPCTSTR *field_name*, LPCTSTR *new_value*)

Returns

Returns 0 if there are no errors. If an error occurs, it returns the appropriate error code.

Parameters

message_type Indicates the type of message whose field needs to be set.

field_name Indicates the name of the field whose value has to be set.

new_value Indicates the value to be set to the field.

Description

This method assigns the contents of *new_value* (that must be BSTR or LPCTSTR type data) to the field specified in *field_name* which belongs to the message in *message_type*. Remember that before assigning the field values, the message has to be initialized using *InitMessage()*. If *message_type* is null, the last message handled by *TakeMessage()* will be used. If an error occurs, the *OnError* event handler will be triggered.

7.1.18 SetFieldDATE method (Library)

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

Returns

Returns 0 if there are no errors. If an error occurs, it returns the appropriate error code.

Parameters

message_type Indicates the type of the message whose field needs to be set.

field_name Indicates the name of the field whose value has to be set.

new_value Indicates the value (a date) to be set to the field.

Description

This method assigns the contents of *new_value* (which must be a date string with format YYYYMMDD) to the field specified in *field_name* which belongs to the message in *message_type*. Remember that before assigning the field values, the message has to be initialized using *InitMessage()*. If *message_type* is null, the last message handled by *TakeMessage()* will be used. If an error occurs, the *OnError* event handler will be triggered.

7.1.19 SetFieldLONG method (Library)

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

Returns

Returns 0 if there are no errors. If an error occurs, it returns the appropriate error code.

Parameters

message_type Indicates the type of the message whose field needs to be set.

field_name Indicates the name of the field whose value has to be set.

new_value Indicates the value to be set to the field.

Description

This method assigns the contents of *new_value* (which must be a *long* number) to the field specified in *field_name* which belongs to the message in *message_type*. Remember that before assigning the field values, the message has to be initialized using *InitMessage()*. If *message_type* is null, the last message handled by *TakeMessage()* will be used. If an error occurs, the *OnError* event handler will be triggered.

7.1.20 SetFieldULONG method (Library)

```
int CdcApi :: SetFieldULONG (LPCTSTR message_type, LPCTSTR field_name, ULONG new_value)
```

Returns

Returns 0 if there are no errors. If an error occurs, it returns the appropriate error code.

Parameters

message_type Indicates the type of the message whose field needs to be set.

field_name Indicates the name of the field whose value has to be set.

new_value Indicates the value to be set to the field.

Description

This method assigns the contents of *new_value* (which must be an *unsigned long* number) to the field specified in *field_name* which belongs to the message in *message_type*. Remember that before assigning the field values, the message has to be initialized using *InitMessage()*. If *message_type* is null, the last message handled by *TakeMessage()* will be used. If an error occurs, the *OnError* event handler will be triggered.

7.1.21 TakeMessage method

```
VARIANT CdxapiCtrl :: TakeMessage (BSTR message_type, BSTR message_id)
```

int CdcApi :: **TakeMessage** (LPCTSTR *message_type*, ULONG *message_num*)

Returns

Returns 0 if there are no errors. If an error occurs, it returns the appropriate error code.

Parameters

message_type Indicates the type of the message. Use the same value as received in the event *OnMessageReceived*.

message_num Indicates the message number. Use the same value as received in the event *OnMessageReceived*. This is a unique identifier of the message and is directly associated with the message generated by the DataServer.

message_id Indicates the identifier of the message. Use the same value as received in the event *OnMessageReceived*. This is a unique identifier of the record and is directly associated with the message generated by the DataServer.

Description

This method prepares a message to be read using the *GetField* method. The DAPI can receive messages from various DataServers within the system. These messages are assigned a unique identifier and on reception of the message, the *OnMessageReceived* event is triggered. This unique identifier must be used in *TakeMessage()* to prepare the message to be read.

7.1.22 Método IsDefined

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

Valor de retorno

Returns 1 if field exist in the message. Otherwise returns 0.

Parámetros

field_name: Indicates the field name of which it is required to confirm its existence into the message.

Descripción

This method searches the field *field_name* in the message.

7.1.23 Método GetCompilationType

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

Valor de retorno

Returns always 1.

Parámetros

compilation_type: The Dapi compilation platforms is stored.

Descripción

This method assigns a string to *compilation_type*; in case the Dapi build platform is 32-bit, it assigns "x86"; otherwise assigns "x64".

7.1.24 Método GetVersion

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

Valor de retorno

Returns always 1.

Parámetros

version: The Dapi versión is stored.

Descripción

This method assigns a string to *version*; with the version number of the generated Dapi.

7.2 Events available in the DAPI

Event handlers are predefined methods which the DAPI calls when a particular event occurs. The application should contain the necessary source code to perform the actions required for each event in the corresponding event handler. The event handler informs the application that an event has occurred.

7.2.1 Events in the DAPI ActiveX control (OCX)

In the OCX version of the DAPI, an event is triggered based on the Windows messaging scheme. Therefore, all the events are processed in a sequence and without internal interference between events and the execution of the event handlers.

Visual Basic applications must use the *Invoke* method to execute an event handler provided by the DAPI and therefore pass information to the window (see the example in Visual Basic supplied with the distribution kit). This is necessary because the thread which attends to a particular window of the application is independent of the other event-related asynchronous processes.

7.2.2 Events in the DAPI(Library) library

In this case, the wrapper application should periodically call the *ProcessMyEvents* method (see *drvapitest.cpp*). This interaction will cause certain delay in processing the messages, but this delay is directly related to the time interval between each call of *ProcessMyEvents*. Excessive calls of this method will result in unnecessary use of CPU time.

7.2.3 OnMessageReceived event

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

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

Parameters

message_type This parameter indicates the type of the message received. This value must be used in *TakeMessage* to prepare a message to be read.

message_num The number assigned by the DAPI to this message. This value must be used in *TakeMessage* to prepare a message to be read.

message_id The number assigned by the DAPI to this message. This value must be used in *TakeMessage* to prepare a message to be read.

Description

This event is triggered when the DAPI receives a message from the DataServer. The application can access the message by calling *TakeMessage()* followed by *GetField()*. The number or id must also be used with the *FinishMessageProcess()*. The DAPI maintains a list of messages received until the *FinishMessageProcess()* is called or until disconnection. On reconnection it will receive all those messages which have not been marked as processed.

7.2.4 OnConnection event

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

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

Parameters

result_code This code indicates the result of the connection. Zero would mean that the connection has been established satisfactorily. If unable to establish the connection, the ASCII code of the letter received from the DataServer is returned.

Description

Connection is attempted by calling *Login()*, and at the end of each attempt the DAPI executes this event to report the result of the connection attempt.

7.2.5 OnDisconnection event

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

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

Parameters

reason_code

Shows the actual state of the connection.

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

reason_text The possible cause of the disconnection.

Description

The connection is maintained by the DAPI for an indefinite period of time. Once the connection has been established, if there is a disconnection the DAPI will generate this event. Note that in the event of an unexpected connection loss, the messages which were received but not marked as processed will be re-received by the DAPI on reconnection.

7.2.6 OnError event

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*)

Parameters

error_code

error_text

An explanatory text about the error.

next_action_code:

Suggested next action. The possible values are:

Value	Definition	Meaning
0	DAPI_NEXT_ACTION_CONTINUE	The DAPI can continue running.
1	DAPI_NEXT_ACTION_RECONNECT	Can try again to connect, correcting the user name or the password used.
999	DAPI_NEXT_ACTION_EXIT	The DAPI cannot continue to run because of this error.

next_action_data

Extra data required to execute the next action. Not available in the current version.

Description

This event is generated when the DAPI encounters an abnormal condition or when a method was not executed successfully. The DAPI records each *OnError* event in its log and the error is made available to the wrapper application for its use.

7.2.7 OnStatusChanged event

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

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

Parameters

status_code

Indicates the actual state of the connection. The possible values are:

Value	Definition	Meaning
7	TCP_ERROR_CANT_CONNECT	The TCP/IP connection has failed. Another attempt will be made, and according to certain pre-established rules a different server may be used in this attempt.
21	TCP_STATUS_OPEN_CONNECTION	A TCP/IP connection has been established with the server.
20	TCP_STATUS_CLOSED_CONNECTION	The TCP/IP connection with the server has been lost.

27	TCP_SHUTTING_DOWN	A connection close has been requested. This connection closure has been started, and the last pending data is being sent.
----	-------------------	---

status_text

Indicates the status text:

Definition	Value	Meaning
TCP_STATUS_OPEN_CONNECTION	"Initiating translation" log	A log file is being retro-alimented and simulates a connection.
TCP_STATUS_CLOSED_CONNECTION	"Finalizing translation" log	A log file is being retro-alimented and simulates a disconnection.
TCP_STATUS_OPEN_CONNECTION	"Connected"	A TCP/IP connection with the server has been established.
TCP_STATUS_CLOSED_CONNECTION	"Disconnected"	The TCP/IP connection with the server has been lost.
TCP_ERROR_CANT_CONNECT	"Can't connect"	The TCP/IP connection has failed. Another attempt will be made, and according to certain pre-established rules a different server may be used in this attempt.
TCP_SHUTTING_DOWN	"Attempts to establish a connection and processing messages have been suspended"	A connection close has been requested. This connection closure has been started, and the last pending data is being sent.

Description

This event is generated on establishing a connection, on a connection loss, or on logout of the DAPI.

7.2.8 OnDataReceived event

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

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

Parameters

source_id

The name of the connection. When there are multiple connections defined in the system.

message_name

Type o message.

message_data

Message content. This event is designed specifically for verification purposes.

Description

This event cuts the received message at the first null character and places the content in the *message_data* field.

8 DAPI code examples

There are two sample programs that show how to interact with the DAPI.

VbdxApiTest – A Windows example made in Visual Basic using the DAPI Activex control.

Inxdrvapitest– A Linux example made in C++ using the libdapi.a or libdapi.so libraries.

These examples have been prepared so that they highlight the following actions of the DAPI:

- The DAPI login process using *login()*.
- Triggering of the event *OnMessageReceived()* on reception of a message.
- Selection of a message to obtain its contents using *TakeMessage()*
- Accessing the fields of the message using *GetField()*
- Construction of a message to be sent to the system using *InitMessage("Message Name")* and *SetField()*
- Construction of a partially changed message which had been received from the system and is to be sent back to the system using *InitMessage("Message Name")* and *SetField()*
- Event triggered on connection to the DataServer *OnConnection()*
- Event triggered on disconnection from the DataServer *OnDisconnection()*
- Event triggered on encountering an error *OnError()*

8.1 Example of how to process a message in Visual Basic

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

8.2 Example of how to send a message using C++

Example 1:

```
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
```

Example 2:

```
m_dapi.TakeMessage (L_MessType, mess_num ); // Inicia un mensaje en blanco
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 Sample workflow process for gathering transaction information for back-office applications

It is important to carefully identify and select the pieces of information that the back-office application will need. Therefore, this section will outline a minimalistic process to properly identify the data gathered by the DAPI from the system.

The most useful message that the DAPI receives from the system is the *Orden_Transaccion* which is send every time an action affecting the market window occurs. Examples of such actions are: entering, modifying, withdrawing, or hitting bids/offers.

For practical purposes the following workflow will focus on transactional events only. See **Error! Reference source not found..**

1. Once *Orden_Transaccion* is received read *clase_mensaje* field and depending on its value proceed to the branch accordingly.
2. Regardless of the branch, identify the transaction with a unique identifier which is formed by the union of the following fields: *mercado+fecha+hora+codigo_postura+codigo_postura_1*. For example, the unique identifier of a transaction can be: 83+20091028+152114+A00CHZF3R9+A00CHZF3RA (i.e 36 characters).
3. Optionally, if it is necessary to differentiate between hits and registers read the content of the *bandera* field. If it is empty the message is a hit, otherwise an "R" marks it as a register.
4. Extract particular information as needed. Refer to the accompanying Excel file for a complete field listing of the *Orden_Transaccion* message.

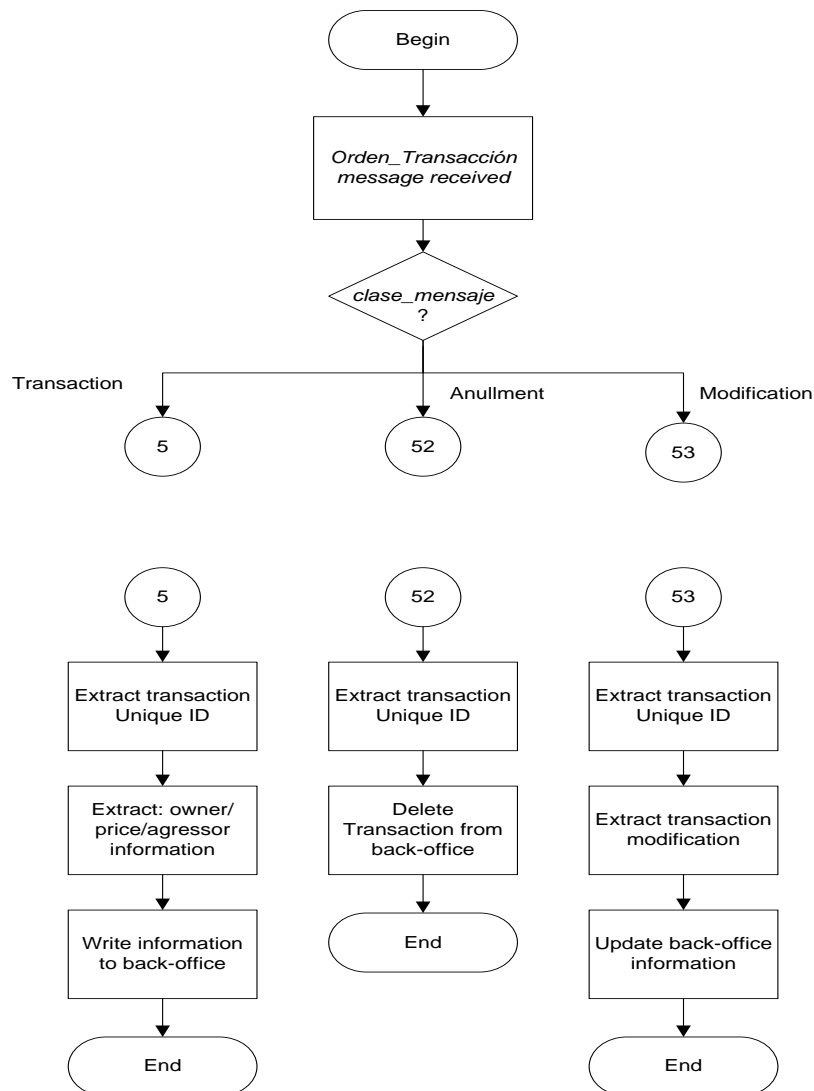


Figure 4 Flowchart depicting the recommended processing of an *Orden_Transaccion* message for back-office logging.

Note: This workflow is a sample and can be taken as a base for the programming of transactions processing. However, it could be other variables of each environment that must be taken into account by each client, when an application that uses the DAPI is being designed/developed.

9 Configuration of the DAPI in the DMclient

The administrator of the system should setup the identity of the DAPI using the DataManager Client. To perform this, select from the Master Files window the institution and branch to which this new DAPI user will belong to and create a new Primaryuser (PU) as shown in **Error! Reference source not found.** and **Error! Reference source not found..** Note that the DAPI's PU has a special property called Dapi Client which must be set to Yes (shown highlighted for clarity).

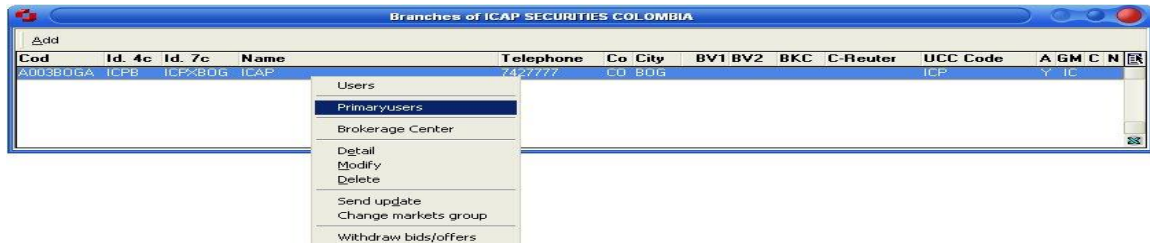


Figure 5 Select the branch and right-click to view the menu



Figure 6 Addition of a new PU assigned to the branch of the DAPI user

Once the PU is created proceed to add a new DAPI user. There is no need to grant market access for this user. The only restriction that has to be taken into account for this user is that it must use the Primaryuser with special properties created previously.



Figure 7 Properties of the new DAPI user.

In the example above the key information needed for the dapi.cfg file is the branch code (A003BOGA), the user code (A05L), and the primary user code (A05Q) used for the *local_code* directive in the dapi.cfg.

Finally, assign a password to the user with the DataManager Client. Note that this password is case sensitive, does not expire, and is automatically blocked after three unsuccessful attempts.



Figure 8 User passwords option in the DataManager Client

Note: This is a sample of the general configuration of a DAPI user. There are other types of DAPI users that depends on the Institution configuration, that applies to certain environments and are not treated in this document.

10 Configuration of the DAPI in the client (dapi.cfg)

In the DataManager the administrator of the system should setup the identity of the DAPI as a separate user. The user name, password and a configuration file called DAPI.cfg should be made available to the DAPI site by the administrator. This file will contain, amongst other information, the branch code and the user code assigned for the DataManager.

The following is an example of the contents of the DAPI.cfg file. The character '#' is used at the beginning of comment lines:

```
# -----
# Datatec Api (DAPI) sample configuration file.
# -----

oper_file_version = "DAPI WEBB WINDOWS"
data_server       = $02B
ds_ip_address     = 100.100.32.206
ds_tcp_port       = 7006
# PrimaryUser code is the local_code value.
local_code        = 02I6
user_code         = 030B
branch_code       = DVNBBOGA
user_name         = DAPIDVNB
password          = DATA1234

# 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

# This is the esquema code, its size is 8 characters.
# It is used in environments where it is possible to get the colors
# of the elements in the DAPI messages. By now MX RF environment.
# Uncomment the next line in MX RF.
#esquema_code     = ICAP0002

# 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
```

Figure 9 DAPI.CFG File.

Parameter	Example	Description
oper_file_version	SET-FX DAPI	Version description
local_code	G0OC	Primary user code assigned to the DAPI by the administrator of the system using the DataManager
user_code	G0IM	User code assigned to the DAPI by the administrator of the system using the DataManager. A User code is different from a Primary user code.
send_mode	2	Communication protocol. 0 - Encrypted IDEA (Deprecated) 1 - Not Encrypted & Not Compressed 2 - Compressed using BSD or RLE, depending on the size of the message (Default) 3 – Encrypted using IDEA and Compressed. (Deprecated)
use_ssl	0	0 – Without SSL. (Default) 1 – Connect using SSL. If this parameter is set to 1, which we recommend to secure the connection, the following files should be copied to the same directory where the executable is located:: - ssl.cfg (with -v 3 option) - ssl/certs/server.pem - ssl/certs/cuduser.pem - ssl/private/ca.crt - The *.pem files with their names changed to their hash numbers under the ssl subdirectory of the execution directory. For example: - ssl/certs/6_users/5b7f9658.0 - ssl/certs/6_users/5f552eb8.0 Additionally, the ds_tcp_port argument must be the corresponding SSL port of the Dataserver. Note: These files are included in the Linux and Windows samples.
branch_code	G0JKUIOA	DAPI User branch code.
data_server	\$02Q	DataSource code. Note: If another dataserver is defined to connect the DAPI application, it is necessary to specify the name, ip and port (ds_ip_address y ds_tcp_port). Note: If there were more dataservers in the environment, their names must be specified even if the DAPI application is not connected to them.

ds_ip_address	192.168.112.22	IP address of the DataServer indicated in the parameter <i>data_server</i>
ds_tcp_port	2008	<p>TCP port of the DataServer indicated in the parameter <i>data_server</i>. This port has to be enabled by the system administrator.</p> <p>If <i>use_ssl</i> argument is set to 1, then this port must be the SSL port that the <i>data_server</i> is listening to. When connecting a DAPI application through Internet, the SSL port is usually the 443 port.</p>
transaction_feed	1	<p>0 – Block reception of transaction messages emitted by the system – this would be used in the case of DAPIs which use only statistics messages. 1 – Receive transaction messages</p> <p>Note: <i>guibos_feed</i> and <i>transaction_feed</i> cannot be specified simultaneously. Only one of them can be defined with value 1.</p>
guibos_feed	1	<p>0 – Block reception of Guibos messages emitted by the system – this would be used in the case of DAPIs which use only statistics messages. 1 – Receive Guibos messages</p> <p>Note: <i>guibos_feed</i> and <i>transaction_feed</i> cannot be specified simultaneously. Only one of them can be defined with value 1.</p> <p>Note: Guibos are persistent (not high and not low priority messages for log) messages and the DAPI asks for persistent log seven days back by default. If there are processed messages, then the persistent log is asked from the date of the latest Guibos message marked as processed. Therefore, the following options are not used for <i>guibos_feed</i>.</p> <ul style="list-style-type: none"> - <i>log_days_high_messages</i> - <i>log_days_low_messages</i> - <i>log_last_processed_message</i>
table_code	31 Grupos_Vigentes	<p>Specifies the table code and name of global table codes that DAPI subscribes to. Format: Table_code <table_internal_code> <msg_type></p> <p>Where: <table_internal_code> is the code of the table. This information must be provided by Icap del Ecuador. <msg_type> Is the message type that corresponds to the table and depends directly of given the code.</p> <p>Several tables can have the same message type. The codes and types must be provided by</p>

		<p>icap del Ecuador and depends on the the market and data received by DAPI.</p> <p>Note. For example, for Fixed Income markets 69,140, 141 they are: 31 Grupos_Vigentes 36 Mercados_Renta_Fija_Vigentes 6906 Resúmenes_Vigentes 14006 Resúmenes_Vigentes 6901 Posturas_Vigentes 6902 Posturas_Vigentes 14001 Posturas_Vigentes 14002 Posturas_Vigentes 14101 Posturas_Vigentes 14102 Posturas_Vigentes 6907 Hechos_Vigentes 6905 Índices_Vigentes</p>
testing_file	Dapi_test.log	<p>The name of the file which will hold the DAPI log. This log will contain all the messages sent/received from the system.</p> <p>This option allows testing the functionality of the DAPI in a real connection. In this way the DAPI can receive messages without being connected to a real environment. Additionally, it is possible to debug possible errors of previous executions where the dapi log is available.</p>
testing_start	20091215091500	Specifies the beginning date (yyyymmdd) and time (hhmmss) from which the DAPI reads the <i>testing_file</i> .
testing_end	20091215091500	Specifies the ending date (yyyymmdd) and time (hhmmss) until the DAPI reads the <i>testing_file</i> .
testing_output_file	Test_dapi_output.log	Output log file for reading of <i>testing_file</i> .
language	ENGLISH	<p>Specifies the language to be used in the messages types that are sent to the application that uses the DAPI.</p> <p>Possible values: SPANISH,ENGLISH</p> <p>If the parameter is not defined, DAPI will use the original message names.</p> <p>For example, Original name: precios_por_hora Spanish name: mensaje_precios_por_hora_monedas English name: currency_price_by_hour_message</p>
log_last_processed_message	F,L,N	<p>Default value = L</p> <p>It controls from which message it is required the high and low priority log messages when the DAPI gets connected to the Dataserver, taking into account the values specified in log_days_high_messages and log_days_low_messages parameters.</p> <p>Possible values are:</p>

		<p>F: Get the log from the oldest message that DAPI had not marked as processed.</p> <p>L: Get the log from the more recent message that DAPI had marked as processed. In this case, any previous message that DAPI had not marked as processed will not be requested.</p> <p>N: Do not use marked messages. Get the log of all the messages since the beginning of the day.</p> <p>Note: <i>If the application always mark as processed the received messages by DAPI, there is no difference in the messages that DAPI gives to the application because all the messages marked as processed are discarded. Of course, there is an exception when the DAPI log files are deleted.</i></p> <p>Example: log_days_high_messages=0 log_days_low_messages=0 log_last_processed_message=N When DAPI ges connected to Dataserver, it request the log messages sent today since the beginning of the day.</p> <p>Example: log_days_high_messages=7 log_days_low_messages=7 log_last_processed_message=F When DAPI ges connected to Dataserver, it request the log messages sent maximum seven days ago, beginning in the oldest message that had not been marked as processed.</p> <p><i>Note: The default value in previous versions than 1.061 was F.</i></p>
use_msg_code	121 in markets 69 140 141	<p>To indicate a different message number for Orden_Transaccion message.</p> <p>In the sample it is indicated to use message number 121 (instead of default 111) in markets 69, 140, 141.</p>
send_Msg_Delay_Milli sec	500	Delay used to send messages, in milliseconds.

Note: All the DataServers available in the system environment to which the DAPI is to be connected must be included in the DAPI.cfg.

10.1 Temporal Configuration File (dapi_tmp_previous_date_log_request.txt)

Extraordinarily, the administrator, can provide the temporary file “*dapi_tmp_previous_date_log_request.txt*”, which instructs the dapi to ask the retransmission of all messages of a defined day in the past. This file has to be created in the same directory of dapi.cfg and current directory of the application. As precaution to avoid future retransmissions the file is deleted after being read.

The required date must be defined in the first characters of the first line and in format YYYYMMDD, if there is any error reading or deleting the file, or with recognizing the date, the error is returned and the normal DAPI process is not continued.

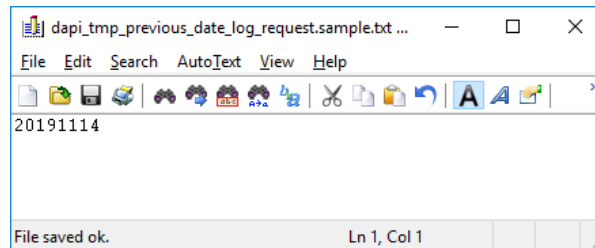


Figure 10 Dapi_tmp_previous_date_log

The possible errors are:

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

11 Installation of the DAPI in Windows

11.1 Installation and use of the Windows ActiveX control (OCX)

The Activex form of DAPI has two main files:

1. The **dxapi.ocx**
2. The **dapi.cfg**
3. The **dapi_tmp_previous_date_log_request.txt**

If being installed over a previous version of the DAPI, please uninstall the previous version executing the following instruction from within the directory where the Dxapi.ocx is found:

Regsvr32 /u dxapi.ocx

Then overwrite the previous version of the Dxapi.ocx with the new version.

For a new installation, copy the Dxapi.ocx file to the directory where the wrapper application will run. Then from within the directory where the new version of the Dxapi.ocx has been copied, type the following instruction to register it:

Regsvr32 dxapi.ocx

This command registers the ocx library in the host so that the OS can properly map calls to that library.

On Windows 64-bit systems the ocx should be installed under the \windows\sysWOW64 folder.

Note: These commands to register/unregister the ocx control must be executed with Administrative privileges.

The next step is to configure the parameters in the DAPI.cfg depending on the chart provided above.

Finally code has to be included in the wrapper/interface application in order to interact with the Dxapi.ocx. Usually the DAPI would be included in a window (e.g. a dialog box), a variable would be created, and the event handlers would be redefined as necessary (especially the *OnMessageReceived* event). If messages have to be sent to the system, the appropriate methods provided by the DAPI for this should be called, using the same variable.

11.1.1 Check the DAPI version in the OCX File

It is possible to check the DAPI version in the OCX file. In order to do that, **right-click** the file and choose **Properties**, then click on **Details** tab.

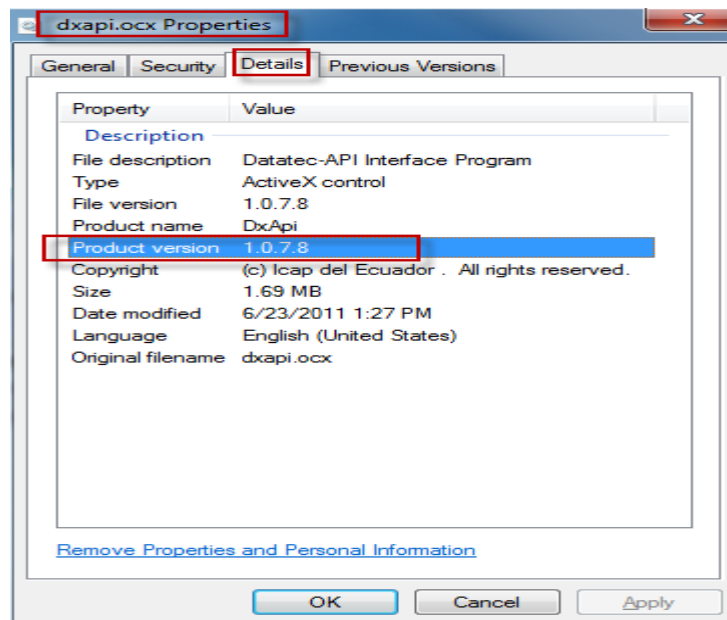


Figure 11 Dxapi.ocx properties

11.2 Installation and use of the Windows library

Copy the library onto the directory where the project is going to be built and import it into the project environment.

Configure the parameters in the DAPI.cfg depending on the chart provided above.

Review the methods defined in the file DRVAPI.cpp, and especially those methods which are event handlers for DAPI runtime generated events. If required, a new class may be derived using `CdcApi` as the base class.

11.3 VbdxApiTest - Windows example of the use of DAPI

11.3.1 Note for 64-bit Windows machines running Visual Basic example

To execute the Visual Basic example in Windows 64 bits platforms that doesn't have Microsoft Visual C++ 2005 installed it is required to install the package Microsoft Visual C++ 2005 Redistributable Package which includes Visual C++ runtime components.

This package installs runtime components such as C Runtime (CRT), Standard C++, ATL, MFC, OpenMP and MSDIA libraries. In the case of libraries that allow the simultaneous implementation model those components are installed in the native assembled cache folder or WinSxS.

Before installing this package is necessary to verify that there is no similar package installed, checking in **Control Panel -> Programs and Features**. If a similar package is installed it must be uninstalled before installing the new package.

Next, install the package by opening a DOS window and executing the following command:

```
vcredist_x64 /Q
```

When the installation is completed the same will be visible in the Program and Features window:

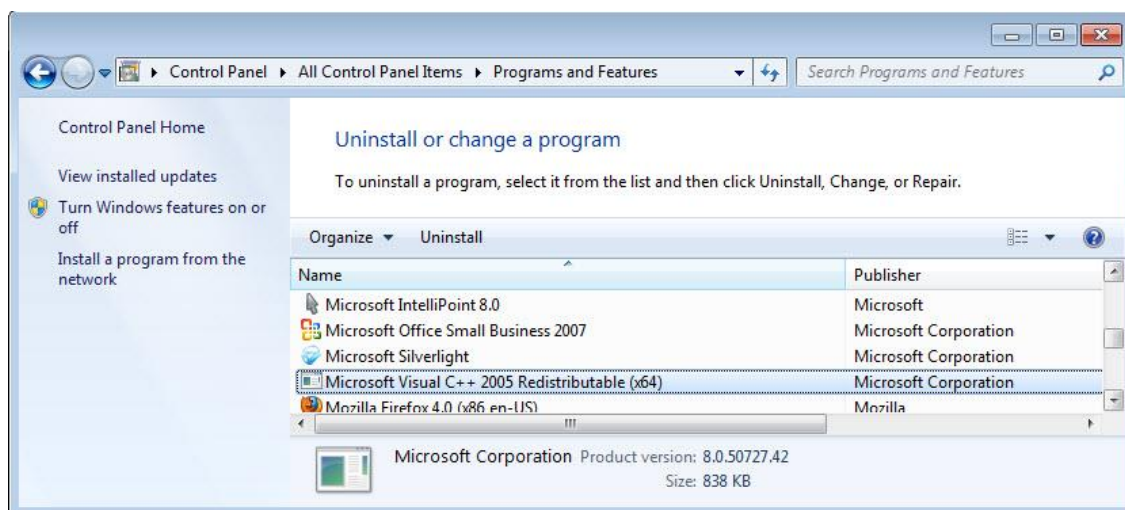


Figure 12 Microsoft Visual C++ redistributable package installed on the client machine

11.3.2 Running Visual Basic example

A simple and easy to use example has been created using Visual Basic to illustrate the use of the DAPI and its function calls. It is found under the `\samplewindows\vbdxapitest\bin` directory of the distribution kit. Copy the contents of this directory to the local disk of the machine where the example is to be run and follow the steps indicated in **README.TXT**.

The example **vbdxapitest** is distributed in the folder *sample*. Before executing this example you should verify the following:

1. The configuration of the DAPI user in the DataManager.
2. The configuration of the user data and of the DataServer in the file *dapi.cfg*. (*dapi.sample.cfg* is provided).
3. The file *dapi.cfg* should be in the same folder where the example program *vbdxapitest* will run.
4. The folder from where the program will be run must be folder on a local drive. The Visual Basic DAPI will not run from a network drive.
5. The ActiveX Control *dxapi.ocx* must have been registered (see chapter on the DAPI instalation).

On executing the example, you should click on the **Login** button, and optionally fill in the requested username and password fields.

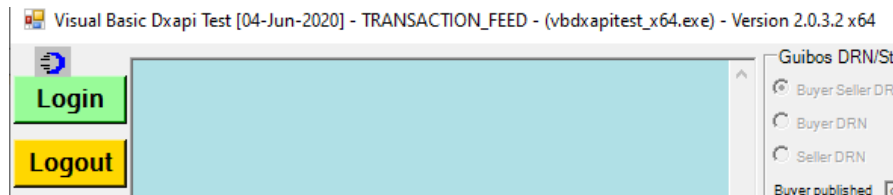


Figure 13 VbdxApi Test Application Main Window prior to login

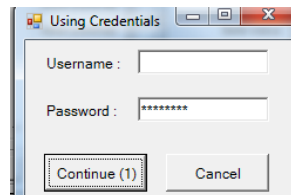


Figure 14 VbdxApi Test Credentials window (can be ignored).

The following screen shot shows how the example appears after a successful connection.

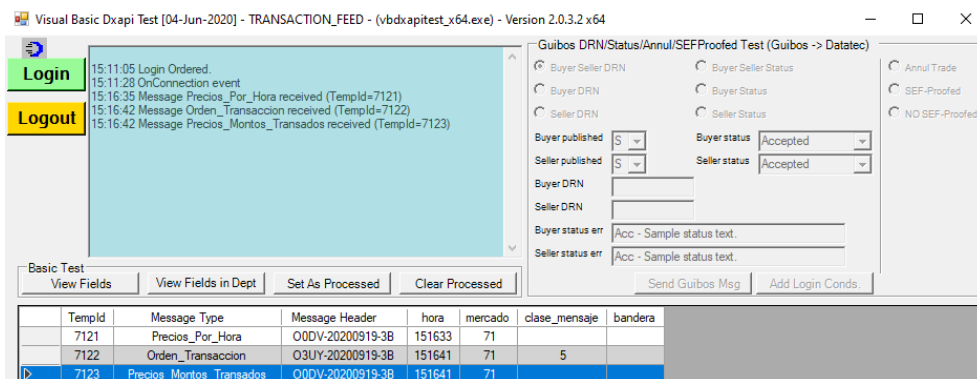


Figure 15 VbdxApi Test Application Main Window

In this example, the ActiveX Control (*dxapi.ocx*) was included in the window (*Form1*). The name given to this control was *Axdxapi1*.

11.3.3 Login and Logout buttons

The button **Login** calls the method **Login** of *Axdxapi1* as can be seen in the line of code in *Form1.vb*:

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

The button **Logout** calls the method **Logout** of *Axdxapi1* in a similar way to **Login**.

In order to manage events in Visual Basic the method **invoke** must be used. This method avoids concurrent execution conflicts between *dxapi.ocx* and the Visual Basic application. The following code summary shows the process:

```
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

    TextBox1.Invoke(New Delegate_OnConnection(AddressOf DlgOnconnection), _
        New Object() {e})
End Sub

Public Sub DlgOnconnection(ByVal e As AxdxapiLib._DdxapiEvents_OnConnectionEvent)

    PrintText("OnConnection Event", TO_SCREEN_AND_LOG)
    PrintText([String].Concat("Result Code:", e.result_code), TO_LOG)
End Sub
```

The abovementioned problem does not exist in other languages like Visual C++ or Visual C#, by design.

11.3.4 View Fields and View Fields in Depth buttons

The button **View Fields** is an example of how to access all the fields in a message, and the button **View Fields in Depth** shows that there are detailed fields that depend on the previous fields. The essential part of the code is the following:


```

Dim fieldvalue As Object
Dim fieldstr As String

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

'Get the first field name
fieldstr = Axdxapil.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 = Axdxapil.GetField(fieldstr)

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

```

The results of the execution are passed to the windows, and appear as follows:

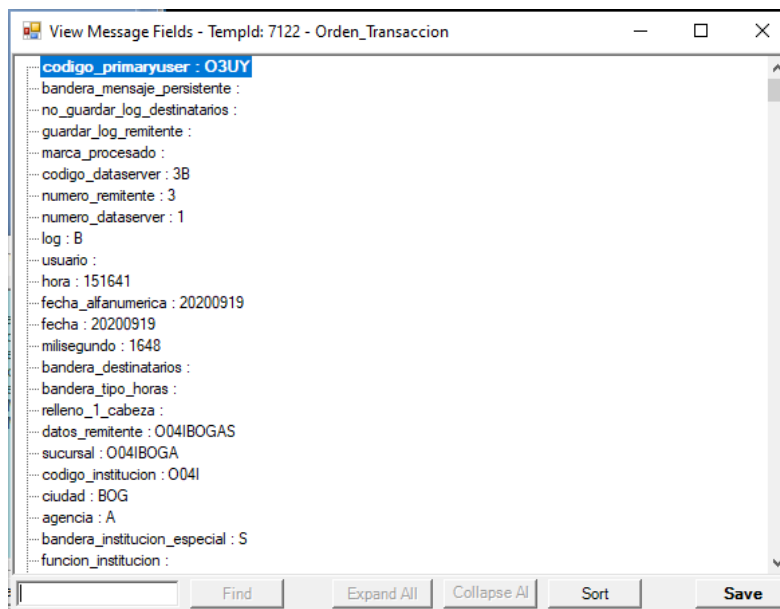


Figure 16 View Fields Button Window

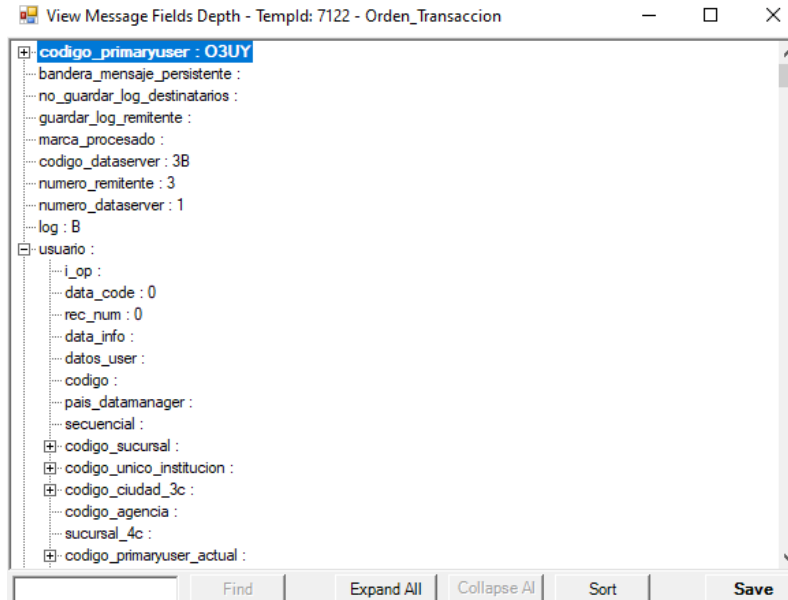


Figure 17 View Fields in Depth Button Window

11.3.5 Set As Processed button

The button **Set As Processed** calls the method **FinishMessageProcess**, necessary in order that the DAPI can mark the message as having been processed, to free the resources reserved for this process, and to avoid the re-processing of this message in the event of a re-connection.²

12 DAPI Linux installation

Linux distribution contains the following files:

- libdapi.a (static library).
- libdapi.so (dynamic library).
- sample\lnxdrvapitest_a.zip (static library sample).
- sample\lnxdrvapitest_so.zip (dynamic library sample).

12.1 Default configuration under Linux

The DAPI (Linux) has been generated and tested in the following environments:

- Red Hat Enterprise Linux Server release 5
- Red Hat Enterprise Linux Server release 6.8
- Red Hat Enterprise Linux Server release 7.1
- gcc (GCC) 4.8.3.
- OpenSSL 1.1.1.b

12.2 OpenSSL

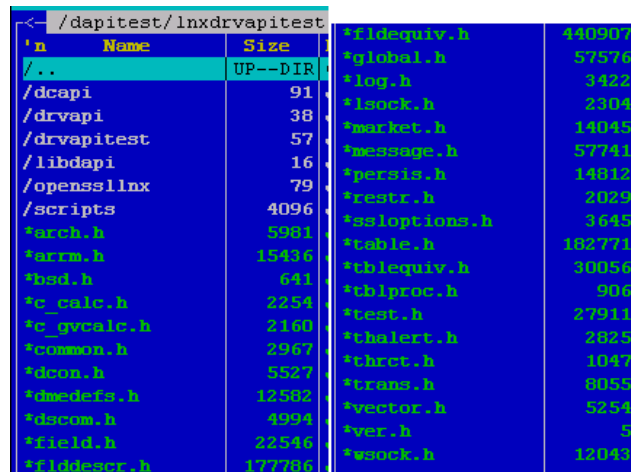
It is not required to compile the OpenSSL library

12.3 Installation and use of the Linux version of DAPI

After selecting the Linux platform to use (whether it be 5, 6,8 or 7) unpackage the Inxdrvapitest.zip the Inxdrvapitest directory and under this folder you will find several files which will be detailed below.

The DAPI libraries for Linux are **libdapi.a** and **libdapi.so**, they are distributed as precompiled file using the gcc compiler version indicated previously. Therefore the application should be linked with this library to use the DAPI methods.

The header file for DAPI is dcapi.h and it is located in: **Inxdrvapitest/dcapi/dcapi.h**



n	Name	Size
/..	UP--DIR	
/dcapi		91
/drvapi		38
/drvapitest		57
/libdapi		16
/opensslnx		79
/scripts		4096
*arch.h		5981
*arm.h		15436
*bad.h		641
*c_calc.h		2254
*c_gvcalc.h		2160
*common.h		2967
*deon.h		5527
*dsedefs.h		12502
*dscum.h		4994
*field.h		22546
*fidescr.h		177706
*fidequiv.h		440907
*global.h		57576
*log.h		3422
*lsock.h		2304
*market.h		14045
*message.h		57741
*persis.h		14812
*restr.h		2029
*ssloptions.h		3645
*table.h		182771
*tblequiv.h		30056
*tblproc.h		906
*test.h		27911
*tblert.h		2025
*thrt.h		1047
*trans.h		8055
*vector.h		5254
*ver.h		5
*wsock.h		12043

Figure 18 Dapi Linux structure

12.4 Building the library in Linux

It is not required to compile the DAPI library as it is already distributed. Additionally, the source code of this library is not distributed.

12.5 Building the Linux sample (drvapitest)

The distribution kit for the DAPI(Linux) includes an already compiled example that uses the **libdapi.a** or **libdapi.so** libraries.

If you need recompile the drvapitest program, you should install:

- gcc 4.8.3
- glibc-devel-2.17-78.el7.i686.rpm

In order to recompile the sample code, issue the following commands:

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

Note that the generation of the DAPI and its example has been done using scripts. The use of scripts implies that these should be in same format as the shell would expect them to be in, which is the Unix format.

If a problem is encountered while executing the scripts which points to incompatibility of the formats, the following instructions may be used, which would convert the *mk* script to Unix format:

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

The script formats could be affected if copy operation is done from a Windows operating system.

12.6 Possible error messages during the compilation process

If the required gcc libraries are not installed in the host used for compiling the sample code, it is possible that the *mk* script throws errors such as:

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

Figure 19 Sample error message when gcc libraries are not found

To install the 32-bit compatibility libraries use the **glibc-devel-2.5-81.i386** RPM library. In order to install this package issue the yum command.

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

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

Figure 20 Using yum to install compatibility libraries

When compiling sample code in a 64-bit OS host make sure that the correct *libdapi.a* library is installed. That is, the 32-bit DAPI library compatible with a 64-bit OS. If the incorrect DAPI library is use, the following error may arise:

```
[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/drvapitest
-> /home/datatec/tmp/DAPI/2012/sample/linux/lnxdrvapitest/drvapitest/xrelease/drvapitest
/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<CIdKey const, int>>, std::default_alloc_template<true, 0>>::deallocate<std::Rb tree node<std::pair<CIdKey const, int>>>*, 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<CFixedIncomeKey const, int>>, std::default_alloc_template<true, 0>>::deallocate<std::Rb tree node<std::pair<CFixedIncomeKey const, int>>>*, 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<CMarketKey const, int>>, std::default_alloc_template<true, 0>>::deallocate<std::Rb tree node<std::pair<CMarketKey const, int>>>*, unsigned int>':
```

Figure 21 Sample error message when wrong libdapi.a version is used

The following screenshot shows a successful sample code compilation in a 32-bit OS host.

```
datatec@support05:~/tmp/DAPI/2012/sample/linux/lnxdrvapitest/scripts
[datatec@support05 scripts]$ ./mk drvapitest clean
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.
Deleting object files
Deleting library object files
Making Completed
[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
Making Completed
[datatec@support05 scripts]$
```

Figure 22 Successful compilation of the sample program

12.7 drvapitest - Linux example using the DAPI

The DAPI distribution for Linux includes an example showing the usage of the *libdapi.a* and *libdapi.so* libraries. The library is called by the *drvapitest* executable (C++ source code is also included). To ensure that the application runs properly it is necessary to follow the procedure detailed below:

1. Configure the DAPI user with the DataManager Client.
2. Update the *dapi.cfg* configuration file with the DAPI user information and the correspondent DataServers a los que puede conectarse.
3. Ensure that *dapi.cfg* file is located in the same directory where the application *drvapitest* is going to be executed.

The executable of DAPI simple is under the directory *lnxdrvapitest/drvapitest/xrelease*. Therefore, it is necessary to go to this directory and then execute the application:

```
[myuser@ xrelease]$ ./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.

```

Figure 23 Execution of the sample program

13 Message descriptions

In an accompanying Excel file each message to be used by the DAPI is described in detail using a separate spreadsheet.

The Excel file description of each message uses the following columns:

Nombre del campo

The field name in Spanish. This is the field name that must be used when referencing a field in the DAPI program. In a future version you will also be able to use the field name in English.

Field name

The field name in English.

A few fields are redefined as two or more sub-fields. In this case the master field names are in BLUE type, and the sub-fields are indented by four characters. The DAPI can use either the master field name to manage all the characters as one string, or individual sub-fields can be referenced to manage a sub-string.

Field type

The field type can be:

alphanumeric

Any alphanumeric character or " " (space).

alphabetic

Any letter A-Z, a-z, or " " (space).

time

A time expressed in format HHMMSS

date

A date expressed in format YYYYMMDD

short

Signed integer occupying two bytes, -32,768 to 32,767.

long

Signed integer occupying four bytes, -2^{31} to $(2^{31} - 1)$

numeric ascii

A number stored in ascii character format, using a decimal point if there are decimals

numeric ascii signed

A number stored in ascii character format, using a decimal point if there are decimals, and with a leading negative sign if the number is negative.

Length

The total length of the field. If the field is *numeric ascii* and has decimals, the length is the sum of the digits, the decimals, plus one character for the decimal point. If the field is *numeric ascii signed*, the length of the field is the sum of the digits, the decimals plus one (if there are decimals), plus one for the sign.

Digits

The number of digits of a *numeric ascii* field.

Decimals

The number of decimals of a *numeric ascii* field.

Descripción

The field description in Spanish.

Description

The field description.

14 Error Code List

The following table contains the error codes reported by the DAPI application.

Code	Error identification
1	DAPI_ERR_INTERNAL_ERROR
	Error received by the DAPI from the operating system. Check the last operation attempted by the DAPI
2	DAPI_ERR_CONFIG_NOT_FOUND
	The specified configuration file can not be opened. Check if the file exists and check that the application has read permission for this file.
3	DAPI_ERR_CONFIG_BAD_DATA
	Syntax error in the configuration file. Check the specified line.
4	DAPI_ERR_NOT_INITIALIZED_YET
	A method call has been attempted, but the <i>login</i> method has not been called before.
5	DAPI_ERR_REJECTED_CONNECTION
	Datatec server has rejected the connection attempt, for an unknown reason. Contact operations support.
6	DAPI_ERR_RC_NOT_AUTHORIZED_USER (X)*
	The Datatec server has rejected the connection attempt as the user is not authorized.
7	DAPI_ERR_RC_USERNAME_DONOT_EXIST (N)*
	The Datatec server has rejected the connection attempt because the user name used is not registered in the Datamanager.

8	DAPI_ERR_RC_ILLEGAL_PASSWORD (C)*
	The Datatec server has rejected the connection attempt because the wrong password was sent.
9	DAPI_ERR_RC_NO_DMACHINES_TRY_LATER (D)*
	The Datatec server has rejected the connection attempt because there is no Datamanager on line to validate the user.
10	DAPI_ERR_RC_DUPLICATED_MACHINE_CODE (R)*
	The Datatec server has rejected the connection attempt because there is another user connected using the same code.
11	DAPI_ERR_RC_MACHINE_NOT_REGISTERED (E)*
	The Datatec server has rejected the connection attempt because the code of the computer from which the attempt was made is not registered in the Datamanager.
12	DAPI_ERR_RC_ILLEGAL_BRANCH_PASSWORD (L)*
	The Datatec server has rejected the connection attempt as the branch password is incorrect.
13	DAPI_ERR_STATUS_NOT_EXPECTED
	The connection status has changed to an undefined status. Report this issue to operations support.
14	DAPI_ERR_IMPROPER_OPERATION
	Parameters passed to a DPI method have invalid values. The error detail explains more about the origin of this problem.
15	DAPI_ERR_ILLEGAL_RECORD_TYPE_SPECIFIED
	The name of the record used as an argument for a method is invalid.
16	DAPI_ERR_ILLEGAL_FIELD_NAME_SPECIFIED
	The name of the field used as an argument for a method is invalid.
17	DAPI_ERR_ALREADY_INITIALIZED
	Occurs when Login method is called when the DAPI is already connected .
18	DAPI_ERR_RECORD_NOT_FOUND
	The number or identification of a requested message does not exist. Either the parameter is incorrect or the message has been discarded using <i>FinishMessageProcess</i> .
19	DAPI_ERR_CONDITION_EVALUATION
	DAPI can not evaluate the reception conditions given in the <i>SetLoginCondition</i> method. Check the number of operators and that the operations are in the BNF format.
20	DAPI_ERR_COULDNT_USE_LOG_FILES
	The selected log file can not be opened. Check the access privileges for the execution directory.
21	DAPI_ERR_NO_DS_DIRECTIVE
	The configuration file does not define any DataServers. This is necessary in order to be able to log on to the system.
22	DAPI_ERR_ILLEGAL_ADDRESS_TYPE_SPECIFIED
	The specified address type is invalid. Check the addresses.

* The letter in parenthesis is the value received from the DataServer at the end of a failed Login attempt.