ASX Trade Refresh

ASX Trade Open Interface Function Calls

December 2020

Table of Contents

1 Introduction 5

1.1 Software Distribution Restrictions 5
1.2 Supported Platforms 5
1.3 ASX Trade Support 5
1.4 ASX Trade OI Documentation Suite 5
1.5 Restrictions 6

1.5.1 Trading Participant Specific Information 6
1.5.2 Broker Service Providers 6

1.6 Version History 7

2 Error Messages 8

2.1 Error Values Returned from Function Calls 8
2.2 Error Messages Returned from Transactions, Queries and Broadcasts 11

3 OI Function Calls 12

3.1 omniapi_create_session – Create a Session with the OI 12

3.1.1 Format 12
3.1.2 Returns 12

3.2 omniapi_close_session – Close Session with the OI 12

3.2.1 Format 12

3.3 omniapi_login_ex – Login to ASX Trade 12

3.3.1 Format 12
3.3.2 Arguments 12
3.3.3 The omni_login_t Definition 13
3.3.4 Returns 13
3.3.5 Return Values 13
3.3.6 Calling Example 14

3.4 omniapa_logout_ex – logout from ASX Trade 15

3.4.1 Format 15
3.4.2 Arguments 15
3.4.3 Returns 15
3.4.4 Return Values 15
3.4.5 Calling Example 15

3.5 omniapi_set_newpwd_ex – Set a New Password 16

3.5.1 Format 16
3.5.2 Arguments 16
3.5.3 The omni_set_password_t Definition 16
3.5.4 Returns 17
3.5.5 Return Values 17
3.5.6 Calling Example 17

3.6 omniapi_tx_ex – Issue a Transaction 18
3.6.1 Format 18
3.6.2 Arguments 18
3.6.3 Returns....18
3.6.4 Return Values....19
3.6.5 Calling Example 19
3.7 omniapi_query_ex - Issue a Query....19
3.7.1 Format 19
3.7.2 Arguments 19
3.7.3 Returns....20
3.7.4 Return Values....21
3.7.5 Calling Examples....21
3.8 omniapi_set_event_ex – Request Subscription of Broadcasts....21
3.8.1 Format 21
3.8.2 Description....21
3.8.3 Arguments 21
3.8.4 Examples of Information Objects 24
3.8.5 Returns....25
3.8.6 Return Values....25
3.8.7 Calling Examples....25
3.9 omniapi_clear_event_ex – Cancel Event Subscription....27
3.9.1 Format 27
3.9.2 Description....28
3.9.3 Arguments 28
3.9.4 Returns....28
3.9.5 Return Values....29
3.9.6 Calling Examples....29
3.10 omniapi_read_event_ext_ex – Read Events....29
3.10.1 Format 29
3.10.2 Description....29
3.10.3 Arguments 29
3.10.4 Setting up and Clearing Subscriptions....31
3.10.5 Setting up Subscriptions....31
3.10.6 Clearing Subscriptions....32
3.10.7 Hints....32
3.10.8 omniapi_read_event_ext_ex - Event Types 32
3.10.9 Returns....33
3.10.10 Return Values....33
3.10.11 Calling Example 34
3.11 omniapi_read_event_block – Blocking Read Event 35
3.11.1 Description....35
3.11.2 Arguments 36
3.11.3 Returns....36
3.11.4 Return Values....36
3.11.5 Calling Example 37

3.12 omniapi_get_info_ex – Get Environment Information 37
3.12.1 Format 37
3.12.2 Description 37
3.12.3 Arguments 38
3.12.4 Returns 39
3.12.5 Return Values 39
3.12.6 Calling Example 39
3.13 omniapi_get_message_ex – Get an Exchange Message 39
3.13.1 Format 39
3.13.2 Description 40
3.13.3 Arguments 40
3.13.4 Returns 40
3.13.5 Return Values 40
3.13.6 Calling Example 40
3.14 omniapi_set_option_ex – Set Options for OI Session 41
3.14.1 Format 41
3.14.2 Description 41
3.14.3 Arguments 41
3.14.4 Encryption 41
3.14.5 Compression 42
3.14.6 Concurrent Broadcasts 42
3.14.7 Returns 42
3.14.8 Return Values 42
3.15 omniapi_set_option_default – Set Default Values for OI Session 43
3.15.1 Format 43
3.15.2 Description 43
3.15.3 Arguments 43
3.16 omniapi_cvt_int – Convert an Integer 43
3.16.1 Format 43
3.16.2 Description 43
3.16.3 Arguments 43
3.17 omniapi_cvt_string – Convert a String to/from the Central Format 43
3.17.1 Format 44
3.17.2 Description 44
3.17.3 Arguments 44
3.17.4 Example 44
3.18 omniapi_convert_timestruct – Convert Timestructs 44
3.18.1 Format 44
3.18.2 Description 44
3.18.3 Arguments 44
3.18.4 Return Values 45
4 Appendix 1 – Concurrent Broadcast Feature 46

1 Introduction

ASX Trade contains Open Interface (OI) function calls that are used to enable communication between ASX Trade and the participant's site via a gateway.

There are 18 OI function calls in ASX Trade. These include:

- omniapi_create_session – Create a session with the OI
- omniapi_close_session – Close session with the OI
- omniapi_login_ex – Login to ASX Trade
- omniapi_logout_ex – Logout from ASX Trade
- omniapi_set_newpwd_ex – Set a new password
- omniapi_tx_ex – Issue a Transaction
- omniapi_query_ex - Issue a Query
- omniapi_set_event_ex – Request subscription of broadcasts
- omniapi_clear_event_ex – Cancel event subscription
- omniapi_read_event_ext_ex – Read events
- omniapi_read_event_block – Blocking read event
- omniapi_get_info_ex – Get environment information
- omniapi_get_message_ex – Get an exchange message
- omniapi_set_option_ex – Set options for OI session
- omniapi_set_option_default – Set default values for OI session
- omniapi_cvt_int – Convert an integer
- omniapi_cvt_string – Convert a string to/from the central format
- omniapi_convert_timestruct – Convert timestructs.

1.1 Software Distribution Restrictions

Restrictions on the distribution of the OI software are detailed in the Developer's Agreement.

1.2 Supported Platforms

The following platforms are supported by ASX Trade:

- Linux Redhat Rhel6.10 x86 (32 and 64 bit)
- Linux Redhat Rhel7 x86 (32 and 64 bit)
- Windows 6.3 x86 (Windows Server 2012 R2 - 32 and 64 bit)
- Windows 10 x86 (Windows Server 2016 - 32 and 64 bit) ASX

1.3 Trade Support

For ASX Trade Open Interface Support, contact ASX Customer Technical Support team either via email on cts@asx.com.au or phone 1800 663 053 (or on +61 2 9227 0372 from outside Australia).

1.4 ASX Trade OI Documentation Suite

ASX Trade Open Interface documentation has been created as a suite of documents that reference each other. The suite of documentation includes the following documents:

- ASX Trade Introduction and Business Information – This includes an introduction to ASX Trade for Open Interface developers and application providers. It also details business functionality to enable ASX Trade to be fully utilised.
- ASX Trade Open Interface Function Calls – This details the Open Interface function calls that enable communication between ASX Trade and the participant.

- ASX Trade Transactions – This contains the transactions that are used to instruct ASX Trade to perform particular actions.
- ASX Trade Queries – This details the queries that are used to retrieve information from ASX Trade.
- ASX Trade Broadcasts – This includes the broadcasts that are used to notify participants of an event or change occurring in ASX Trade.

1.5 Restrictions

Certain confidential information is prescribed by ASX as ‘restricted information’. Details of what constitutes restricted information are set out below.

Some ASX Trade information is restricted information and may not be divulged to anyone who is not a Designated Trading Representative (DTR), except where that person is employed by an ASX Trading participant and has a need to access that data as part of their duties.

1.5.1 Trading Participant Specific Information

Trading Participant Specific Information is the information specific to the trading participant that instigated a transaction on ASX Trade and which is not distributed by ASX to other participants. Trading Participant Specific Information must not be divulged to anyone who is not a Designated Trading Representative of the trading participant, except where the person is employed by the trading participant and that person has a need to access that data as part of their duties.

Trading participant specific information includes, but is not limited to, the following:

- Client and Info references on orders and trades
- Total quantity for Iceberg orders and undisclosed quantities on orders
- The unique identifier of a trading participant allocated by ASX, i.e. the trading participant number, or the participant name in relation to Products other than Listed Funds, Warrants and Structured products, Exchanged Traded Options and Futures
- Some order types, e.g. shortsell
- Signum (user/session) on orders and trades
- Expiry dates on orders
- Centre Point orders
- The short sell information on orders and trades
- Regulatory data
- Certain trade types e.g. BP (Booking Purpose); LN (Loan); LR (Loan Return)
- Booking reports resulting from Unintentional Crossing Prevention.

Participant Specific information is blanked out in enquiries where the order or trade does not belong to your trading participant ID.

1.5.2 Broker Service Providers

The trading participant may use dealing/information systems provided by an information vendor.

If the trading participant requests, ASX can provide the vendor with:

- All of the trading participant’s specific information as detailed in Trading Participant Specific Information above.
- The vendor can then integrate this information into their dealing/information systems for the trading participant.
- A vendor that has access to Trading Participant Specific information is known as a Broker Service Provider (BSP).
- The BSP must keep this Trading Participant Specific Information confidential and must not collate or distribute this information to anyone other than the relevant trading participant.

1.6 Version History

This document has been revised according to the table below:

Version	Date	Comment
v2.1	October 2018	• Updated to new ASX branding
v3.0	September 2019	• Updated for ASX Trade Refresh
v3.1	January 2020	• BI9 description updated (in sections 3.9 and 3.10) to a non-forced subscription
v3.2	June 2020	• Updated omniapi_get_message_ex encryption details
v3.3	August 2020	• Appendix 1 – Concurrent Broadcast Feature updated • omniapi_login_ex – Login to ASX Trade return values updated
v3.4	December 2020	• Appendix 2 - Environment Variable OAPI_TIMEOUT added

2 Error Messages

2.1 Error Values Returned from Function Calls

The following error values are common codes that can be returned from a function call.

Error Value	Error Name	Description
2028	OMNIAPI_PWD_IMPEND_EXP	Login succeeded, but password will expire in the coming days.
2027	OMNIAPI_PWD_CHANGE_REQ	Password has expired. The application will only get a restricted access to the system until the password has been changed successfully.
-1	OMNIAPI_FAILURE	Failure completion. Internal error in the OI.
-2	OMNIAPI_NOT_READY	OAPI not ready. The OI is not ready for an OMNIAPI call.
-3	OMNIAPI_FACID_NOT_VALID	Facility ID not in range. An invalid facility type, most likely zero, was used during an omniapi_tx_ex or omniapi_query_ex call.
-4	OMNIAPI_INVALID_TABLE	Invalid table type. The transaction syntax and verification subsystem has received an invalid transaction syntax table.
-5	OMNIAPI_NOT_LOGGED_IN	OAPI not logged in. The gateway considers the user as logged out.
-9	OMNIAPI_NOT_INITIALIZED	OAPI not initialised. Internal OI error code.
-10	OMNIAPI_NO_INFO_RCVD	No network information received. Internal OI error code.
-11	OMNIAPI_NOBACKEND	Backend communication failure. There are communication problems with the central system.
-12	OMNIAPI_TX_ABORTED	Transaction aborted. The transaction was aborted. The following (TXSTAT) code explains the cause.
-13	OMNIAPI_TRUNCATED	Data truncated. A query message response message was truncated.
-14	OMNIAPI_CNV_NO_RANGE	Table conversion range error. Internal OI error code.
-15	OMNIAPI_CNV_NOT_SORTED	Table conversion sort error. Internal OI error code.
-16	OMNIAPI_CNV_OFFS_ERROR	Table conversion offset error. Internal OI error code.
-17	OMNIAPI_NO_SUCH_ID	Invalid transaction type. The transaction type of the transaction is incorrect or unauthorised.
-18	OMNIAPI_VER_FIELD_ERROR	Transaction verification error. The transaction contents (after the transaction type) are incorrect or not recognised.
-19	OMNIAPI_VER_INT_ERROR	Message verification internal error
-20	OMNIAPI_VER_TABLE_ERROR	Table verification error. Internal OI error code.
-21	OMNIAPI_TX_TIMEOUT	Transaction timeout. This may have been caused by workload, network problems or a backend problem.
-22	OMNIAPI_TX_DECLFAIL	Transaction requester declaration failure. A connection to a particular facility failed.
-23	OMNIAPI_TX_FAILURE	Transaction failure. The following (TXSTAT) code explains the cause.
-24	OMNIAPI_DYNMEM	Error obtaining dynamic memory. Memory allocation error in the gateway.
-25	OMNIAPI_INVARG	Invalid argument. An invalid facility type argument was specified to omniapi_tx_ex or omniapi_query_ex routines.
-26	OMNIAPI_NOT_FOUND	Requested data not found. Buffer is empty. No more events have been received or returned from an omniapi_read_event_ext_ex

Error Value	Error Name	Description
		call. Message also occurs when omniapi_get_info_ex is called with an invalid information request.
-27	OMNIAPI_ITV_ERROR	Information table verification error. Internal OI error code.
-28	OMNIAPI_NO_USR_OR_PASSW	Username and/or password missing. The username or the password was not provided with the login request.
-29	OMNIAPI_NO_NET_PATH	Net path data missing. The gateway could not identify its network path.
-30	OMNIAPI_INVEVT	Invalid event type. An invalid event type was specified in a call to omniapi_clear_event_ex.
-32	OMNIAPI_INVTXTYPE	Invalid transaction type. A zero was passed as facility type to an omniapi_tx_ex or omniapi_query_ex call.
-35	OMNIAPI_FATAL	Fatal OAPI error. Internal OI error code.
-39	OMNIAPI_NOTAUTH	The user is not authorised to perform the requested action.
-40	OMNIAPI_PASSW_EXPIRED	The password has expired.
-41	OMNIAPI_INVALID_PASSW	Password is not valid. The provided new password is not valid. The new password breaches the password restrictions set by the exchange. The reason for this may be that the password is shorter than the minimum number of required characters or that the password has already been used during the last few months.
-50	OMNIAPI_ENDIAN_ERROR	Endian error, byte-swap failed. Error when byte-swapping the transaction or query in the gateway.
-53	OMNIAPI_NETERR_LINK	Network error between client and gateway.
-54	OMNIAPI_NETERR_TIMEOUT	Timed out while waiting for results from the gateway.
-55	OMNIAPI_CONCUR_ERROR	Error in concurrent connection. The OI has lost concurrent connection with the gateway.
-56	OMNIAPI_CONCUR_DISABLE	Concurrent connection is currently disabled.
-57	OMNIAPI_ERROPTCONC	Concurrent connection not supported.
-2000	OMNIAPI_PROBLEM	Error in OMNIAPI internal call. Internal OI error code.
-2001	OMNIAPI_INTFAILURE	Internal OMNIAPI error. Internal OI failure in the OMnet API library.
-2003	OMNIAPI_BADNARGS	Bad number of arguments. An incorrect number of arguments were passed to the OMNIAPI routine.
-2004	OMNIAPI_BADARGVAL	Bad argument value. At least one argument contains an incorrect value.
-2005	OMNIAPI_NONETWORK	The network software is unavailable for inter process communication.
-2006	OMNIAPI_OSBADCONFIG	The operating system is incorrectly configured.
-2008	OMNIAPI_NOTCONNECTED	Invalid operation before login. The OI application has no network link to the gateway.
-2009	OMNIAPI_NOGWYSRV	Unable to connect to gateway. Possible causes are the host or socket name is incorrect, the gateway does not exist or a previous session with the gateway was incorrectly terminated.

Error Value	Error Name	Description
-2010	OMNIAPI_BADHOSTNAME	Host name could not be translated. An unknown or syntactically incorrect host name for the gateway node was specified in a login request.
-2011	OMNIAPI_ERRSOCKET	Error assigning a local socket.
-2012	OMNIAPI_ERRCONNECT	System error occurred during a login request.
-2013	OMNIAPI_ERRBIND	System error occurred during a socket bind operation.
-2014	OMNIAPI_NOSESSION	Session is aborted. The network link to the gateway was disconnected.
-2015	OMNIAPI_ERRSEND	Error on send(). An error occurred during a send operation.
-2016	OMNIAPI_ERRMEM	Error on malloc(). An error occurred when allocating dynamic memory.
-2017	OMNIAPI_APIOLD	OMNIAPI too old for the gateway. The OI application is linked with an old unsupported version of the OMnet API library.
-2019	OMNIAPI_SESINUSE	Gateway OI session already in use. There is an ongoing OI call for the session, or a previous session has been incorrectly terminated.
-2020	OMNIAPI_ERROPTCMPZ	Compression option rejected. The requested compression option was rejected by the gateway.
-2021	OMNIAPI_ERROPTENCRYPT	Encryption option rejected. The requested encryption option was rejected by the gateway.
-2022	OMNIAPI_ERR_SECURE	Failed to establish secure link between the OI client application and the gateway.
-2023	OMNIAPI_LOGIN_DISABLED	User login has been disabled in the central system.
-2024	OMNIAPI_INVALID_LOGIN_ATTEMPT	Invalid login attempt due to a user setup related problem.
-2025	OMNIAPI_OMNBE_BUSY	Please wait - backend is busy. Central processes are busy and cannot handle user login requests at the moment.
-2026	OMNIAPI_LOGIN_FAILED	Communications/database problems. There may have been problems to communicate with the central system, or internal database problems caused the login to fail.
-2029	OMNIAPI_INVALID_CHRS	The new password contains invalid characters. Characters 0x01 to 0x19, “%” and “\” are not allowed in passwords.
-2030	OMNIAPI_NO_NULL_TERMINATION	String has to be null-terminated.
-2031	OMNIAPI_WRONG_TX_SIZE	The provided buffer size is not correct.
-2032	OMNIAPI_CALL_NOT_SUPPORTED	The call is not supported by the exchange.
-2033	OMNIAPI_INVALID_LOGIN_LOCKED	Login denied. User account is locked.
-2034	OMNIAPI_INVALID_LOGIN_SUSPENDED	Login denied. User account is suspended.
-2035	OMNIAPI_RELOGIN_NOT_ALLOWED	Login denied, re-login is not allowed. There is an already on-going session with this user account. The exchange has configured that re-login is not allowed.
-2036	OMNIAPI_NOT_APPLICABLE	The requested information is not applicable to the user or the session.

Error Value	Error Name	Description
-2037	OMNIAPI_INVALID_LOGIN_INACTIVE	Login denied. User account is inactivated.
-2038	OMNIAPI_INVALID_BEF_CHG_PWD	Invalid request before password is changed. The user's password has expired. The user will have restricted access to the system until the password has been changed successfully.
-2043	OMNIAPI_PART_NOT_FOUND	Backend partition not found.
-2044	OMNIAPI_PART_NOT_REACHABLE	Backend partition not reachable for the given transaction.
-2045	OMNIAPI_CONCUR_EXISTS	Concurrent connection already exists for the session.
-2046	OMNIAPI_INVALID_IDENT	Invalid concurrent identifier, session may be lost. An invalid session was encountered by the gateway when establishing a concurrent connection.

2.2 Error Messages Returned from Transactions, Queries and Broadcasts

Refer to *Genium INET – System Error Messages Reference* for details on why transactions, queries and broadcasts are being aborted. This is located at: https://www.asxonline.com/public/documents/asx-trade-refresh-manuals.html

3 OI Function Calls

The following OI function calls can be used in ASX Trade.

3.1 omniapi_create_session – Create a Session with the OI

This routine enables the use of multiple sessions in the same process. Each session with the OI must log in with a separate user ID.

3.1.1 Format

```
omniapi_session_handle hSession = omniapi_create_session()
```

3.1.2 Returns

hSession

hSession returns a handle to be used for all other omniapi function calls.

Note:
It is important to close the session before opening a new session. Each session occupies system resources within the OI that are freed when it is closed.

3.2 omniapi_close_session – Close Session with the OI

This routine closes the session with the OI and the socket used. It should be called after logout.

3.2.1 Format

```
void omniapi_close_session(
omniapi_session_handle hSession) // in
```

hSession

This must have been created using the omniapi_create_session() function.

Note:
It is important to log out before closing the session; otherwise the resources created for the login at the gateway will not be released immediately.

3.3 omniapi_login_ex – Login to ASX Trade

The omniapi_login_ex routine is used to log in to ASX Trade.

3.3.1 Format

```
int32 cstatus = omniapi_login_ex(
omniapi_session_handle     hSession,     // in
int32 *                    txstat,       // out
omni_login_t *             login_data)   // in
```

3.3.2 Arguments

hSession

This argument must have been previously created with the omniapi_create_session call.

txstat

This argument is used to get further information about the routine completion.

If cstatus	txstat contains
differs from OMNIAPI_SUCCESS	If txstat differs from zero, a secondary status code from the trading system or gateway is present.

login_data

The omni_login_t structure has the following layout. Note that all strings must be NULL terminated.

```
typedef struct
{                                          /* OMnet login data structure */
   omni_username    user_s;                /* user identification */
   omni_password    pass_s;                /* password */
   char             gateway_node_s[128];   /* gateway node name or ip-number */
   uint32_t         port_u;                /* gateway port number */
   char             appl_ident_s[32];      /* program identification */
   uint8_t          forced_u;              /* LOGIN_NORMAL=normal login, */

                                           /* LOGIN_FORCED=forced login */
   char             filler_3_s[3];         /* filler */
} omni_login_t;
```

3.3.3 The omni_login_t Definition

Type	Name	Size	Mandatory	Description
omni_username	user_s	32*	Yes	The username string. Must be NULL terminated.
omni_password	pass_s	32*	Yes	The user password string. Must be NULL terminated.
char	gateway_node _s	128*	Yes	Gateway host/node name or TCP/IP address of the gateway executor node. Must be NULL terminated.
uint32_t	port_u	4	Yes	Port number of the gateway process. This integer is required to remain in native endian format. Do **not** byte swap this number.
char	appl_ident_s	32*	No	Program identifier of the application. For example: “QUICKTRADE V24-3”. Must be NULL terminated.
uint8_t	forced_u	1	Yes	Flag for normal/forced login. If a login error is received indicating that a user is already logged on, setting this flag to one overrides the existing user and forces them to be disconnected.

*Includes NULL termination, i.e. ‘\0’

3.3.4 Returns

cstatus

The completion code is used for checking the operation of the call. If cstatus is negative, the call has failed. The completion code gives an indication of the problem. If cstatus is OMNIAPI_SUCCESS or positive then the operation was successful. In either case, the txstat argument may hold more detailed information.

3.3.5 Return Values

If the error OMNIAPI_PWD_CHANGE_REQ is returned from the login function, the session is only able to execute the following functions until the password is changed (see *omniapi_set_newpwd_ex – Set a New Password*):

- omniapi_get_info_ex

- omniapi_get_message_ex
- omniapi_read_event_ext_ex
- omniapi_set_newpwd_ex
- omniapi_logout_ex.

After the password has been successfully changed, full functionality is available.

If the error OMNIAPI_PWD_IMPEND_EXP is returned, it indicates that the password will expire within a certain number of days. The precise number of days till expiry may be retrieved using the omniapi_get_info_ex function call. For more information see *omniapi_get_info_ex – Get Environment Information*.

The following error values are specific codes that omniapi_login_ex can produce.

Error Code	Error Message	Description
-28	OMNIAPI_NO_USR_OR_PASSW	Username and/or password missing. The username or the password was not provided with the login request.
2027	OMNIAPI_PWD_CHANGE_REQ	Password has expired. The application will only get a restricted access to the system until the password has been changed successfully.
2028	OMNIAPI_PWD_IMPEND_EXP	Login succeeded, but password will expire in the coming days.
-2031	OMNIAPI_WRONG_TX_SIZE	The provided buffer size is not correct.
-2032	OMNIAPI_CALL_NOT_SUPPORTED	The call is not supported by the exchange.
-2033	OMNIAPI_INVALID_LOGIN_LOCKED	Login denied. User account is locked.
-2034	OMNIAPI_INVALID_LOGIN_SUSPENDED	Login denied. User account is suspended.
-2035	OMNIAPI_RELOGIN_NOT_ALLOWED	Login denied, re-login is not allowed. There is an already on-going session with this user account. The exchange has configured that re-login is not allowed.

3.3.6 Calling Example

The following program excerpt demonstrates the use of the omniapi_login_ex call.

```
int32_t APISAMPLE_Login(char* username, char* password, char* node, char* port )
{
  omni_login_t      loginData;
  omni_password     newpassword;
  int32_t           cstatus;
  int32_t           txStatus;

  memset( &loginData, 0, sizeof(loginData) );
  memset( &newpassword, 0, sizeof(newpassword) );


  strncpy( loginData.user_s, username,     sizeof(loginData.user_s) );
  strncpy( loginData.pass_s, password,     sizeof(loginData.pass_s) );
  strncpy( loginData.gateway_node_s, node, sizeof(loginData.gateway_node_s) );
  loginData.port_u = atoi(port);
  strncpy( loginData.appl_ident_s, "APISAMPLE", sizeof(loginData.appl_ident_s) );
  loginData.forced_u = LOGIN_NORMAL;

  cstatus = omniapi_login_ex( hSession, &txStatus, &loginData );

  if (cstatus == OMNIAPI_SUCCESS)
  {
    printf("Successfully logged in\n");
  }
  else if (cstatus > 0)
```

3.4 omniapa_logout_ex – logout from ASX Trade

The omniapi_logout_ex routine is used to logout from ASX Trade. Once called, the session should be closed and not reused for subsequent logons.

3.4.1 Format

```
int32 cstatus = omniapi_logout_ex(
omniapi_session_handle     hSession,     // in
int32 *                    txstat)       // out
```

3.4.2 Arguments

hSession

This argument must have been previously created with the omniapi_create_session call.

txstat

This argument is used to get further information about the routine completion.

3.4.3 Returns

cstatus

The completion code is used for checking the operation of the call. If cstatus is negative, the call has failed. The completion code gives an indication of the problem. If cstatus is OMNIAPI_SUCCESS or positive then the operation was successful. In either case, the txstat argument may hold more detailed information.

3.4.4 Return Values

The following error value is a specific codes that omniapi_logout_ex can produce.

Error Code	Error Message	Description
-2008	OMNIAPI_NOTCONNECTED	Invalid operation before login. The OI application has no network link to the gateway.

3.4.5 Calling Example

The following program excerpt demonstrates the use of the omniapi_logout_ex call.

```
int32_t APISAMPLE_Logout()
{
   int32_t       cstatus;
   int32_t       txStatus;

   cstatus = omniapi_logout_ex ( hSession, &txStatus );

   if ( OMNIAPI_SUCCESS == cstatus )
   {
     printf("Successfully logged out\n");
   }
   else
   {
     printf("Error: Logout failed, completion status = %d\n", cstatus);
   }
   return cstatus;
}
```

3.5 omniapi_set_newpwd_ex – Set a New Password

The omniapi_set_newpwd_ex routine is used to set a new password.

This routine may be called any time after logging on to the system. It should be called if the error OMNIAPI_PWD_CHANGE_REQ is returned from the login function (see *omniapi_login_ex – Login to ASX Trade* for more information).

3.5.1 Format

```
int32_t cstatus = omniapi_set_newpwd_ex (
     omniapi_session_handle   hSession,       // in
     int32_t *                txstat,         // out
     omni_set_password_t *    pwd_data)       // in
```

3.5.2 Arguments

hSession

This argument must have been previously created with the omniapi_create_session call.

txstat

This argument is used to get further information about the routine completion.

If cstatus	txstat Contains
differs from OMNIAPI_SUCCESS	If txstat differs from zero, a secondary status code from the trading system or gateway is present.

pwd_data

```
typedef struct
{
       omni_password pass_s;
       omni_password new_pass_s;
}      omni_set_password_t
```

3.5.3 The omni_set_password_t Definition

Type	Name	Size	Mandatory	Description
omni_password	pass_s	32	Yes	The old password.
omni_password	new_pass_s	32	Yes	The new password.

The new password must meet the following criteria:

- The password string has to be NULL (‘\0’) terminated.
- The password must contain at least eight characters.
- The length of the password may not exceed 32 characters including the NULL termination character.
- The characters in the password must belong to the printable character set (valid characters are hex 0x20 – 0x7E except for 0x25 and 0x5C).
- There must be at least one alphabetic character from A-Z in the password.
- There must be at least one numerical character from 0-9 in the password.
- There must be at least one special character in the password.

Special characters include:

! @ # $ ^ & * ( ) - _ = + , < . > / ? ;: ' " [{] } |

- The password should not be found in the dictionary of common words.

- A password cannot be re-used within a period of 12 months.

Note:
'%' and '\' are **not** allowable characters in a password.

3.5.4 Returns

Cstatus

The completion code is used for checking the operation of the call. If cstatus is negative, the call has failed. The completion code gives an indication about the problem. If cstatus is OMNIAPI_SUCCESS or positive then the operation was successful. In either case, the txstat argument may hold more detailed information.

3.5.5 Return Values

The following error values are specific codes that omniapi_set_newpwd_ex can produce.

Error Code	Error Message	Description
-41	OMNIAPI_INVALID_PASSW	Password is not valid. The provided new password is not valid. The new password breaches the password restrictions set by the exchange. The reason for this may be that the password is shorter than the minimum number of required characters or that the password has already been used during the last few months.
-2029	OMNIAPI_INVALID_CHRS	The new password contains invalid characters. Characters 0x01 to 0x19, "%" and "\" are not allowed in passwords.
-2030	OMNIAPI_NO_NULL_TERMINATION	String has to be NULL terminated.

3.5.6 Calling Example

The following program excerpt demonstrates the use of the omniapi_set_newpwd_ex call:

```
int32_t APISAMPLE_SetNewPassword(char* currentPassword,
                                 char* newPassword )
{
   omni_set_password_t setNewpwd;
   int32_t             cstatus;
   int32_t             txStatus;

   memset( &setNewpwd, 0, sizeof(setNewpwd) );

   strncpy(setNewpwd.pass_s, currentPassword, sizeof(setNewpwd.pass_s));
   strncpy(setNewpwd.new_pass_s, newPassword, sizeof(setNewpwd.new_pass_s));

   cstatus = omniapi_set_newpwd_ex( hSession, &txStatus, &setNewpwd);

   if (cstatus >= OMNIAPI_SUCCESS)
   {
     printf("Successfully set a new password \n");
   }
   else
   {
     printf("Set new password failed. Completion status = %d\n", cstatus);
   }
   return cstatus;
}
```

3.6 omniapi_tx_ex – Issue a Transaction

The omniapi_tx_ex routine is used for entering the transactions defined in *ASX Trade Transactions.*

3.6.1 Format

```
    int32 cstatus = omniapi_tx_ex (
        omniapi_session_handle   hSession,     // in
        int32 *                         txstat,       // out
        uint32                          factyp,       // in
        omni_message **          txmsgs,       // in
        uint32 *                 txidnt,       // out
        uint32 *                 ordidt);      // out
```

3.6.2 Arguments

hSession

This argument must have been previously created with the omniapi_create_session call.

txstat

This argument is used to get further information about the routine completion.

If cstatus	txstat Contains
Is zero or positive	A secondary status code from the trading system.
Is negative	A code showing the cause of the failure, coming from the OI, the Enhanced Transaction Router (ETR) Messages or the trading system.

factyp

The facility type is described in *ASX Trade Introduction and Business Information - Facilities.*

txmsgs

This is a vector of pointers to message buffers (omni_message) that should be sent to ASX Trade. The vector must be NULL terminated. Currently, only one omni_message buffer is supported.

This field contains the message structures defined in *ASX Trade Transactions*.

txidnt

This argument references an eight byte block receiving a unique 64-bit identification of the transaction.

ordidt

The ordidt argument holds a reference to an 8 byte (64 bit) order identification number. Users need this entity to recognise matched orders when parsing broadcasts and queries. Users should note that the order identifier returned here is in native endian format, and therefore PUTORDERID macro must not be used. Refer to *ASX Trade Introduction and Business Information* for more information.

3.6.3 Returns

cstatus

The completion code is used for checking the operation of the call. If cstatus is negative, the call has failed. The completion code gives an indication of the problem. If cstatus is OMNIAPI_SUCCESS or positive then the operation was successful. In either case, the txstat argument may hold more detailed information.

3.6.4 Return Values

A list of common error codes can be found in *2 Error Messages*

3.6.5 Calling Example

The following program excerpt demonstrates the use of the omniapi_tx_ex call:

```
int32_t APISAMPLE_SendTx( void* buffer, uint32_t length, uint32_t facilityType )
{
  omni_message* message[2]; /* only use the first item */
  char sendBuffer[MAX_REQUEST_SIZE];
  int32_t retStatus;

  if( facilityType == 0 )
  {
    printf("Error! SendTx called with facilityType == 0.\n");
    return -1;
  }

  message[0] = (omni_message*)sendBuffer;
  message[1] = NULL;

  memcpy( &sendBuffer[0], &length, sizeof(length) );
  memcpy( &sendBuffer[sizeof(length)], buffer, length );

  retStatus = omniapi_tx_ex(
        hSession,
        &glTransactionStatus,
        facilityType,
        message,
        &glTransactionId[0],
        &glOrderId);

  return retStatus;
}
```

3.7 omniapi_query_ex - Issue a Query

The omniapi_query_ex routine is used for sending the query requests defined in *ASX Trade Queries*. It is used to retrieve information from ASX Trade.

3.7.1 Format

```
    int32 cstatus = omniapi_query_ex (
        omniapi_session_handle    hSession,     // in
        int32 *                          txstat,       // out
        uint32                           factyp,       // in
        omni_message *                   qrymsg,       // in
        int8                      retflg,       // in
        int8 *                           rcvbuf,       // out
        uint32 *                         rcvlen,       // in/out
        uint32 *                         txidnt,       // out
        uint32 *                         ordidt )      // out
```

3.7.2 Arguments

hSession

This argument must have been previously created with the omniapi_create_session() call.

txstat

This argument is used to get further information about the completion of the routine.

If cstatus	txstat Contains
Is zero or positive	A secondary status code from the trading system application.
Is negative	A code showing the cause of the failure, coming from the OI, ETR or the ASX Trade.

factyp

The facility type is described in *ASX Trade Introduction and Business Information*.

qrymsg

This argument is a reference to an omni_message buffer containing the query structure.

retflg

This argument must be set to one.

rcvbuf

This argument is a reference to a buffer about to receive the resulting message. This buffer must be allocated by the calling routine and is populated with an answer structure.

rcvlen

The receive length argument specifies the length of the buffer provided with the rcvbuf argument. This argument has two interpretations:

- On input - the length of the caller's provided buffer.
- On completion - number of bytes used in response.

Note:
When querying several segments, this argument must be reset to the size of the buffer between every call, since it is modified to the length of the data returned.

Note:
This is returned in native endian format.

txidnt

This argument references an 8 byte block receiving a unique 64 bit identification of the query.

ordidt

The ordidt argument holds a reference to an eight byte (64 bit) order identification number. This parameter is not utilised in the OI, and its value in this function call can be ignored.

3.7.3 Returns

cstatus

The completion code is used for checking the operation of the call. If cstatus is negative, the call has failed. The completion code gives a hint about the problem. If cstatus is OMNIAPI_SUCCESS or positive then the operation was successful. In either case, the txstat argument may hold more detailed information.

3.7.4 Return Values

A list of common error codes can be found in *2 Error Messages.*

3.7.5 Calling Examples

```
int32_t APISAMPLE_Query( void* qryBuffer, uint32_t qryLength, uint32_t
            facilityType, void* rcvBuffer, uint32_t* rcvLength )
{
   int8_t sendBuffer[MAX_REQUEST_SIZE];
   omni_message* queryMessage;
   quad_word dummyOrderId;
   int32_t cstatus;

   queryMessage = (omni_message*)sendBuffer;

   memcpy( &sendBuffer[0], &qryLength, sizeof(qryLength) );
   memcpy( &sendBuffer[sizeof(qryLength)], qryBuffer, qryLength );
```

3.8 omniapi_set_event_ex – Request Subscription of Broadcasts

The omniapi_set_event_ex() routine is used to set up a subscription to broadcasts.

3.8.1 Format

```
int32 cstatus = omniapi_set_event_ex (
omniapi_session_handle     hSession,     // in
uint32              evtype,       // in
int8 *              buffer)       // in/out
```

3.8.2 Description

The omniapi_set_event_ex() routine is used for setting up broadcast subscriptions. A user can either filter subscriptions in order to receive only certain broadcasts of interest, or request to receive everything for which they are authorised.

The omniapi_set_event_ex() call must precede the first call to omniapi_read_event_ext_ex() except when the omniapi_read_event_ext_ex() is called with two SHOW options.

3.8.3 Arguments

hSession

This argument must have been previously created with the omniapi_create_session call.

evtype

ASX Trade authorises the event types a user may subscribe to. The omniapi_read_event_ext_ex() routine must be called to get a list of such types. The list is returned as a comma separated, NULL terminated string, e.g. ”1, 2, 001\0”. The OI application can parse the list and call omniapi_set_event_ex() for each event type in it.

Event type one designates broadcasts from the trading system itself. In this case the subscription filtering can be used (see the buffer argument below). For all other event numbers the buffer argument must be NULL.

buffer

The very first call to omniapi_set_event_ex() determines whether the subscription filtering is to be used. If the subscription mechanism is not used (that is, this buffer argument is NULL), the gateway automatically requests subscription for all authorised information. In this case it is not possible to set up a new filtered subscription.

If the subscription filtering is used, this buffer argument contains the memory address of a structure containing the filter information. The subscription filtering is only supported for event type one. In all other cases, the buffer is ignored and should be set to NULL by the user.

Using the set_event_list_t structure, the buffer contains the following layout:

Length Field (4 bytes)	Item 1 (20 bytes)	…	Item *'n'* (20 bytes)

Field	Explanation
Length Field	The length of the whole buffer. This length field is included.
Item	An item that identifies a subscription filtering request.

The format of a subscription item has the following format (defined as subscr_item_t):

Information Object (12 bytes)	Subscription Handle (4 bytes)	Status (4 bytes)

Field	Explanation
Information Object	Is filled in by the caller. Identifies the subscription filter information.
Subscription Handle	Identifies the request. Filled in when the subscription is completed. The subscription handle is used when subscriptions are cleared, that is, when the routine omniapi_clear_event_ex() is called.
Status	Indicates whether the request was accepted. Filled in when the subscription is completed. The following values could be returned in the status field: OMNIAPI_SUCCESS – Successful completion; the subscription request was accepted. OMNIAPI_NOTAUTH – Not authorised to subscribe to the requested information. OMNIAPI_ALR_SET – This event has already been requested.

An information object structure is used to specify a single subscription request. It is a 12 byte structure (defined as infobj_t) with the following layout:

Information Source (2 bytes)	Information Type (2 bytes)	Broadcast Type (4 bytes)	Attribute (4 bytes)

The value zero in the above fields serves as a wildcard. However, not every field is allowed to have a wildcard. The table below provides details on each field:

Field	Explanation
Information Source	Identifies the information origin, i.e. the exchange code. For ASX Trade this value is always 15. This field cannot have a wildcard.

Field	Explanation
Information Type	Identifies how the attribute field (see below) is to be interpreted. **Possible values include:** 1 = General 2 = Derivatives 3 = Underlying 4 = Dedicated 5 = *Not used* 6 = *Reserved for internal use* 7 = Instrument Class 8 = Instrument Dedicated. **Note**: Wildcards are not allowed. Broadcasts listed in *ASX Trade Broadcasts* indicate what sort of information type is supported.
Broadcast Type	The actual broadcast identifier, for example BI41. See *ASX Trade Broadcasts* for all broadcasts.
Attribute	This field varies depending on the information type. **General Information Type (value = 1)** The purpose of the general broadcast is to send general text messages, market information, and market updates, e.g., "market has opened". For this information type, the attribute should be set to zero. **Derivative Information Type (value = 2)** The purpose of this attribute is to get information for one instrument class for a certain expiration date. For this information type the attribute is interpreted as three fields with the following format: Commodity (2 bytes) Expiration Date Year (1 byte) Expiration Date Month (1 byte). The value for the *Commodity* code is identical to the one used in the commodity code of the series binary code. Refer to series_t in *ASX Trade Broadcasts*. It can contain a wildcard. The Expiration Date, Year and Month fields are **obsolete** and exist only for historical reasons. They should be set to zero. This selection is narrower than for the underlying information type described below as it pertains only to derivatives. **Underlying Information Type (value = 3)** The purpose of the underlying information type is to get price information for an underlying, or its derivatives. This information type can also be used for status information on an underlying, e.g. to check if an underlying has been suspended. The attribute for this information type is a single field – Commodity (four bytes). Although it differs in byte size, the values to be used here are identical to those used in the commodity value in the series binary code. It can contain a wildcard. Refer to *ASX Trade Broadcasts* for more information. **Dedicated Information Type (value = 4)** The purpose of this attribute is to receive individually dedicated broadcasts, e.g. order information and contract information. Broadcasts with this information type are dedicated to a specific user or participant. For this information type, the attribute is a structure of two fields with the following format:

Field	Explanation
	Set to 1 (2 bytes) Member info (2 bytes). If all broadcasts directed to the participant (that is, all users with the same participant code) are to be received, the Member Info must be set to zero. If only broadcasts directed to the specific user are to be received, the Member Info must be set to one. **Instrument Class Information Type (value = 7)** The purpose of this attribute is to filter broadcasts based on instrument class. For the instrument class information type the attribute is interpreted as three fields with the following format: Underlying (2 bytes) Market (1 byte) Instrument Group (1 byte). The underlying can contain a wildcard. The market and instrument group can contain a wildcard only if both do so. A market and wildcard group cannot be specified, or vice versa. **Instrument Dedicated Information Type (value = 8)** The purpose of this information type is to filter broadcasts based on the attributes participant and instrument type. Instrument type is the combination of the 1 byte codes for the market and the instrument group respectively. Refer to series_t in *ASX Trade Broadcasts.* The attributes to this type are: Member info (2 bytes) Market (1 byte) Instrument Group (1 byte). Participants (member info) and instrument types are given using their internal numerical representations. It is possible to wildcard the subscription as follows: Any permutation of explicit participant and instrument type Explicit participant and wildcard instrument type Wildcard participant and wildcard instrument type Own participant and explicit instrument type Own participant and wildcard instrument type. A zero in the member part of the attribute means “all participants”. A zero in the instrument type part of the attribute means “all instrument types”. “Own Participant” is represented with an internal reserved value (65535). This reserved value is switched to the actual Participant ID by the gateway. Unless specifically permissioned (e.g. BSP user type), users are only able to subscribe to their “Own Participant” information.

3.8.4 Examples of Information Objects

Information Object	Description
15-1-BI41-1	This filter matches broadcasts from ASX (Exchange = 15) that are general in nature (Information Type = 1), relating to the ‘BI41’ broadcast. The attribute field is not in use and is currently always set to one.
15-2-BI63-0:0:0	This filter matches broadcasts from ASX (Exchange = 15) that relate to derivatives (Information Type = 2) and involve the ‘BI63’ broadcast for any commodity (attribute = 0, i.e. wildcard).

Information Object	Description
15-7-BO14-5080:2:6	This filter matches broadcasts from ASX (Exchange = 15) that relate to instrument class (Information Type = 7) and involve a 'BO14' broadcast for BHP American Call Options (underlying = 5080, market = 2, group = 6).

Note:
When querying several segments, this argument must be reset to the size of the buffer between every call, since it is modified to the length of the data returned. For any particular broadcast, the OI does **not** allow users to subscribe using wildcards if they have only been permitted for specific attributes of that broadcast. E.g., if a user is permitted for
15-7-BO15-0-101-200 and 15-7-BO15-0-102-200, etc., and then attempts to subscribe using an information object of 15-7-BO15-0-0-0, an error is returned. This is as a result of attempting to wildcard the market and group attributes when it has only been permitted as specific values. Where a wildcard has been permitted, it can be used or the user can be more specific if required.

3.8.5 Returns

cstatus

Status of the C function completion – less than zero if request failed; zero or greater if subscription was successful.

3.8.6 Return Values

A list of common error codes can be found in *2 Error Messages.*

3.8.7 Calling Examples

The following program demonstrates the use of the omniapi_set_event_ex() call setting up subscriptions for event type one (general broadcasts). The first example shows how to subscribe to all authorised information objects. The second example shows how to subscribe to specific broadcasts.

Example 1 – Subscribe to all Authorised Information Types

```
void APITEST_SubscribeAll()
 {
   uint32_t  buffSize = 10000;   /* strbuffer size */
   char      *strBuffer;
   int32_t   getEventStatus;
   char      *strPtr;
   uint32_t  eventType;
   int32_t   cstatus;

   strBuffer = (char*)malloc(buffSize);
   getEventStatus = APITEST_GetAllEventTypes(&strBuffer, &buffSize);
   if( getEventStatus != OMNIAPI_SUCCESS )
   {
       free(strBuffer);
       strBuffer = 0;
return;
   }

   strPtr = &strBuffer[0];
   do {
eventType = strtoul(strPtr, &strPtr, 10);
if (eventType != 0) {
         completionStatus = omniapi_set_event_ex( hSession, eventType, NULL );
```

```
  if (*strPtr != '\0') strPtr++; /* skip the comma delimiter */
      }
   } while (*strPtr != '\0' && eventType != 0 && completionStatus >= 0);

   free(strBuffer);
   strBuffer = 0;
   return;
}
```

Example 2 – Subscribe to Specific Broadcasts

```
int32_t APISAMPLE_SubscribeIndividualEvents()
{
   /*
    * Subscribe for Information source=15, Informationtype=4, Broadcast=BI7,
    * for all members
    */
   int32_t cstatus;
   uint16_t infsrc = 15;
   uint16_t inftyp = 4;
   int8_t module = 'B';
   int8_t server_type = 'I';
   uint16_t broadcast_number_n = 7;

   uint16_t deriv_commodity_n = 0;    /* Used for inftyp = 2 */
   uint8_t deriv_exp_year_c = 0;      /* Used for inftyp = 2 */
   uint8_t deriv_exp_month_c = 0;     /* Used for inftyp = 2 */

   uint32_t underl_commodity_u = 80;  /* Used for inftyp = 3 */

   uint32_t member_info_n = 0;        /* Used for inftyp = 4 */

   cstatus = APISAMPLE_SetEvent(
         infsrc,
         inftyp,
         module,
         server_type,
         broadcast_number_n,
         deriv_commodity_n,
         deriv_exp_year_c,
         deriv_exp_month_c,
         underl_commodity_u,
         member_info_n,
         dissemination_u
   );
   return cstatus;
}


int32_t APISAMPLE_SetEvent(
      uint16_t infsrc,
      uint16_t inftyp,
      int8_t module,
      int8_t server_type,
      uint16_t broadcast_number_n,
      uint16_t deriv_commodity_n,
      uint8_t deriv_exp_year_c,
      uint8_t deriv_exp_month_c,
      uint32_t underl_commodity_u,
      uint32_t member_info_n,
      uint32_t dissemination_u )
{
   infobj_t* bcinfo;

   /*
    * The type definition only includes a single item, even
    * though it is possible to have a list of items.
    */
   set_event_list_t list;

   list.buflen_i = sizeof(set_event_list_t);
   bcinfo = &list.subitm_x[0].infobj_x;

   bcinfo->infsrc_n = infsrc;
   bcinfo->inftyp_n = inftyp;
   bcinfo->brdcst_x.central_module_c = module;
   bcinfo->brdcst_x.server_type_c = server_type;
   bcinfo->brdcst_x.transaction_number_n = broadcast_number_n;

   switch( inftyp )
   {
   case 1:
      bcinfo->attrib_x.general_x.no_use_u = 1; /* Always 1 */
      break;
   case 2:
      bcinfo ->attrib_x.derivative_x.commodity_n = deriv_commodity_n;
      bcinfo ->attrib_x.derivative_x.exp_year_c  = deriv_exp_year_c;
      bcinfo ->attrib_x.derivative_x.exp_month_c = deriv_exp_month_c;
      break;
   case 3:
      bcinfo ->attrib_x.underlying_x.commodity_u = underl_commodity_u;
      break;
   case 4:
      bcinfo->attrib_x.dedicated_x.no_use_n = 1; /* Always 1 */
      bcinfo->attrib_x.dedicated_x.member_info_n = member_info_n;
      break;
   /* Have not yet done case 7 and case 8 */
   default:
      /* Unknown inftyp */
      return -1;
   }
   return omniapi_set_event_ex( hSession, 1, (char*)&list);
}

int32_t APITEST_GetAllEventTypes(char **strBuffer, uint32_t *buffSize)
{
   int32_t cstatus;

   glTransactionStatus = 0; /* Clear it, since we return a completion status */

   cStatus = omniapi_read_event_ext_ex(
hSession,
OMNI_EVTTYP_SHOW,
       *strBuffer,
buffSize,
0,
0);

   if( cstatus != OMNIAPI_SUCCESS )
   {
      printf("Error: Couldn't retrieve subscription event types\n");
   }
   return cstatus;
}
```

3.9 omniapi_clear_event_ex – Cancel Event Subscription

This routine is called in order to cancel a subscription of specific event types or broadcasts.

3.9.1 Format

```
int32 cstatus = omniapi_read_event_ext_ex(
       omniapi_session_handle     hSession,     // in
uint32                       evtype,       // in
int8 *                    rcvbuf,      // out
uint32 *                  rcvlen,      // in/out
uint32 *                  evtmsk,      // reserved
uint32                    optmsk)      // in
```

3.9.2 Description

This call cancels receipt of broadcasts for the event type specified.

Note:
If the buffer argument is used, all objects in the buffer are cancelled from the subscriptions setup.

All subscriptions that have been set up with omniapi_set_event_ex() can be cancelled with omniapi_clear_event_ex().

If only a subset of the subscriptions is to be cancelled, the user is required to fill the buffer with the actual objects.

If buffer is set to NULL (0), all subscriptions are cancelled.

Note:
Subscription filtering is only supported for event type 1 (general events).

3.9.3 Arguments

hSession

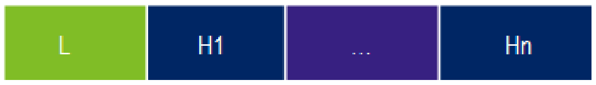
This argument must have been previously created with the omniapi_create_session call.

evtype

This argument specifies the event type of the events for which the broadcasts will be cancelled.

buffer

The first four bytes declare the size of the buffer (length field included). The rest of the buffer contains a list of subscription handles to be cleared. A subscription handle is retrieved when the application enables the subscription in the call to omniapi_set_event_ex().



Field	Explanation
L, 4 bytes	The length of the whole buffer, including this length field.
H, 4 bytes	A subscription handle.

If this buffer is specified as NULL, all active subscriptions will be cleared (see *omniapi_read_event_ext_ex – Read Events* for more information).

Note:
The buffer argument is only supported in conjunction with event type 1 (general types).

3.9.4 Returns

cstatus

Status of the C function completion – less than zero if request failed; zero or greater if cancellation of subscription(s) was successful.

3.9.5 Return Values

A list of common error codes can be found in *2 Error Messages.*

3.9.6 Calling Examples

The following program demonstrates the use of the omniapi_clear_event_ex call to cancel a subscription for events type 1 (general events). In the example below, the information object that was subscribed to earlier is cleared.

```
void APISAMPLE_ClearSubscription (uint32_t subscriptionHandle )
{
      int32_t   clearStatus;
      uint32_t  clrVec[2];


      clrVec[0] = sizeof(clrVec);
      clrVec[1] = subscriptionHandle;

      clearStatus = omniapi_clear_event_ex ( hSession,
                                      1,
                                      clrVec );
      return clearStatus;
}
```

3.10 omniapi_read_event_ext_ex – Read Events

This routine must be called regularly to receive broadcast messages unless the OI user is making use of the concurrent broadcast feature on which case omniapi_read_event_block is to be used.

3.10.1 Format

```
int32 cstatus = omniapi_read_event_ext_ex (
             omniapi_session_handle    hSession,     // in
uint32                      evtype,                  // in
int8 *                      rcvbuf,                  // out
uint32 *                    rcvlen,                  // in/out
uint32 *                    evtmsk,                  // reserved
uint32                      optmsk )                 // in
```

3.10.2 Description

This routine retrieves broadcasts that have been buffered at the gateway.

Note:
The user must read broadcasts for which they have set up subscriptions in a timely manner. One method of doing this is to use nested loops, where the outer loop polls with some reasonable time interval, while the inner loop iterates as long as there is data available.

3.10.3 Arguments

hSession

This argument must have been previously created with the omniapi_create_session call.

evtype

This parameter may specify either a single event type or all (OMNI_ EVTTYP_ALL) in order to check and fetch events.

If set to OMNI_EVTTYP_SHOW, a list of enabled event types is received. A typical string received in the rcvbuf argument would be “1,2,3\0”.

Note:
The string is NULL terminated.

If set to OMNI_EVTTYP_SHOW_SUBSCR a list of broadcasts that the user is authorised to receive is returned.

For more information, see *omniapi_read_event_ext_ex – Event Types* for full details.

rcvbuf

This argument references the receive buffer that is to be updated with event data.

rcvlen

The receive length argument specifies the length of the buffer provided with the rcvbuf argument. This argument has two interpretations:

Interpretation	Explanation
On input	The length of the caller’s provided buffer.
On completion	The length of the buffer received from the gateway.

evtmsk

This argument is reserved and currently ignored. Set to NULL.

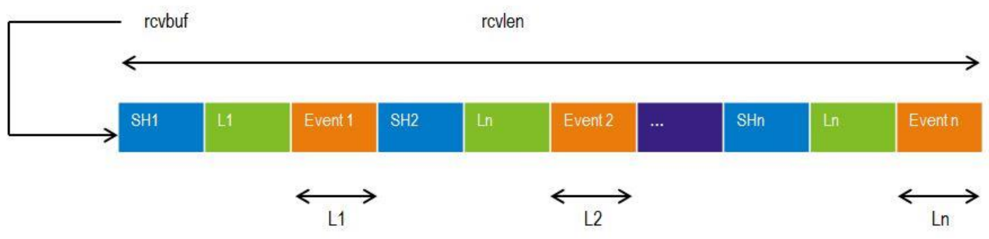
optmsk

The optmsk argument is a bit mask value stating optional behaviour of the omniapi_read_event_ext_ex() call. If the least significant bit is set (optmsk set to READEV_OPTMSK_MANY), the buffer depicted by the rcvbuf and rcvlen arguments is completed with broadcast messages chained together.

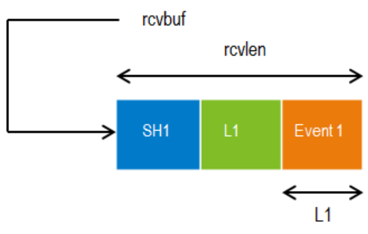
If the least significant bit is not set (optmsk set to READEV_OPTMSK_SINGLE), then the buffer contains just one broadcast message. The buffer can contain either one header (just the length) or two headers (the length and a subscription header) prefacing each event.

Field	Explanation
Lx, 2 bytes	The Lx length fields are two byte (little endian) unsigned integers. Users on big endian platforms have to byte-swap the contents of this field to determine the length of the event buffer. The event buffer is always padded with zeros to a longword (4 byte) boundary. The preceding length field includes the padding.
SHx, 4 bytes	Added if the application has filtered subscriptions when omniapi_set_event_ex() was called. No valid information is stored in this position.

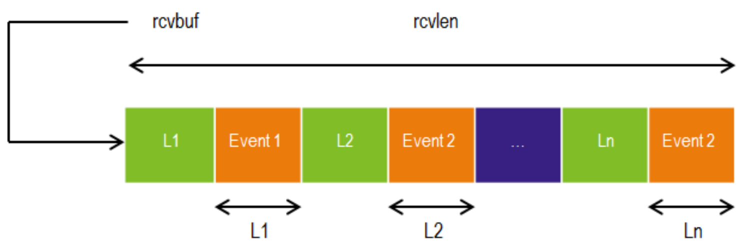
When the user has filtered subscriptions and used the READEV_OPTMSK_MANY option, the rcvbuf has the following format (mapping to the definition subscribed_event_t, which is 6 bytes in size).



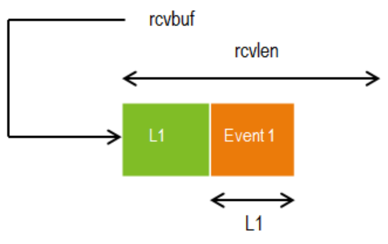
When the user has filtered subscriptions and used the READEV_OPTMASK_SINGLE option, then the rcvbuf has the following format.



If filtering subscriptions have **not** been used (omniapi_set_event_ex() was invoked with NULL as the parameter), the buffer has the following format.



If a single message is requested, it has the following format.



Each event is always padded with zeros to a longword(byte) boundary. The preceding length field includes the padding.

3.10.4 Setting up and Clearing Subscriptions

The header of an event is always either six or two bytes as described above. Only subscriptions concerning event type 1 (refer to *omniapi_set_event_ex – Request Subscription of Broadcasts*) affect the size of the event header.

3.10.5 Setting up Subscriptions

When setting up subscriptions, the following rules apply:

- Only the very first subscription setup affects the event header size.
- If the first subscription is a specific subscription, i.e. a buffer is provided as third argument to the omniapi_set_event_ex routine; the event header size is six bytes (subscribed_event_t structure).

- If the first subscription is a subscription for all events of a particular event type, i.e. a NULL buffer is provided for the third argument to the omniapi_set_event_ex routine, the event header size is two bytes (length of event).

Note: A participant who wishes to receive Price Information Heartbeat messages is required to subscribe to BI9. If the participant doesn't have any other subscriptions or event types set up previously, the event header size will be six bytes.

Examples

If a subscription for event type 990 (Dedicated Events) is set up at first, and then a subscription for a specific event, the event header is still two bytes in length.

If a subscription for a specific broadcast is set up at first, and then a subscription for event type 990 (Dedicated Events), the event header is still six bytes in length.

3.10.6 Clearing Subscriptions

When clearing subscriptions (refer to *omniapi_clear_event_ex – Cancel Event Subscription*), the following rules apply:

- If the event header size is six bytes, and the subscription for event type 1 is cleared, the event header is two bytes in size from that point on.
- If the event header size is six bytes, and the last specific filtered subscription is cleared, the event header size is two bytes from that point on.
- The event header size cannot change from two bytes to any other size when unsubscribing for events.

Examples

If a subscription for a specific event is set up, and then a subscription for event type 1 is set up, the event header is six bytes. When the event type 1 subscription is cleared, the event header becomes two bytes.

The last item of the buffer is empty and filled with zeros.

3.10.7 Hints

When fetching broadcasts, if the length is –1 (signed) or 65535 (unsigned) the header size is six bytes, and the first four bytes is a handle, which is always minus one (-1).

If a subscription for event type one is set up at first, the header is always two bytes. This means that if only specific subscriptions are required, there is a possibility to set up a subscription for event type one. Then the specific subscriptions are set up, and the subscription for event type one is cleared. This always provides a header of two bytes even if subscriptions are added or removed, until the last specific subscription is removed and new ones set up.

3.10.8 omniapi_read_event_ext_ex - Event Types

The evtype argument depicts the type and number of messages to be read or fetched. This includes the following.

Value	Type of Result
OMNI_EVTTYP_ALL	Any event message.
OMNI_EVTTYP_SHOW_SUBSCR	Lists all information objects for which the user is authorised to subscribe.
OMNI_EVTTYP_SHOW	List of event types enabled by the trading configuration.
OMNI_EVTTYP_NETWORK	Any network status message.

In order to identify the valid event types, users should issue an omniapi_read_event_ext_ex call (using the OMNI_EVTTYP_SHOW option) before any other event handling calls, in order to get a list of valid event types. The event type literals are defined in the omnifact.h file.

In a normal configuration, calling the omniapi_read_event_ext_ex with OMNI_EVTTYP_SHOW option, the following string would be returned:

"1001,1,990,\0"

When calling the omniapi_read_event_ext_ex() with the OMNI_EVTTYP_SHOW_SUBSCR option, a structure with a list (show_subsc_list_t) of authorised information objects (auth_infobj_t) will be returned. The list contains a 32 bit number specifying the number of authorised information objects returned, with the objects following the number:

Item Count (4 bytes)	Auth. Info. Obj. 1 (16 bytes)	…………..	Item *'n'* (16 bytes)

Each authorised information object (auth_infobj_t) has the following format:

Information Object (12 bytes)	Force Flag (4 bytes)

The force flag contains two possible values:
0 = not automatically subscribed by gateway
1 = automatically subscribed by gateway.

If the force flag field has a value of one, the OI automatically requests a subscription for that particular information object.

The Information object may contain wildcards for any of its broadcast types or attributes. A wildcard indicates that there is no authorisation control for that particular field (wildcard = 0).

Note:
For any particular broadcast, the OI does **not** allow users to subscribe using wildcards if they have only been permitted for specific attributes of that broadcast. E.g., if a user is permitted for 15-7-BO15-0-101-200 and 15-7-BO15-0-102-200, etc., and then attempts to subscribe using an information object of 15-7-BO15-0-0-0, an error is returned. An error is returned as the user is attempting to wildcard the market and group attributes when they have only been permitted as specific values.

Only where a wildcard has been permitted it can be used, or the user can be more specific if required.

3.10.9 Returns

cstatus

Status of the C function completion – less than zero if request failed; zero or greater if subscription was successful.

Note:
If there is no broadcast available, the return code is either the negative OMNIAPI_NOT_FOUND or the positive OMNIAPI_ALLEVTS. The OMNIAPI_NOT_FOUND is not to be regarded as an error.

3.10.10 Return Values

The following error values are specific codes that omniapi_login_ex can produce. A list of common error codes can be found in *2 Error Messages.*

Error Value	Code	Description
4	OMNIAPI_ALLEVTS	All events collected.
3	OMNIAPI_OVERFLOW	Success, but at least one event message was lost. Users should begin to recover their market picture if they receive this return value.

Error Value	Code	Description
-26	OMNIAPI_NOT_FOUND	Success, but there was no broadcasts available in the buffer.
-30	OMNIAPI_INVEVT	An invalid event type was specified.
-39	OMNIAPI_NOTAUTH	User is not allowed to read this event type.

3.10.11 Calling Example

The example how to use omniapi_set_event() also explains how to use omniapi_read_event_ext_ex with the OMNI_EVTTYP_SHOW options. The example program below polls all broadcasts, stores them in a buffer, and based on the broadcast, performs the proper action.

```
void APISAMPLE_PollBroadcasts()
{
    static char buffer[MAX_RESPONSE_SIZE];
    int32_t cstatus;
    uint32_t buffSize;

    cstatus = OMNIAPI_SUCCESS;

    while( cstatus == OMNIAPI_SUCCESS )
    {
       buffSize = sizeof(buffer);
       cstatus = omniapi_read_event_ext_ex(
hSession,
OMNI_EVTTYP_ALL,
buffer,
&buffSize,
0,
READEV_OPTMSK_MANY);

       if( cstatus >= OMNIAPI_SUCCESS )
       {
    if ( cStatus == OMNIAPI_OVERFLOW ) {
               printf("The gateway buffer has overflowed!!\n”);
printf(“Start recovery process!!\n");
return;
            }
    else if ( cStatus == OMNIAPI_ALLEVTS ) {
               printf("All events collected.\n");
    }
           APISAMPLE_ParseAndDistributeBroadcasts( buffer, buffSize );
       }
       else if ( OMNIAPI_NOT_FOUND == completionStatus )
       {
    printf("No broadcasts in buffer\n");
       }
       else
       {
           APISAMPLE_ReportError(completionStatus);
       }
   }
}
void APISAMPLE_ParseAndDistributeBroadcasts( int8_t* buffer, uint32_t buffsize )
{
   char *current; /* Pointer to the current pos in the in buffer */
   uint32_t totalLength;
   uint16_t eventLength = 1;
   uint32 skipSH = 0;

   totalLength = buffsize;
   current = buffer;

   if(gIndividualBdxSubscr)
   {
      /* The Subscription handle of 4 bytes before each broadcast
       * has to be skipped.
       */
      skipSH = sizeof(uint32_t);
   }

   while( (buffer + buffsize >= current) && (eventLength > 0) )
   {
      /* Skip Subscription handle if necessary */
      current = current + skipSH;
      /* Get the length of the event */
      eventLength = *(uint16_t*)current;
      /* Step to the start of the actual broadcast */
      current = current + sizeof(uint16_t);

      if (eventLength > 0)
      {
         APISAMPLE_HandleBroadcast( current, eventLength );
      }

      /* Step to the beginning of the next header */
      current = current + eventLength;
   }
}
void APISAMPLE_HandleBroadcast( int8_t* broadcast, uint16_t length )
{
   static broadcast_type_t typeBI9 = {'B','I',9};
   broadcast_type_t* bc;

   bc = (broadcast_type_t*)broadcast;

   if( memcmp( bc, &typeBI9, sizeof(broadcast_type_t) ) == 0 )
   {
      APITEST_HandleBroadcastBI9(broadcast, length );
      return;
   }
   printf("Broadcast <%c%c%i> received\n", bc->central_module_c,
                  bc->server_type_c, bc->transaction_number_n);
}

void APISAMPLE_HandleBroadcastBI9( int8_t* broadcast, uint16_t length )
{
   info_heartbeat_t* info_heartbeat;
   info_heartbeat = (info_heartbeat_t*)broadcast;
   printf("Broadcast <BI9>: Heartbeat every %d secs for %s.\n",
        info_heartbeat->heartbeat_interval_c,
        info_heartbeat->description_s);
}
```

3.11 omniapi_read_event_block – Blocking Read Event

This routine is used for retrieving broadcasts when the concurrent broadcast feature is enabled for the session.

```
int32 cstatus = omniapi_read_event_block(
omniapi_session_handle     hSession,     // in
uint32                     evtype,       // in
int8 *                     rcvbuf,       // out
   uint32 *                rcvlen,       // in/out
uint32                     timeout,      // in
uint32                     waittype)     // in
```

3.11.1 Description

This function is used on a concurrent connection to read events, blocking for the specified time period if no broadcasts are available for retrieval. For this function to succeed, the concurrent broadcast feature must be enabled for this session. For more information on concurrent broadcasts refer to *Appendix 1 – Concurrent Broadcast Feature*.

3.11.2 Arguments

hSession

This argument must have been previously created with the omniapi_create_session call.

evtype

This parameter may specify either a single event type or all (OMNI_ EVTTYP_ALL) in order to check and fetch events.

If set to OMNI_EVTTYP_SHOW, a list of enabled event types is received. A typical string received in the rcvbuf argument would be "1,2,3\0". **Note**: the string is NULL terminated.

If set to OMNI_EVTTYP_SHOW_SUBSCR a list of broadcasts that the user is authorised to receive is returned.

See *omniapi_read_event_ext_ex* for full details.

rcvbuf

This argument references the receive buffer that is to be updated with event data.

rcvlen

The receive length argument specifies the length of the buffer provided with the rcvbuf argument. This argument has two interpretations:

Interpretation	Explanation
On input	The length of the caller's provided buffer.
On completion	The length of the buffer received from the gateway.

timeout

This argument specifies the time period (in milliseconds) the function will block if no broadcasts are available for retrieval. The function will block until the condition specified by the argument *waittype* is satisfied. The minimum value that can be specified is zero and the maximum value cannot be greater than 10 seconds (10,000 milliseconds).

waittype

This argument specifies the condition for the wait. Currently only one value is supported:

OMNIAPI_BDXBUFFER_ATLEAST_ONE - Wait till at least one broadcast is available.

The buffer, specified by the rcvbuf argument, is completed with chained broadcast messages (events). For more details refer to *omniapi_read_event_ext_ex – Read Events* under the description of READEV_OPTMSK_MANY event mask.

3.11.3 Returns

cstatus

Status of the C function completion – less than zero if request failed; zero or greater if subscription was successful.

Note:
If there is no broadcast available, the return code is either the negative OMNIAPI_NOT_FOUND or the positive OMNIAPI_ALLEVTS. The OMNIAPI_NOT_FOUND is not to be regarded as an error.

3.11.4 Return Values

A list of common error codes can be found in *2 Error Messages.*

3.11.5 Calling Example

```
int PollBdx(omniapi_session_handle hSession)
{
int sts, cont=0;
unsigned int len;
short bdxlen, txn;
        char *buff;
        broadcast_type_t *bdx;
        len=64000;
        /*
        * Call blocking read_event API call for retrieving broadcasts.      * This will block for 2 
seconds if no broadcasts are available.
        */
                sts = omniapi_read_event_block(hSession, OMNI_EVTTYP_ALL, 
        bdx_buff,&len, 2000, OMNIAPI_BDXBUFFER_ATLEAST_ONE);
                if (sts < OMNIAPI_SUCCESS)
                {
                        printf("Poll Error : ");
                        printf("Status : %d (%s)\n", sts, GetErrorMsg(hSession, 
        sts));
                        return 0;
                }
                /*
                * Run through the buffer and print received broadcasts
                * Only the broadcast type and its size is printed
                */
                buff = bdx_buff + 4;
                bdxlen = *(uint16 *)buff;
                PUTSHORT(bdxlen, bdxlen);
                printf(" ");
                while (bdxlen > 0)
                {
                        bdx = (broadcast_type_t *) (buff + sizeof(uint16));
                        PUTSHORT(txn, bdx->transaction_number_n);
                        printf("[%c%c%-3i] ",bdx->central_module_c, bdx-
        >server_type_c, txn);
                        buff = buff + bdxlen + 6;
                        bdxlen = *(uint16 *)buff;
                        PUTSHORT(bdxlen, bdxlen);
                        ++count;
                        if (!((count) % 4)) /* Print 4 events per line */
                                printf("\n ");
                }
                printf("\n");
                printf(" Total Bdx:%d, Size:%d\n\n", count, len);
        return 1;
```

3.12 omniapi_get_info_ex – Get Environment Information

This routine returns system information according to the information type requested.

3.12.1 Format

```
int32 cstatus = omniapi_get_info_ex (
omniapi_session_handle     hSession,     // in
int32 *                    reason_pi,    // out
uint32                     inftyp_u,     // in
uint32 *                   inflen_pu,    // in/out
void *                     infbuf_p)     // out
```

3.12.2 Description

This is a generic routine for retrieving OI related information.

For example, this routine may be used for retrieving the numeric value for a facility type.

Note:
This call can **only** be performed when **logged in**.

3.12.3 Arguments

hSession

This argument must have been previously created with the omniapi_create_session() call.

reason_pi

Upon completion, this argument contains additional information about the cause of a call failure. The message text relating to the failure can be retrieved with the omniapi_get_message_ex() routine.

inftyp_u

This argument specifies the type of information requested. The following information types exist.

Information Type	Description
OMNI_INFTYP_USERCODE	Gets information about the user code assigned to the logged in user. The information buffer is assigned 12 characters.
OMNI_INFTYP_FACTYP_E0	Retrieves the facility number for the first external facility on the trading system. The information buffer is assigned a four byte integer.
OMNI_INFTYP_OMEXVERSION	Gets information about the current OI version. The version will be presented as a 128 byte character string.
OMNI_INFTYP_BANDWIDTH	Gets information about the bandwidth limitation and usage.
OMNI_INFTYP_OMEXEXCHNAME	Gets the exchange name as an acronym. Name is returned as a NULL-terminated string.
OMNI_INFTYP_OMEXEXCHCODE	Gets the exchange code. Returned data struct is omni_unsigned_num_t.
OMNI_INFTYP_COMPRESSION	Gets information on whether compression is used for the connection. Returned data struct is omni_signed_num_t. Returned values are OMNIAPI_OPVAL_DISABLE or OMNIAPI_OPVAL_ENABLE (in native endian format).
OMNI_INFTYP_CONCURRENT_BDX	Gets information about the concurrent broadcast support on the gateway. Returned data struct is omni_signed_num_t. Returned value is OMNIAPI_OPVAL_ENABLE (in native endian format) if concurrent broadcasts are enabled.
OMNI_INFTYP_ENCRYPTION	Gets information on whether encryption is used for the connection. Returned data struct is omni_signed_num_t. Returned values are OMNIAPI_OPVAL_DISABLE or OMNIAPI_OPVAL_ENABLE (in native endian format).
OMNI_INFTYP_PWD_EXPIRATION	Gets the number of days to password expiration. Returned data struct is omni_signed_num_t. If the returned value is negative then the password has expired, the indicated number of days ago. If the password has no expiration, OMNIAPI_NOT_APPLICABLE is returned as completion status. If the gateway does not have the information, OMNIAPI_NOT_FOUND is returned as completion status.
OMNI_INFTYP_TIME_UTC	Gets the Coordinate Universal Time (UTC) of the trading system. Returned data struct is omni_time_info_t.
OMNI_INFTYP_TIME_LOCAL	Gets the local time of the trading system. Returned data struct is omni_time_info_t.

inflen_pu

This argument has two functions:

- Before the call it describes the size of the provided information buffer.
- On call completion the argument holds the size of the returned information.

infbuf_ps

This argument references a call-provided buffer that is about to be updated with the requested OI information.

3.12.4 Returns

cstatus

Status of the C function completion – less than zero if request failed; zero or greater if the information was available.

3.12.5 Return Values

A list of common error codes can be found in *2 Error Messages.*

3.12.6 Calling Example

```
int32_t APISAMPLE_GetFacilityTypes()
{
   uint32_t factypeSize;
   int32_t cstatus;

   factypeSize = sizeof(glFacilityType_EP0);

   cstatus = omniapi_get_info_ex(
                           hSession,
                          &glTransactionStatus,
                           OMNI_iNFTYP_FACTYP_E0,
                          &factypeSize,
                          &glFacilityType_EP0);

   if( cstatus != OMNIAPI_SUCCESS )
  {
     printf("Error: Couldn't get Facility type. Completion status = %d\n",
            cstatus);

  }

   return cstatus;
}
```

3.13 omniapi_get_message_ex – Get an Exchange Message

This routine returns a message string associated with a message code for a logged in user.

3.13.1 Format

```
int32 cstatus = omniapi_get_message_ex(
omniapi_session_handle            hSession,     // in
int32                        msgcod,              // in
char                         msgstr,              // out
uint32                       msglen,       // in/out
int32                        simple(              // in
```

3.13.2 Description

All messages listed in the document *OMex System Error Messages Reference AS,* are returned, except positive OI routine return codes.

Note:
A “not connected” error message can cause problems by establishing a loop, if returned improperly.

3.13.3 Arguments

hSession

This argument must have been previously created with the omniapi_create_session() call.

msgcod

The message argument holds a message return from an earlier omn_api call.

msgstr

This argument references the string buffer about to receive the message string associated with the message code. The received string is not NULL terminated, meaning that the number of bytes that is written to the string buffer matches the msglen argument.

msglen

The message length argument specifies the length of the buffer provided with the msgstr argument. This argument has two meanings:

On input it has the length of the caller’s provided buffer

On output it has the length of the string received from the gateway.

simple

This variable is not used.

3.13.4 Returns

cstatus

Status of the C function completion – less than zero if request failed; zero or greater if the information was available.

3.13.5 Return Values

A list of common error codes can be found in *2 Error Messages.*

3.13.6 Calling Example

```
void APISAMPLE_ErrorTest(int32_t errorId)
{
char errorMsg[100];
int32_t errorMsgLength;
int32_t compStat;
if (errorId >= 0)
{
return;
}
scanf("%d", &errorId);
errorMsgLength=sizeof(errorMsg);
compStat = omniapi_get_message_ex( hSession, errorId, errorMsg,
&errorMsgLength, 0);
if( compStat == OMNIAPI_SUCCESS )
{
printf(" Completion status: %10d -> %s\n", errorId, errorMsg);
}
}
```

3.14 omniapi_set_option_ex – Set Options for OI Session

This routine is called in order to set up options for the OI session, such as:

- Compression
- Encryption
- Concurrent broadcast.

3.14.1 Format

```
int32 cstatus = omniapi_set_option_ex (
omniapi_session_handle     hSession,     // in
int32                      opttype,      // in
int32                      optval)       // in
```

3.14.2 Description

This routine sets options for already created sessions. The options can only be set before a login transaction has been sent.

3.14.3 Arguments

session

This argument must have been previously created with the omniapi_create_session call.

opttype

This argument is the option to set. Definitions for the options are defined in omniapi.h.

optvalue

This argument is the value of the option. Definitions for the values used with options are specified in omniapi.h.

3.14.4 Encryption

The ASX OMNet API does not support encryption. The OMNIAPI_OPT_ENCRYPT opttype parameters OMNIAPI_OPVAL_ANY and OMNIAPI_OPVAL_ENABLE will result in a connection rejection.

When setting encryption options, the OMNIAPI_OPT_ENCRYPT definition is used as the opttype parameter. Valid values for encryption options are:

Values	Description
OMNIAPI_OPVAL_ANY	This is the default value. The setting for the gateway is used.
OMNIAPI_OPVAL_ENABLE	The user demands encryption, and if encryption is disabled on the gateway or not implemented in the OI on the current platform, the login is rejected.
OMNIAPI_OPVAL_DISABLE	The user demands that encryption must not be used. If the gateway has enforced encryption, the login is rejected.

The following table shows how the connection is set up with different settings. Please note the encryption setting is to disallow.

API Client Encryption Setting	Gateway Encryption Setting		
	Any	Required	Disallow
OMNIAPI_OPVAL_ANY	No encryption	Encryption	Connection is rejected
OMNIAPI_OPVAL_ENABLE	Encryption	Encryption	Connection is rejected
OMNIAPI_OPVAL_DISABLE	No encryption	Connection is rejected	No encryption

3.14.5 Compression

When setting compression options, the OMNIAPI_OPT_COMPRESS definition is used as the opttype parameter. Valid values for compression options are:

Values	Description
OMNIAPI_OPVAL_ANY	This is the default value. The setting for the gateway is used.
OMNIAPI_OPVAL_ENABLE	The user demands compression. If compression is disabled on the gateway, the login is rejected.
OMNIAPI_OPVAL_DISABLE	The user demands that compression must not be used. If the gateway has enforced compression, the login is rejected.

The following table shows how the connection is set up with different settings:

OI Client Compression Setting	Gateway Compression Setting		
	Any	Required	Disallow
OMNIAPI_OPVAL_ANY	No compression	Compression	No Compression
OMNIAPI_OPVAL_ENABLE	Compression	Compression	Connection is Rejected
OMNIAPI_OPVAL_DISABLE	No Compression	Connection is Rejected	No Compression

3.14.6 Concurrent Broadcasts

When enabling or disabling the concurrent broadcast feature, the OMNIAPI_OPT_CONCURRENT_BDX definition is used as the opttype parameter. Valid values for concurrent options are:

Values	Description
OMNIAPI_OPVAL_ENABLE	Enables the concurrent broadcast feature for the current session. An error will be returned if the gateway does not support concurrent broadcasts or if the feature is already enabled for this session.
OMNIAPI_OPVAL_DISABLE	Disables the concurrent broadcast feature for the current session. An error will be returned if the feature is already disabled for this session.

3.14.7 Returns

cstatus

Status of the C function completion – less than zero if request failed; zero or greater if request was successful.

3.14.8 Return Values

A list of common error codes can be found in *2 Error Messages.*

3.15 omniapi_set_option_default – Set Default Values for OI Session

This routine is used to set default values for sessions created after this routine has been called.

3.15.1 Format

```
int32 cstatus = omniapi_set_option_default(
int32  opttype,      // in
int32  optval )      // in
```

3.15.2 Description

This routine is used to set default values on options for sessions created after the routine has been called. This routine does not affect any sessions that have already been created.

3.15.3 Arguments

opttype

This argument is the option to set. Definitions for the options are defined in omniapi.h.

optvalue

This argument is the value of the option. Definitions for the values used with options are specified in omniapi.h.

For argument explanation see *omniapi_set_option_ex*.

3.16 omniapi_cvt_int – Convert an Integer

This routine is called when the application needs to convert two or four byte binary integer data from little endian format to big endian format or vice versa.

3.16.1 Format

```
void omniapi_cvt_int (
void*  number,       // in/out
int16 numsiz )       // in
```

3.16.2 Description

This routine is called when the application needs to convert two or four byte binary integer data from little endian format to big endian format or vice versa. It is not available on computers using little endian byte order.

3.16.3 Arguments

number

The number argument references the integer about to be converted.

numsiz

The numsiz argument gives the byte size of the integer about to be converted. This argument must be either two or four bytes.

3.17 omniapi_cvt_string – Convert a String to/from the Central Format

This routine is called when the application wants to exchange application strings with a trade application.

3.17.1 Format

```
cstatus = omniapi_cvt_string (
int8          toctrl,       // in
uint8*        ebtstr )      // in/out
```

3.17.2 Description

This routine is used to convert an application 8 bit string from a non-ISO Latin-1 system to the character string encoding of the trade application.

3.17.3 Arguments

toctrl

This boolean argument ("to trade") states whether conversion is to or from the trade format, that is, the ISO Latin-1 character set.

- A boolean TRUE, normally a value of one, represents that the conversion is TO the trade format.
- A boolean FALSE, normally a value of zero, represents FROM the trade format.

ebtstr

This argument references a string whose contents are converted byte-by-byte from/to the local character representation to/from the central character representation. The string is NULL terminated.

3.17.4 Example

Passing the MS-DOS represented string «ñá to this routine using the TO direction would produce a converted string «ñá interpreted in a central application and a converted string 1/2±ß as seen from the MS-DOS application.

3.18 omniapi_convert_timestruct – Convert Timestructs

This routine is called when the application needs to convert an omni_tm_t data struct to a struct tm data struct.

3.18.1 Format

```
struct tm* omniapi_convert_timestruct(
omni_tm_t *   omni_tm_p,    // in
struct tm *   tm_p)         // out
```

3.18.2 Description

This routine is used to convert from the OI defined omni_tm_t data struct (little endian) to the struct tm (native endian), which is defined within the C language.

3.18.3 Arguments

omni_tm_p

The omni_tm_t struct is in little endian.

tm_p

The struct tm is in native endian, i.e., the endian of the OI client application platform.

3.18.4 Return Values

The tm_p pointer is returned.

If any of the arguments are NULL, the function returns NULL.

4 Appendix

4.1 Appendix 1 – Concurrent Broadcast Feature

- This functionality allows an OI application to use a concurrent thread to read broadcasts. ASX Trade Refresh OMNet API libraries no longer establish two separate TCP connections to the gateway for utilising the concurrent broadcast feature. Instead, the library uses a single connection with a different thread to read the broadcast. This allows a multi-threaded application to use a blocking call for reading broadcasts (one which will not return until there is at least one broadcast) whilst still being able to send in transactions and queries all under the same session. In effect this allows the application to be continually polling for broadcasts without needing to interrupt other tasks.
- This functionality is optional and does not affect the legacy method of retrieving broadcasts from the gateway. However ASX recommends that all latency dependent applications make use of this functionality.

4.2 Appendix 2 – Environment Variable OAPI_TIMEOUT

Omnet API users can timeout Omnet API calls by using "OAPI_TIMEOUT". This setting is in seconds, and it should be exported as an operating system environment variable. The recommendation is that the variable should be set to a time greater than 5 seconds, to cater for network re-convergence.