NASDAQ OMX®

Genium INET℠

OMnet Application Programmer's Interface Manual

HKEX

Version: 2.0.0801

Document ID:	OMnet_API_12
Documentation Release:	GENIUM_Product_a810
Release Date:	2013-04-25
Publication Date:	2013-04-25

Table of Contents

Appendices

List of Figures

1 Summary of Changes

The following table shows the differences between this version and version a682 (2.0.0801).

No	Changes	Comment
1	In Chapter “Forced Login,” a note was added, please see page 31.	

2 Getting Started

2.1 About the Application

The OMnet application has four major purposes:

- To provide the central Genium INET system with validated transactions from trusted sources.
- To distribute market information efficiently.
- To provide user authentication, that is the user is identified by the system via a traditional login that uses information in the CDB.
- To provide user authorization, that is regulation of the user's rights in the system as set in the CDB.

2.2 About the Manual

2.2.1 Audience

The manual is intended for developers who have knowledge about ANSI C.

2.2.2 Purpose

The purpose of this manual is to describe the functionality of the OMnet Application Programmer’s Interface (API) and the interaction between an application and the interface (OMNIAPI).

3 General Features

3.1 OMnet Functionality

1. **Login functionality**

   OMnet handles each user's access to and rights within the Genium INET system (as setup in the CDB). Each user can only access the system from one terminal at a time – multiple access is not possible for security reasons.

2. **Limitations**

   OMnet serves as an interface between Genium INET client applications and the Genium INET backend.

3. **Distribution**

   OMnet distributes the broadcasts.

3.2 The OMnet API Connection

External client applications use the OMnet API library to connect to the OMnet Gateway process. The OMnet API is synchronous. It runs on various operating systems, but resides in the local machine.

This picture shows how client applications (for example, trading applications) connect to an OMnet Gateway process in order to access the Genium INET system.

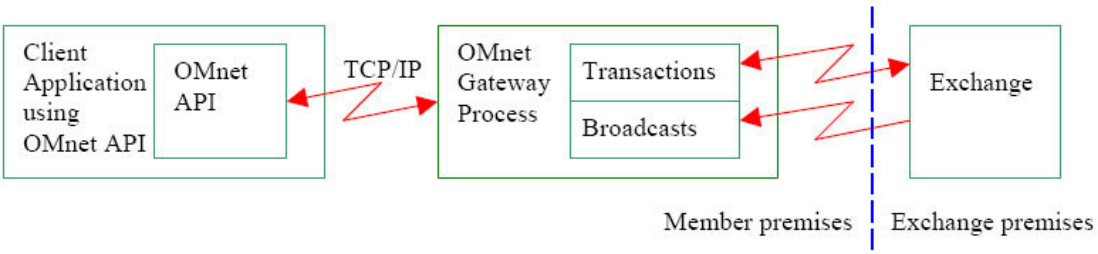

Figure 1: Connections to the OMnet Gateway Process

3.2.1 Transactions and Broadcasts Definition

An accepted transaction is either a successful message or a question from a client to a server with a corresponding reply. A broadcast is information from a server to one or more clients.

Both transactions and broadcasts are named according to the following convention:

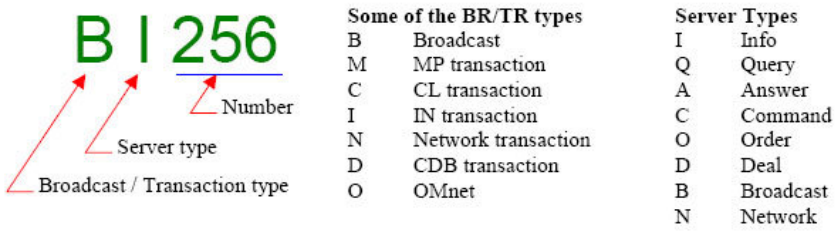

Figure 2: Transactions and Broadcasts Definition

Each transaction and broadcast name corresponds to a record definition. The names and record definitions can be found in the *OMnet Message Reference*.

3.3 Facilities and Facility Types

Facilities grant different applications different access to the Genium INET backend network and separate different types of information.

The system is able to interpret the transaction only if it is sent on the correct facility.

Use **`omniapi_get_info_ex`** to retrieve the numeric value for EP0 (external facility) . When you have the value for EP0 you can also calculate the other values: EP1 = EP0 + 1, EP2 = EP0 + 2, and so on. The same applies for IP0 (internal facility).

The facility number is alpha numeric, the range of its value is 0-9 and continue with A-Z, where A equals 10 and B equals 11, etc. For example facility EPA is calculated as EP0+10.

Example 1: Application Facility Types

`OMNI_FACTYP_OM__XSEEP0`	This is OM's (**OM__**) exchange code (**X**) for Sweden (**SE**) and the external (**E**) production (**P**) facility number zero (**0**). In the facility header file (**OMNIFACT.H**) this is defined as OMNI_FACTYP_OM__XSEEP0.
`OMNI_FACTYP_OM__XSEET0`	The first facility type for external test access to OM Stockholm.

3.4 OMnet Broadcasts

A broadcast message is called an *event*.

A user application may:

- Subscribe to event types or broadcasts
- Cancel subscriptions to event types or broadcasts

Events occur asynchronously and are buffered by the OMnet Gateway process. Since the OMnet API is synchronous, user applications must poll the API regularly to be able to read the events. Please, see also Concurrent Broadcasts and Transactions on page 91.

3.4.1 Broadcast Information Objects

An application has the ability to:

- specify information of particular interest, *or*
- request subscription for everything the user has authorization for (default setting)

For more information about dynamic subscription, see **`omniapi_set_event_ex()`** in *omniapi_set_event_ex-Request Subscription of Broadcasts on page 56*.

An *information object* structure is used to specify a subscription request. This is a twelve-byte structure with the following layout:

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

Wildcards can be used in a subscription request but not in all structures, see details below if wildcards can be used. A field value of zero in the information object is interpreted as a wildcard.

Byte	Explanation
Information source	Identifies the information origin, that is, the numeric exchange code. In this structure, the subscription information retrieved from the backend may contain zero. However, it should not be interpreted as a wildcard but as the numeric exchange code. Also, the user is not allowed to explicitely set information source to zero if the information source retrieved from the backend is not zero. This means that a wildcard search cannot be performed for information source.
Information type	The information type in the information object also defines how the attribute field in the information object should be interpreted. Currently the following types are defined: 1. General 2. Derivatives 3. Underlying 4. Dedicated 5. Dissemination group 6. - 7. Instrument Class 8. Instrument Dedicated information type. **Note:** Number 6 is for internal use only. Not all information types are valid for all exchanges. Wildcards are not allowed.
Broadcast type	Identifies the information structure associated with the information object. A broadcast could, for example, be BI9, “Price Information Heartbeat.” The broadcast structure should be interpreted as two characters and one unsigned short. Wildcards are allowed. You can use either wildcard (all) or a specific broadcast, e.g. BI9, BD6, BE4 For more information about defined broadcast structures, please see *OMnet Message Reference*.

Byte	Explanation
Attribute	Defines the characteristics for the information object. The interpretation is depending on the information type. No validation test on the attribute is done when a subscription is set up. This means that it is possible to set a subscription on something that does not exist and it is possible to set the same subscription twice.
	General information type (1)
	The purpose of the general broadcast is to send general text messages, market information, and market updates, for example, "market has opened". For the general information type (value = 1), the attribute should be set to zero.
	Derivative information type (2)
	The purpose of this attribute is to get information for one instrument class for a certain expiration date. For the derivative information type (value = 2) 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 Commodity code is identical to the one used in the "Series" defined for instruments. For more information on the series concept, see "Series" in the *OMnet Message Reference*. The Expiration Date Year and Month fields are obsolete and exist only for historical reasons. This selection is narrower than for the commodity information type described below.
	Underlying information type (3)
	The purpose of the underlying information type is to get price information for an underlying, or its derivatives. Underlying is also used for status information, for example, whether trading going on, whether an underlying is suspended. For the underlying information type the attribute should be interpreted as the underlying. The underlying code is identical the one used in the "series" defined for instruments.
	Dedicated information type (4)
	The purpose of this attribute is to receive individually dedicated broadcasts, for example, order information and contract information. Broadcasts with this information type are dedicated to a specific member or to users at a specific member. For the dedicated information type (value = 4) the attribute is interpreted as two fields with the following format: • Not used (2 bytes) • Member info (2 bytes) The Not Used field should be set to one (1). If all broadcasts directed to the member (that is, all users with the same member code) are to be received, the Member Info must be set to zero (0).

Byte	Explanation
	If only broadcasts directed to the specific user are to be received, the Member Info must be set to one (1).
	Dissemination group information type (5)
	The purpose of this attribute is to enable efficient distribution of related information. A generalized group of financial instruments [Derivatives] can be defined in CDB. For the dissemination group information type, the attribute should be interpreted as the dissemination group id. **Note:** This information type is no longer used.
	Instrument Class information type (7)
	The purpose of this attribute is to improve efficiency by bundling multiple items in the same broadcast. The use of Virtual underlying is introduced. Usage of Virtual Underlying Code makes it possible to pack items belonging to different regular underlying in the same broadcast, thus reducing number of broadcasts. 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)
	Instrument Dedicated information type (8)
	The purpose of this information type is to filter broadcasts based on the attributes Member and Instrument Type. Instrument Type is the combination of the 1-byte codes for the Market and the Instrument Group respectively. The attributes to this type are: • Member info (2 bytes) • Market (1 byte) • Instrument Group (1 byte) Members and Instrument Types are given using their internal numerical representations It is possible to wildcard the subscription as follows: • Any permutation of explicit Member and Instrument Type. • Explicit Member and wildcard Instrument Type. • Wildcard Member and wildcard Instrument Type. • Own Member and explicit Instrument Type. • Own Member and wildcard Instrument Type. A zero in the Member part of the attribute means “all Members”. A zero in the Instrument Type part of the attribute means “all Instrument Types”. “Own Member” is represented with an internal Reserved Value (65535). This Reserved Value is switched to the actual Member ID by the OMnet Gateway. **Note:** In the case where one would like to configure separately the individual parts of an Instrument Type (Instrument Group; Market) the Product Design application is needed.

Example 2: Information Objects

Information Object	Description
`43-1-BD2-1`	Will match all information from *information source* "43" (= UCA Sweden), *information type* "1" (= general) and *broadcast type* "BD2" (= edited price information). The last "1" is not in use and currently always set to "1".
`43-2-BO2-0`	Will match all information from *information source* "43" (= UCA Sweden), *information type* "2" (= derivative) and *broadcast type* "BO2" (= order book changes) and **any** *commodity*.
`43-3-BO14-45`	Will match all information from *information source* "43" (= UCA Sweden), *information type* "3" (= underlying) and *broadcast type* "BO14" (= OB levels) for *commodity* 45.
`43-4-BD6-1-0`	Will match all information from *information source* "43" (= UCA Sweden), *information type* "4" (= dedicated) and *broadcast type* "BD6" (= dedicated trade information transaction) for this **member**. **Note:** The first "1" trailing the BD6 is not in use and should be set to "1".

3.4.2 Broadcast Ordering

The OMNet API delivers broadcasts in a unique order (for a given sequence of transactions) given the following conditions:

1. They are of the same broadcast type (such as BO5).

2. They belong to the same partition (based on, for example, instrument series).

This implies that the API makes no guarantees of the broadcast delivery order for messages of different types (such as BO5 vs BD6). For example, a BO5 may arrive ahead of the BD6 and sometimes after.

Furthermore, the same is true for messages of the same type, but originating from different partitions (for example, BO5 messages from two different underlyings may be merged in different orders by two different gateways).

3.5 API Compression and Encryption

Data sent between the API and the gateway process may be compressed and /or encrypted in a way that is negotiated at API-client session startup. The negotiated setup is decided based upon requirements by the API client, configuration and installed software components.

API compression and encryption by means of configuration is set in the PD application. It is possible to set if configuration and / or encryption is required, optional or not used. The API- client may suggest a compressed and / or encrypted connection between the API and gateway, but the gateway has the final decision on setup. See documentation regarding the **`omniapi_set_option`** API- call further ahead in this document.

Type of encryption used is SSL.

4 Programming Considerations

4.1 Programming Language

The programming interface is designed for and supports ANSI C applications.

4.2 General Considerations for an API Program

Use standard C runtime routines and avoid calls specific to operating systems.

4.2.1 Login Requests

Login is a complex and sensitive operation. Fast repeated login attempts are considered by the system to be Denial of Service attacks, and for that reason the OMnet Gateway process is trying to detect and deal with abuse.

The exchange may configure that after a certain number of failed login attempts the user account gets locked (login request returns OMNIAPI_INVALID_LOGIN_LOCKED). The user will not be able to login without calling the exchange's Help Desk and get the user account unlocked.

The exchange may decide to suspend a user account (login request returns OMNIAPI_INVALID_LOGIN_SUSPENDED). The suspension is used by the exchange when a user or member does not follow or comply with the regulations of the exchange. Contact the exchange's Help Desk for the reason for suspension.

4.2.2 TPS Limitation

There is a TPS limit per user type, which is set in CDB as number of transactions per second. This limit is enforced by the OMnet Gateway in such a way that transactions are only sent with the maximum rate defined for every second, that is, they are smoothed out. The API is strictly synchronous; thus, an application attempting to send at a substantially higher rate than the TPS limit will appear 'hung' in the API call most of the time.

4.2.3 Bandwidth Limitation

If the exchange has configured the Network Gateway, user or user type to have restrictions on bandwidth usage the user has to control his bandwidth usage. To get information about bandwidth limit and bandwidth usage use function **`OMNIAPI_GET_INFO_EX`**, see Section 6.3.12 on page 74. The structure that **`omniapi_get_info_ex`** returns is described in omniapi.h (omni_bw_info_t).

4.2.4 API Client Activity

The Genium INET system has extensive heartbeat functionality built in to protect the trader. The API client is expected to issue API calls at the rate configured in CDB as heartbeat interval for the user-type (cdb_uhb). The Gateway will count each API call as an interaction and set an 'alive' bit. A special transaction is sent to the central system (cdb_uhb) periodically as long as the API client behaves well. If the 'alive' transaction

does not appear on proper time, the central system will consider the user 'lost' and take action, depending on policy for the exchange in question, for example, pulling all quotes from the order book.

Thus, it is necessary that an API client application interacts with the OMnet Gateway at least with the frequency specified for the user type (typically by issuing **`omniapi_read event_ext_ex()`** (read event) calls).

4.2.5 Subscription Obligations

An API client may subscribe for broadcasts from the central system. It is of the outmost importance that the client establishes reasonable subscriptions. The bandwidth requirements must be within bounds for both the connection between the API client and the Gateway (usually at least 10 Mb/s LAN) and the connection between the Gateway and the Central system. The Gateway (and the network) is usually shared between several API users.

It is also important that the API client calls o**`mniapi_read event_ext_ex()`** (read event) with sufficient frequency to avoid broadcasts being buffered in the Gateway. Clients not reading their broadcasts sufficiently will be disconnected.

4.2.6 Anticipating Changes in the API

The Genium INET system provides as much 'backward' compatibility as possible, for example:

Applications written for Version X of the system should ideally work fine after a recompile for version X+1. At the same time the various exchanges using Genium INET has very high expectations of new functionality; thus, the Genium INET system has to provide extensions to the existing API. The API has a programming model, where a lot of data is passed in structures. It is quite possible that structures may be extended with new members in the future; thus, it is wise to always explicitly set predictable values in all structures before use, for example:

```
memset(a_struct,0,sizeof(a_struct));
```

This ensures that any new member of the structure will have a zero value, which is not necessarily sufficient in all cases, but a zero is preferable to just random values.

4.2.7 String Formats

For structures used in communication with the backend, that is, for transactions, queries, and broadcasts, the string fields are of fixed length. The strings should be padded with spaces at the end and not null-terminated.

> **Note:** String fields in structures used for the OMnet API programming interface that are defined in this manual expect C-style null-terminated strings. This is the case, for example for the omni_login_t structure.

4.3 Endian Issues

Use the

- PUTORDERID
- PUTLONG

- PUTQUAD
- PUTSHORT

macros (provided with the `omniapi.h` header file) when storing and fetching numbers.

4.4 Data Types

The following points apply when writing portable OMnet applications in C:

- Use the supplied definitions for integers (defined in om_inttypes.h header file).
- Pay attention to alignment issues, header files for the Genium INET system have #pragma directives to ensure consistent packing of structures. One may have to use compiler flags, such as “-misalign” on Solaris, when building API clients or explicitly do memcopy of data to properly aligned locations for ‘native’ access.

4.5 The Session

The session between the OMnet API application and the OMnet Gateway process is established by the *login* request and terminated by the *logout* request.

The following steps are executed during a login:

1. A connection to the OMnet Gateway process is established.
2. A connection from the OMnet Gateway to the Central System system is established.
3. OMnet verifies the application user.
4. OMnet establishes an application session.

The OMnet Gateway may reject a login request for the following reasons:

- The given username and/or password is invalid.
- The Gateway is not configured for an access from this node or by this user.
- The user is not configured to use the used Application.
- Too many applications are attached to the OMnet Gateway. The limit depends on several factors, such as network traffic, hardware configuration and OS.
- The previous session has been abnormally interrupted and the Gateway is still processing a transaction.

If a login request returns OMNIAPI_PWD_CHANGE_REQ, the session will have restricted access to the Genium INET system. The following requests may be made:

- **`omniapi_get_info(..)`**
- **`omniapi_get_message(..)`**
- **`omniapi_read_event_ext(..)`**
- **`set new password request`**
- **`logout request`**

If other requests are made, an error code will be returned.

Even if the API Client Application has only restricted access to the Genium INET system, the Application still has to oblige to the requirement of showing activity as described in API Client Activity on page 19.

Once a new password is successfully set, the Application gets full access to the Genium INET system.

A session may be terminated by the Gateway for several reasons, such as:

- The application has failed to show activity by calling any routine in the OMnet API within a specified time period.
- The OMnet Gateway has detected an aborted transaction with a fatal **completion status code**.
- The OMnet Gateway has detected that the socket between the API Client Application and the OMnet Gateway is closed.

Where stated, the API applications must always submit a valid session handle to the omniapi functions. It is not possible to reuse a session, that is, when the session has been logged out the session has to be closed. The session has to be closed after all logouts, both normal logout and logouts initiated by the Genium INET system.

4.5.1 Forced Logout

A central subsystem or an operator may force a user logout, requiring another login by the user. The OMnet API application will detect this by finding the return code OMNIAPI_NOT_LOGGED_IN from all OMnet API routines.

Example 3:

- A session will be logged out if someone else logs in from another source using the same username.
- Any logged-in trading application will be logged out at once per day. The default time is during the system night batch, when all users are automatically logged out.
- An API session can be logged out because of exceeded bandwidth limitation, see Bandwidth Limitation on page 19.

4.6 Network Event Types

4.6.1 Network Events

These events contain status information about the network. They are transmitted to everybody. See **omni.h** for details (omni.h on page 36). The network events have a similar format to other Genium INET exchange events. Two predefined type definitions exist.

```
typedef struct
{
  int32         status_u;
  uint16        msglen_n;
} omni_netwrksts;
typedef struct
{
  int8           central_module_c;
  int8           server_type_c;
  uint16         broadcast_number_n;
  omni_netwrksts network_status_x;
```

```
} omni_broadcast;
```

Variable	Explanation
`status_u`	The status holds message code and a copy of the message string received. **Note:** The information in the value is a signed integer (int32).
`msglen_n`	The message length depicts the length of the message string starting immediately after the message length word.
	The message text is following the **omni_netwrksts,** but is not a part of the structure. The length of the network status message is the size of **omni_broadcast + msglen_n**.

4.6.2 BN1 – Network Status Events

The OMnet network status events have the broadcast type **BN1.**

The possible status values returned in the **BN1** broadcast type are defined in **omni.h** and described below:

Status Value	Explanation
OMDU_DATALOST	One or several broadcast messages may have been lost. The severity of this problem depends on the application. The reason for data loss is that OMdu is designed to provide reliable delivery of broadcasts in strict sequence order. If a broadcast is missing, OMdu will initiate retransmission, and buffer 'pending' broadcasts until the missing one is received. If there are not enough resources available, for example, the pending_receive queue in the OMnet Gateway is not large enough for this to work successfully, then this event will occur. The string associated with the message provides the name of the sender service at which losses have been detected.
OMDU_LINKLOST	The connection to the broadcast system has been lost. This is a serious problem. The application program should call the OMnet API to cancel the current subscriptions and then re-subscribe them.
OMDU_SENDERLOST	The broadcast system has lost the connection to a sender. The OMnet API will automatically receive new broadcasts from that sender when it becomes available again. This event can be ignored by most applications, since the Genium INET systems typically are running at least two instances of each server process, thus if a primary server process dies (and is lost for OMdu) when the standby instance of the process will become active and start sending. The event could be interpreted as 'fail-over in progress'

The message texts associated with the OMdu message codes get available by calling the **`omniapi_get_message_ex()`** routine described below.

4.6.3 BN2 – New Subscription Available

The OMnet *new subscription* event has broadcast type **BN2**. It tells the API application that a new subscription type is available.

The event received has the following format:

`Broadcast Type (4 bytes) = BN2`	Authorized Information Object *(16 bytes)*

The name of the structure is **new_subscr_t** and it can be found in **omniapi.h**.

See **`omniapi_set_event_ex()`** for how to subscribe to the new broadcast, (Section 6.3.9 on page 56).

> **Note:** This will only work if the program uses the subscription mechanism. The program must call **`omniapi_read_event_ext_ex()`** with **OMNI_EVTTYP_SHOW_SUBSCR** and then set the subscription with **`omniapi_set_event_ex()`**.

4.6.4 BN3 – Bandwidth Limitation Network Event

The Gateway measures bandwidth usage between the Gateway and API clients, and the Gateway takes action when a limit is reached. The Bandwidth Limitation Network Event has broadcast type **BN3**. The BN3 network event is sent to the API application when the application is using a certain percentage of the allowed bandwidth. The limit, the action and the percentage specifying when to send a BN3 are configured by the exchange. The same limit applies to inbound and outbound traffic.

A BN3 is also sent when network usage is back to normal.

The actions are:

- Log out user
- Do nothing

The possible status values returned in the **BN3** broadcast type are defined in **omni.h** and described below:

Status Value	Explanation
OMNI_BWUSAGE_HIGH	Bandwidth usage is above configured levels. This message is intended as warning that bandwidth usage is approaching the limit.
OMNI_BWLIMIT_EXCEDED	Bandwidth usage is above the limit. This should be considered a serious problem since the exchange might be configured to force a logout if this occurs too frequently.
OMNI_BWUSAGE_NORMAL	Bandwidth usage is back below configured levels

The message texts associated with message codes get available by calling the **`omniapi_get_message_ex()`** routine described below.

4.6.5 BN4 – Network Events

The **BN4** event is sent from the Network Gateway to indicate that transactions or broadcasts are not working.

The possible status values returned in the **BN4** broadcast type are defined in **omni.h** and described below:

Status Value	Explanation
OMNI_NE_LOGOUT_IMMINENT	The backend has issued a forced logout due to network problems

The message texts associated with message codes get available by calling the **`omniapi_get_message_ex()`** routine described below.

4.6.6 BN5 – Network Events Related to Cached Data

The Network Events Related to Cached Data has the broadcast type **BN5**.

Some customers may use a local cache situated on a Network Gateway in order to provide better performance when bandwidth between the member site and the exchange is limited. This cache component stores order and trade history data and serves queries sent from an API client through the Network Gateway.

There may be cases when the cached set of data has been incomplete for some reason. When the cached set of data is restored, the logged in API clients are informed of this by the BN5 broadcast that is generated from the Network Gateway. This broadcast brings information on those queries that the API client is recommended to issue.

/* BROADCAST TYPE = BN5 */

```
typedef struct history_cache_complete_item
{
   transaction_type_t query_type;
   char reserved_s [4];
} history_cache_complete_item_t;
typedef struct history_cache_complete
{
   broadcast_type_t broadcast_type;
   uint16_t items_n;
   char filler_2_s [2];
   history_cache_complete_item_t item [30];
} history_cache_complete_t;
```

The event is logged in the Gateway log with a tag of the user.

4.7 Transaction Identification

4.7.1 The OMnet Transaction Identification

The **`omniapi_tx_ex()`**, **`omniapi_query_ex()`** and **`omniapi_get_results()`** routines return the transaction identification. It is an eight-byte data structure that uniquely identifies a transaction (in native endian).

4.7.2 The Genium INET Order Identification

The order identification (an eight-byte data structure) uniquely identifies an order. It is returned from **`omniapi_tx_ex()`**, **`omniapi_query_ex()`** and **`omniapi_get_results()`** calls (in native endian).

Central applications use it to identify orders and requests. It is also used for later references to an order or a request. Matched orders are sent as events and identify the source orders by order identification.

For swapping of the Order Identification see Order Identifier on page 32.

4.8 Compression and Encryption

There is a possibility to set options whether compression or encryption should be used for a session. These options must be set before the login transaction is sent. The options can be set for a specific session or as a default option for all sessions that will be created after the option has been set.

Compression and encryption options are set up for each Gateway as well, which means that options set by the client application might not always be met. The options for the Gateway are set up by the exchange.

Compression and encryption may affect performance on the machine where it is being used.

When using encryption please read Appendix B – OpenSSL Open Source License on page 111.

4.9 API Call Timeout

In order for an API client application to be guaranteed not to hang in an API call, a call timeout can be used, for example all API calls are guaranteed to return within specified time. If an API call times out it will return an error code. If this occurs the session is terminated and the TCP/IP link to the Gateway is closed.

4.9.1 Configuration Parameters

An API call timeout can be set at system level. It is set by issuing the command in the command window of the machine using OMNI API:

Environment variable	Value	Description
`OAPI_TIMEOUT`	Specified time **Note:** If no value is set, a default value of 3600s is used.	The API call time-out is specified.
`OAPI_DATATIMEOUT`	Time interval in seconds. Default 10 seconds.	This value is specified to fine tune poor networks that have frequent outages more than 10 seconds.

5 Program Design

5.1 Function Return Status Codes

The return value (called the *C Status Code*) from an OMnet API function is used to send information and error status to the calling application. If the value is negative, the operation has failed.

C Status Code values are defined in the **omni.h** and **omniapi.h** header files. See *The Interface on page 35* in this document for descriptions of the different codes and their meaning.

A suitable test in C for the status of an OMnet API call would be:

```
csts_i = omniapi_tx_ex ( ... );
if ( csts_i >= 0 )
{
    /* success */
}
else
{
    /* problem */
}
```

5.1.1 Error Message Strings

All erroneous C Status Codes have a related text string. These strings are obtained by calling the **`omniapi_get_message_ex`** function. Note that a session must be established when calling this routine.

Error C Status Codes are described in *Error Messages on page 96*.

5.1.2 Generic Error Codes

There are a couple of error return codes that are not routine specific and the following error codes can be retrieved when calling any of the callable routines.

- OMNIAPI_FAILURE
- OMNIAPI_FATAL
- OMNIAPI_INTFAILURE
- OMNIAPI_BADARGVAL
- OMNIAPI_NOTCONNECTED
- OMNIAPI_NOSESSION
- OMNIAPI_ERRSEND
- OMNIAPI_SESINUSE
- OMNIAPI_NOT_LOGGED_IN

Note: OMNIAPI_BADARGVAL cannot be retrieved from routines with no arguments.

5.2 Commonly Used Data Structures and Data Types

The structures and types used by the OMnet API are described in the **omni.h** and **omniapi.h** files.

5.2.1 omni_message – The Generic OMnet Transaction Message

The **omni_message** type is used for passing transaction messages and queries to central applications.

Example 4:

```
typedef struct      /*omni transaction message*/
{
 uint32  length_u;  /*number of bytes of message data*/
} omni_message;
```

Contents of the omni_message:

Field name	Type	Interpretation
length_u	uint32	The length of the message starting immediately after omni_message.

5.2.2 omni_login_t

The **omni_login_t** structure is used together with the **`omniapi_login_ex()`** function for starting an OMnet session.

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;
```

The omni_login_t Definition:

Type	Name	Size	Mandatory	Description
Omni_username	user_s	32	Yes	The OMnet username string.
Omni_password	pass_s	32	Yes	The OMnet user password string.
String	gate-way_node_s	128*	Yes	Gateway host/node name or TCPIP address of the OMnet Gateway executor node.

Type	Name	Size	Mandatory	Description
Uint32_t	port_u	4	Yes	Port number of the OMnet Gateway process
String	Appl_ident_s	32*	No	Program id of the application Example: “QUICKTRADE V24-3”.
Uint8_t	forced_u	1	Yes	Flag for normal/forced login.

**Includes null termination, i.e. '\0'.*

Below follows a coding example of an OMnet LOGIN request:

```
/* Session Handler */
static omniapi_session_handle* hSession;
int32_t apitest_InitiateData( int argc, char* argv[] )
{
  char*    username;
  char*    password;
  char*    gatewayNode;
  uint32_t gatewayPort;
  int32_t  loginStatus;

  if( argc != 5 )
  {
   printf("Usage:   %s username password gatewayName gatewayPort\n", argv[0]);
    printf("Example:  %s testmm1 splunge piff 1112\n", argv[0]);
    return 1;
  }

  username = argv[1];
  password = argv[2];
  gatewayNode = argv[3];
  gatewayPort = atoi(argv[4]);

  hSession = apitest_createSession();
  hSession = omniapi_create_session(1);

  loginStatus = apitest_login(hSession, username, password, gatewayNode,
gatewayPort);
  if( loginStatus != omniapi_success )
  {
    printf("Couldn't log in. Completion status = %d\n", loginStatus);
    return 1;
  }
    return 0;
}
/*****************************************************************************
 *                                                                           *
 * Function: APITEST_Login                                                   *
 *                                                                           *
 * Description: This function sends a login request                          *
 *                                                                           *
 *                                                                           *
 * Input parameters:                                                         *
 *    username    I     User name                                            *
 *    password    I     Password                                             *
```

```
 *    node        I     Node                                                *
 *    port        I     Port                                                *
 *                                                                          *
 * Return value: Returns the completion status.                             *
 *                                                                          *
 * Comment: None                                                            *
 *                                                                          *
 ****************************************************************************/
int32 APITEST_Login( omniapi_session_handle* hSession,
                     char* username, char* password,
                     char* node, uint32_t port )
{
  omni_login_t    loginData;
  int32_t         completionStatus;
  int32_t         txStatus;
  /* Defensive programming, if a new version of the API
  * has an additional field in the login_message struct
  * it's preferable to have them zero filled*/

   memset(&loginData,0,sizeof(loginData));
   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 = port;
  strncpy(loginData.appl_ident_s, "APICourse", sizeof(loginData.appl_ident_s)
 );
   loginData.forced_u = LOGIN_NORMAL;
   completionStatus = omniapi_login_ex( hSession, &txStatus, &loginData );

   if (completionStatus == 0)
   {
     printf("Successfully logged in \n");
   }
     else if (completionStatus > 0)
   {
     printf("Successfully logged in with Completion status %d\n",
completionStatus);
   }
     else
   {
     printf("Couldn't log in. Completion status = %d\n", completionStatus);
   }

   return completionStatus;
}
```

5.2.3 Login

When logging in, the Gateway host/node name is given as a parameter. The API Client Application node has to, in a normal way, be able to resolve the IP number from the Gateway host/node name. If the Gateway node name is given as an IP-address, no resolution is necessary.

There is a possibility for the exchange to configure the exchange system so that re-login is not allowed. If a session already has been established for a specific user, a new login transaction will be rejected. The default behavior is that a new login with the same username will close the previous session.

5.2.4 Forced Login

Note:

Most Click/Secur/Genium systems are configured in such a way that a log-in request for a user that is already logged in will terminate the previous session and create a new one.

The only case in which the Forced flag is applicable is when you attempt to log in with the flag forced_u=LOGIN_NORMAL and a user ID that is already logged on and the attempt is rejected. This so-called “Forced log-in” procedure will be applicable so that the API application, for example, may decide to forcefully terminate the previous session and start a new one.

If a login attempt is rejected because the specific user is already logged in, the user may override the rejection by using a forced login request. Setting the forced_u flag in omni_login_t data struct to LOGIN_FORCED before sending the login request does a forced log-in request.

Another (deprecated) way is to use the facility type OMNI_FACTYP_FORCED_LOGIN instead of the OMNI_FACTYP_LOGIN.

If the user is not logged in, the forced login request works exactly the same way as a normal login.

5.2.5 Change Password

There are three ways to change the password:

1. Using the **`omniapi_set_newpwd_ex(..)`** function call. Both old/current password and a new one should be given.
2. Using the facility OMNI_FACTYP_SET_NEWPWD. As transaction data the omni_set_password_t is used. Both old/current password and a new one should be given. The transaction is sent by calling the **omniapi_tx_ex(..)** function. This gives the very same result as the **`omniapi_set_newpwd_ex(..)`** function call.
3. Using the facility OMNI_FACTYP_SET_PASSW. A character string is used with the **`omniapi_tx_ex(..)`** function to set the new password. Only the new password should be provided. This alternative is deprecated.

The first way is the preferred way to change the password.

If the password has expired when logging in, the session will get restricted access to the Genium INET system until the password has been successfully changed. See also Section 4.5 on page 21 and Section 6.3.4 on page 38.

If the password is close to expiration the completion code OMNIAPI_PWD_IMPEND_EXP is returned to the login request.

The number of days until expiration may be retrieved by calling **`omniapi_get_info(..)`** with OMNI_INFTYP_PWD_EXPIRATION as parameter.

5.2.5.1 Password Requirements

The new password must comply with the following requirements:

- The password must consist of characters in the ISO 8859-1 character set (8-bit single-byte coded graphic character sets)
- The password may only contain ASCII characters between decimal 32-255 (hex 0x20-0xFF) except for the % (percentage sign, decimal 37, hex 0x25) and \ (backslash, decimal 92, hex 0x5C).
- The password string has to be null ('\0') terminated
- The password must at least consist of one character.
- The length of the password may not exceed 32 characters including the null termination character.

The exchange may configure further restrictions on a password:

- Minimum password length
- The characters in the password has to be of a certain character set:
  - ASCII character set. Valid characters are hex 0x20-0xFF except for 0x25 and 0x5C.
  - Printable character set. Valid characters are hex 0x20 – 0x7E except for 0x25 and 0x5C.
  - Extended alphanumeric set. Valid characters are A-Z, 0-9 and ! @ # $ ^ & * ( ) - _ = + , < . > / ? ; : ' " [{] } |
- Minimum number of alphabetic character A-Z in the password.
- Minimum number of numerical characters 0-9 in the password.
- Minimum number of special characters in the password. Special characters are:

  ! @ # $ ^ & * ( ) - _ = + , < . > / ? ; : ' " [{] } |
- The password should not be found in a dictionary chosen by the exchange.
- Password expiration period
- Password history (preventing the user from re-using a password for a period of time).

The restrictions above are only checked when the password is changed.

5.3 Byte Swap

Numbers received from routines (either as return value or receive parameter) are always in native endian format. Numbers received in buffers (broadcasts, replies to queries, getinfo results) are always in little endian format. Use the PUTSHORT, PUTLONG, PUTQUAD and PUTORDERID to swap numbers.

For your convenience the data structures omni_login_t and omni_set_password_t are passed in native endian. If necessary, the library will perform the byte swapping.

5.3.1 Order Identifier

The 64-bit ordidt argument returned by **`omniapi_tx_ex`** is treated as two 32-bit integers, and therefore it is swapped as two 32-bit numbers. When swapping order identities in buffers, for example the reply to "get my orderbook", the needed swapping may be done with the PUTORDERID macro as follows:

```
PUTORDERID(orderBookAnswer->order_number_u,orderBookAnswer->order_number_u);
```

An alternative is to swap the order identifier as two 32-bit integers as follows:

```
uint32_t *orderPtr;
orderPtr = &orderBookAnswer->order_number_u;
PUTLONG(orderPtr[0],orderPtr[0]);
PUTLONG(orderPtr[1],orderPtr[1]);
```

Note: The ordidt received as a return parameter from **`omniapi_tx_ex`** is in native endian format, and order_number_u received in queries and broadcasts are in little endian format.

6 The Interface

6.1 Overview

The OMnet API callable routines are:

OMnet API routines	Explanation
omniapi_create_session	Creates a session object. The return value (**omniapi_session_handle**) is used as first parameter to those routines below that are handled by the OMnet Gateway.
omniapi_close_session	Closes a previously created session in the OMnet API.
omniapi_login_ex	Logs in the session.
omniapi_logout_ex	Logs out the session.
omniapi_set_newpwd_ex	Changes the password for the user.
omniapi_tx_ex	Transfers messages, such as orders, to the Genium INET System.
omniapi_query_ex	Used for requesting information from the Genium INET System.
omniapi_set_event_ex	Enables the reception of broadcast (asynchronous) messages.
omniapi_clear_event_ex	Disables the reception of broadcast messages.
omniapi_read_event_ext_ex	Polls broadcast. The broadcasts are queued within the OMnet Gateway until they get polled. It is also used to get information on available event types and broadcast subscriptions.
omniapi_get_message_ex	Translates a C Status Code to its related text string.
omniapi_get_info_ex	Retrieves information based on information request type.
omniapi_cvt_string	Converts an 8-bit character string from or to central format. The central OMnet applications, such as the Marketplace, use the ISO Latin-1 character set.
omniapi_cvt_int	Swaps the byte order of an integer, if the local processor has a different byte enumeration, big endian, from the little endian. (Intel processors use the same byte order as Alpha AXP, whereas SUN and HP processors use big endian byte order.) This routine is NOT available in OMnet API on little endian computers. For datatype swapping the PUT macros (defined in omniapi.h) should be used.
omniapi_convert_timestruct	Converts an omni_tm_t data struct to a struct tm.
PUTBYTE **PUTSHORT** **PUTLONG** **PUTQUAD** **PUTORDERID**	The OMnet protocol is in little endian format. These routines may be used for converting numbers on big endian platforms. See also Section 5.3.1 on page 32.
omniapi_read_event_block	Retreive broadcasts from Gateway, and block for a specified time period if no broadcasts are available.

6.2 The Environment Files

Three C inclusion files: **omni.h**, **omniapi.h** and **omnifact.h** define:

- literals
- error codes,
- structures
- data types and routine prototypes
- predefined OMnet facility types for the OMNIAPI

All OMnet API applications should include the environment files statements in the C program modules.

```
#include "omni.h"         /* for type definitions */
#include "omniapi.h"      /* for OMnet API environment */
#include "omnifact.h"     /* for OMnet facility types */
#include "omn_om_inttypes.h"  /* for redefinitions of general data types */
```

Most OMnet API applications require:

```
#include "omex.h"  /* for interaction with the Genium INET System */
```

6.2.1 omniapi.h

This file defines the environment for the use of the OMnet API programming interface. All OMnet API error messages are defined and all OMnet API functions are prototyped.

6.2.2 omni.h

This file defines the OMnet environment that is used by both the OMnet API application and an OMnet Gateway. This file holds the fundamental type definitions for writing portable C applications.

6.2.3 omnifact.h

This header file defines all standard facility and event types. The numbers, starting from 70, are recommendations, but may be altered by the OMnet operator/configurer.

6.2.4 omex.h

This file is exchange specific. It is not delivered within the API kit. It has to be obtained from the exchange. The omex.h file contains structures for transactions and queries used in the exchange system. Exchange-specific error codes are also defined in omex.h.

6.2.5 omn_om_inttypes.h/om_inttypes.h

The API passes binary structures between nodes, which makes it important that the size of the data types used are consistent between platforms. NASDAQ OMX has adopted the use of the C9X proposed standard for integer data types, **inttypes.h**, where, for example, a 32 bit unsigned integer is defined as **uint32_t**.

omn_om_inttypes.h and om_inttypes.h are provided and define the same data types for platforms/compilers that do not yet provide inttypes.h.

Recompiling old applications using the OMnet API might require updates to the data types used.

The file that should be included is omn_om_inttypes.h. When needed, depending on platform, it includes om_inttypes.h.

6.3 The Callable Routines

6.3.1 The Multi-Session API

Multiple sessions per process are supported. However, each session is strictly synchronous.

Example 5:

For one session, all calls to that session are preferably done in the context of the same thread. If you want to use the same session in multiple threads, you must use a synchronization mechanism to ensure that only one thread at a time uses that session.

If your application uses many sessions, you can use one session per thread instead of separating each session in its own process.

In Genium INET, the OMnet API supports concurrent broadcasts and transactions, wherein a single session can be used concurrently in a different thread purely for the purpose of subscribing and polling for broadcasts. For more details refer to Concurrent Broadcasts and Transactions on page 91.

6.3.2 omniapi_create_session – Create a session with the OMnet API

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

Format

hSession = omniapi_create_session ();

Returns

session

Type:	omniapi_session_handle
Access:	Write only
Mechanism:	By reference

Note: It is important to close the session before opening a new session. Each session occupies system resources that become available when it is closed.

6.3.3 omniapi_close_session – Close Session with the OMnet API

This routine closes the session with the API and the socket used.

Format

void omniapi_close_session (session);

Arguments

session

Type:	Omniapi_session_handle
Access:	Write/Read
Mechanism:	By reference

Note: It is important to log out before you close the session, otherwise the resources created for the login will not be released.

6.3.4 omniapi_login_ex – Login to Genium INET system

The **`omniapi_login_ex`** routine is used to log in

Format

omniapi_login_ex (session, txstat, login_data)

Description

The **omniapi_login_ex()** routine is used to log in to the Genium INET system in a simplified way.

The routine performs the same thing as if the Application creates a login transaction and sends it on OMNI_FACTYP_LOGIN/OMNI_FACTYP_FORCED_LOGIN.

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

txstat

Type:	Signed 32-bit integer
Access:	Write only
Mechanism:	By reference

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 Central System or OMnet Gateway is present.

login_data

Type:	omni_login_t
Access:	Read only
Mechanism:	By reference

Note: The omni_login_t should be passed in native endian. The library will take care of an eventual byte swap.

Return values (refer to the Messages chapter)

OMNIAPI_SUCCESS	Successful completion.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAPI_NOT_READY	The LOGIN is not yet complete. Wait and retry the function call.
OMNIAPI_NOBACKEND	The OMnet backbone is absent.
OMNIAPI_TX_ABORTED	The transaction was aborted.
OMNIAPI_TX_TIMEOUT	The transaction timed out and failed.
OMNIAPI_TX_DECLFAIL	The OMnet Gateway failed accessing a facility during LOGIN.
OMNIAPI_TX_FAILURE	Transaction failure; txstatus holds the ID identifying the problem.
OMNIAPI_DYNMEM	The OMnet Gateway failed to allocate dynamic memory.
OMNIAPI_NO_USR_OR_PASSW	Expected username or password string is missing.
OMNIAPI_FATAL	Fatal error in the internal OMnet API encountered by the OMnet Gateway.
OMNIAPI_OAPI	The internal OMnet interface detected an error; the txstat gives the cause of the problem.
OMNIAPI_BADNARGS	Bad number of arguments provided.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOGWYSRV	Unable to connect to gateway.

OMNIAPI_NONETWORK	No network available for OMnet Gateway access.
OMNIAPI_OSBADCONFIG	Operating system is not properly set up; for example, the network and system configurations may be different.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_PASSW_EXPIRED	The password has expired. A new password must be provided in the login transaction. It may not be identical to the old password.
OMNIAPI_INVALID_PASSW	The password is invalid. This message is only returned in response to a *set password* request.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_ERROPTCMPZ	Requested compression option rejected.
OMNIAPI_ERROPTENCRYPT	Requested encryption option rejected.
OMNIAPI_ERROPTCONC	Concurrent connection request rejected.
OMNIAPI_ERR_SECURE	Failed to establish secure session.
OMNIAPI_PWD_CHANGE_REQ	Password has expired. Restricted access until password is changed.
OMNIAPI_PWD_IMPEND_EXP	Password expiration is impending.
OMNIAPI_WRONG_TX_SIZE	The provided buffer size is not correct.
OMNIAPI_CALL_NOT_SUPPORTED	The call that was done is not supported by the exchange.
OMNIAPI_INVALID_LOGIN_ATTEMPT	Invalid log-in attempt.
OMNIAPI_INVALID_LOGIN_LOCKED	Login denied – user is locked.
OMNIAPI_INVALID_LOGIN_SUSPENDED	Login denied – user is suspended.
OMNIAPI_RELOGIN_NOT_ALLOWED	Login denied. The username is already in use. Re-login is not allowed.

Calling Example

The following program excerpt demonstrates the use of the **omniapi_login_ex** call:

```
/*****************************************************************************
 *                                                                           *
 * Function: APISAMPLE_Login                                                 *
 *                                                                           *
 * Description: This function sends a login request.                         *
```

```
 *                                                                          *
 * Input parameters:                                                        *
 *    hSession    I     Session handle                                      *
 *    username    I     User name                                           *
 *    password    I     Password                                            *
 *    node        I     Node                                                *
 *    port        I     Port                                                *
 *                                                                          *
 * Return value: Returns the completion status.                             *
 *                                                                          *
 * Comment: None                                                            *
 *                                                                          *
 ****************************************************************************/
int32_t APISAMPLE_Login( omniapi_session_handle *hSession, char* username,
                         char* password, char* node, char* port )
{
  omni_login_t loginData;
  omni_password newpassword;
  int32_t completionStatus;
  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;
  completionStatus = omniapi_login_ex( hSession, &txStatus, &loginData );
  if (completionStatus == 0)
  {
    printf("Successfully logged in \n");
  }
  else if (completionStatus > 0)
  {
    printf("Successfully logged in with Completion status %d\n",
completionStatus);
  }
  else
  {
    printf("Couldn't log in. Completion status = %d\n", completionStatus);
  }

  if (OMNIAPI_PWD_CHANGE_REQ == completionStatus)
  {
    do
    {
      printf("Your password has expired.\n"
             "Please enter a new one (1-31 characters): ");
      scanf("%s", newpassword);
      completionStatus = APISAMPLE_SetNewPassword( hSession, password,
newpassword);
    }
    while (completionStatus < 0);
```

```
  }
  return completionStatus;
}
```

6.3.5 omniapi_logout_ex – Logout from Genium INET system

The **`omniapi_logout_ex`** routine is used to logout a user.

Format

omniapi_logout_ex (session, txstat)

Description

The **omniapi_logout_ex()** routine is used to logout from the Genium INET system in a simplified way.

The routine performs the same thing as if the Application creates a logout transaction and sends it on OMNI_FACTYP_LOGOUT.

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

txstat

Type:	Signed 32-bit integer
Access:	Write only
Mechanism:	By reference

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 Central System or OMnet Gateway is present.

Return values (refer to the Messages chapter)

OMNIAPI_SUCCESS	Successful completion.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAP_NOBACKEND	The OMnet backbone is absent.

OMNIAPI_TX_ABORTED	The transaction was aborted.
OMNIAPI_TX_TIMEOUT	The transaction timed out and failed.
OMNIAPI_TX_DECLFAIL	The OMnet Gateway failed accessing a facility during LOGOUT.
OMNIAPI_TX_FAILURE	Transaction failure; txstatus holds the ID identifying the problem.
OMNIAPI_DYNMEM	The OMnet Gateway failed to allocate dynamic memory.
OMNIAPI_FATAL	Fatal error in the internal OMnet API encountered by the OMnet Gateway.
OMNIAPI_OAPI	The internal OMnet interface detected an error; the txstat gives the cause of the problem.
OMNIAPI_BADNARGS	Bad number of arguments provided.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOGWYSRV	Unable to connect to gateway.
OMNIAPI_NONETWORK	No network available for OMnet Gateway access.
OMNIAPI_OSBADCONFIG	Operating system is not properly set up; for example, the network and system configurations may be different.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN. The session handle is not logged in.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_PASSW_EXPIRED	The password has expired. A new password must be provided in the login transaction. It may not be identical to the old password.
OMNIAPI_INVALID_PASSW	The password is invalid. This message is only returned in response to a *set password* request.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_WRONG_TX_SIZE	The provided buffer size is not correct.

Calling Example

The following program excerpt demonstrates the use of the **omniapi_logout_ex** call:

```
/*****************************************************************************
 *                                                                           *
 * Function: APISAMPLE_Logout                                                *
 *                                                                           *
 * Description: This function sends a logout request.                        *
 *                                                                           *
 * Input parameters:                                                         *
 *    hSession    I     Session handle                                       *
 *                                                                           *
 * Return value: Returns the completion status.                              *
 *                                                                           *
 * Comment: None                                                             *
 *                                                                           *
 *****************************************************************************/
int32_t APISAMPLE_Logout( omniapi_session_handle *hSession )
{
  int32_t               completionStatus;
  int32_t               txStatus;
  completionStatus = omniapi_logout_ex ( hSession, &txStatus );
  if ( OMNIAPI_SUCCESS == completionStatus )
  {
    printf("Successfully logged out.\n");
  }
  else
  {
    printf("Error: Logout failed, Completion status = %d\n", completionStatus);
  }
  return completionStatus;
}
```

6.3.6 omniapi_set_newpwd_ex – Set a new password

The **`omniapi_set_newpwd_ex`** routine is used to set a new password.

Format

cstatus = omniapi_set_newpwd_ex (session, txstat, pwd_data)

Description

The **omniapi_set_newpwd_ex()** routine is used to set a new password in a simplified way.

The routine performs the same thing as if the Application creates a set password transaction and sends it on OMNI_FACTYP_SET_NEWPWD.

Returns

cstatus

Type:	Signed 32-bit integer with a C function completion code
Access:	Write only
Mechanism:	By value

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 and the *txstat* argument holds more detailed information about the problem.

Otherwise the operation has been successful and the completion code holds altering informational values. Simultaneously, the *txstat* argument may hold an application-dependent status message from the Central System.

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

txstat

Type:	Signed 32-bit integer
Access:	Write only
Mechanism:	By reference

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 Central System or OMnet Gateway is present.

pwd_data

Type:	omni_set_password_t
Access:	Read only
Mechanism:	By reference

Note: The omni_set_password_t should be passed in native endian. The library will take care of an eventual byte swap.

Return values (refer to the Messages chapter)

OMNIAPI_SUCCESS	Successful completion.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAPI_NOBACKEND	The OMnet backbone is absent.
OMNIAPI_TX_ABORTED	The transaction was aborted.
OMNIAPI_TX_TIMEOUT	The transaction timed out and failed.

OMNIAPI_TX_DECLFAIL	The OMnet Gateway failed accessing a facility when setting new password.
OMNIAPI_TX_FAILURE	Transaction failure; txstatus holds the ID identifying the problem.
OMNIAPI_DYNMEM	The OMnet Gateway failed to allocate dynamic memory.
OMNIAPI_FATAL	Fatal error in the internal OMnet API encountered by the OMnet Gateway.
OMNIAPI_OAPI	The internal OMnet interface detected an error; the txstat gives the cause of the problem.
OMNIAPI_BADNARGS	Bad number of arguments provided.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOGWYSRV	Unable to connect to gateway.
OMNIAPI_NONETWORK	No network available for OMnet Gateway access.
OMNIAPI_OSBADCONFIG	Operating system is not properly set up; for example, the network and system configurations may be different.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN. The session handle is not logged in.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_INVALID_PASSW	The password is invalid. This message is only returned in response to a *set password* request.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_INVALID_CHRS	The new password contains invalid characters.
OMNIAPI_NO_NULL_TERMINATION	The provided string is no null-terminated.
OMNIAPI_WRONG_TX_SIZE	The provided buffer size is not correct.
OMNIAPI_CALL_NOT_SUPPORTED	The call that was done is not supported by the exchange.

Calling Example

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

```
/*************************************************************************
```

```
 *                                                                          *
 * Function: APISAMPLE_SetNewPassword                                       *
 *                                                                          *
 * Description: This function sends a login request.                        *
 *                                                                          *
 * Input parameters:                                                        *
 *    hSession    I     Session handle                                      *
 *    username    I     User name                                           *
 *    password    I     Password                                            *
 *    node        I     Node                                                *
 *    port        I     Port                                                *
 *                                                                          *
 * Return value: Returns the completion status.                             *
 *                                                                          *
 * Comment: None                                                            *
 *                                                                          *
 ****************************************************************************/
int32_t APISAMPLE_SetNewPassword( omniapi_session_handle *hSession,
                                  char* currentPassword,
                                  char* newPassword )
{
  omni_set_password_t setNewpwd;
  int32_t completionStatus;
  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));
  completionStatus = omniapi_set_newpwd_ex( hSession, &txStatus, &setNewpwd);
  if (completionStatus == 0)
  {
    printf("Successfully set a new password \n");
  }
  else
  {
    printf("Set new password failed. Completion status = %d\n",
completionStatus);
  }

  return completionStatus;
}
```

6.3.7 omniapi_tx_ex – Issue an OMnet Transaction

The **`omniapi_tx_ex`** routine is primarily used for:

- session control
- entering orders
- sending instructions to clearing and settlement systems

Format

cstatus = omniapi_tx_ex (session, txstat, factyp, txmsgs, txidnt, ordidt)

Description

The **omniapi_tx_ex()** routine is primarily used for business transactions.

Returns

cstatus

Type:	Signed 32-bit integer with a C function completion code
Access:	Write only
Mechanism:	By value

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 and the *txstat* argument holds more detailed information about the problem, see Section 5.2.2 on page 28.

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

txstat

Type:	Signed 32-bit integer
Access:	Write only
Mechanism:	By reference

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

If cstatus	txstat contains
Is positive	a secondary status code from the central application. (For information on this type of completion, see the subsystem documentation for the specific central applications.)
Is negative	a code showing the cause of the failure.

factyp

Type:	Unsigned 32-bit integer
Access:	Read only
Mechanism:	By value

The facility types are described in *Section 3.3 on page 12*.

txmsgs

Type:	Vector of pointers

Access:	Read only
Mechanism:	By reference

A vector of pointers to message buffers that should be sent to the central Genium INET system. The vector must be NULL-terminated. Currently, one **omni_message** buffer is supported.

txidnt

Type:	Eight-byte buffer
Access:	Write only
Mechanism:	By reference

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

ordidt

Type:	Eight-byte buffer
Access:	Write only
Mechanism:	By reference

The **ordidt** argument holds a reference to an eight-byte order identification. Trading applications use this entity for recognition of matched orders (received as broadcasts). Note that the ordidt return parameter is in native endian format, while order identifiers received in broadcasts or queries are in little endian format.

Return values (refer to the Messages chapter)

OMNIAPI_SUCCESS	Successful completion; the txstat argument may hold a secondary message from the central application.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAPI_NOT_READY	The LOGIN is not yet complete. Wait and retry the function call.
OMNIAPI_FACID_NOT_VALID	Unrecognized facility type given.
OMNIAPI_NOT_LOGGED_IN	The OMnet Gateway considers the user to be logged out.
OMNIAPI_NOBACKEND	The OMnet backbone is absent.
OMNIAPI_TX_ABORTED	The transaction was aborted.
OMNIAPI_TRUNCATED	The provided receive buffer was too small.
OMNIAPI_NO_SUCH_ID	The transaction type of the message data was not recognized.
OMNIAPI_TX_TIMEOUT	The transaction timed out and failed.
OMNIAPI_TX_DECLFAIL	The OMnet Gateway failed accessing a facility during LOGIN.
OMNIAPI_TX_FAILURE	Transaction failure; txstatus holds the ID identifying the problem.

OMNIAPI_DYNMEM	The OMnet Gateway failed to allocate dynamic memory.
OMNIAPI_NO_USR_OR_PASSW	Expected username or password string is missing.
OMNIAPI_INVEVT	Invalid event type.
OMNIAPI_FATAL	Fatal error in the internal OMnet API encountered by the OMnet Gateway.
OMNIAPI_OAPI	The internal OMnet interface detected an error; the txstat gives the cause of the problem.
OMNIAPI_BADNARGS	Bad number of arguments provided.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOGWYSRV	Unable to connect to gateway.
OMNIAPI_NONETWORK	No network available for OMnet Gateway access.
OMNIAPI_OSBADCONFIG	Operating system is not properly set up; for example, the network and system configurations may be different.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN. The session handle is not logged in.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_PASSW_EXPIRED	The password has expired. A new password must be provided in the login transaction. It may not be identical to the old password.
OMNIAPI_INVALID_PASSW	The password is invalid. This message is only returned in response to a *set password* request.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_ERROPTCMPZ	Requested compression option rejected.
OMNIAPI_ERROPTENCRYPT	Requested encryption option rejected.
OMNIAPI_ERR_SECURE	Failed to establish secure csession.
OMNIAPI_PWD_CHANGE_REQ	Password has expired. Restricted access until password is changed.
OMNIAPI_PWD_IMPEND_EXP	Password expiration is impending.
OMNIAPI_WRONG_TX_SIZE	The provided buffer size is not correct.
OMNIAPI_CALL_NOT_SUPPORTED	The call that was done is not supported by the exchange.

OMNIAPI_INVALID_LOGIN_LOCKED	Login denied – user is locked.
OMNIAPI_INVALID_LOGIN_SUSPENDED	Login denied – user is suspended.
OMNIAPI_RELOGIN_NOT_ALLOWED	Login denied. The username is already in use. Re-login is not allowed.
OMNIAPI_INVALID_BEF_CHG_PWD	Password has expired. Restricted access to the Genium INET system until the password has been changed.

Calling Example

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

```
/*****************************************************************************
 *                                                                           *
 * Function: APISAMPLE_SendTx                                                *
 *                                                                           *
 * Description:                                                              *
 *    This is a wrapper function for the omniapi_tx_ex() OMnet function.     *
 *    The difference is that it takes a pointer to the actual transaction    *
 *    to be sent (and the length of the transaction message).                *
 *    (The omniapi_tx_ex() function expects a pointer to an array of         *
 *    pointers to transaction buffers, where the transaction buffers         *
 *    includes the length of each transaction message as its first           *
 *    four bytes.)                                                           *
 *                                                                           *
 * Input parameters:                                                         *
 *    buffer         I      buffer to send                                  *
 *    length         I      Length of the buffer                            *
 *    facilityType   I      Facility type                                   *
 *                                                                           *
 * Return value: Returns the completion status.                              *
 *                                                                           *
 * Comment:                                                                  *
 *    Note: The following global variables are updated by this function:     *
 *       glTransactionStatus                                                 *
 *       glTransaction_Id                                                    *
 *       glOrderId                                                           *
 *                                                                           *
 *****************************************************************************/
int32_t APITEST_SendTx( void* buffer, uint32_t length, uint32_t facilityType
)
{
   static omni_message* message[2]; /* only use the first item */
   static 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], (unsigned int *)&glOrderId);
   return retStatus;
}
```

6.3.8 omniapi_query_ex – Issue an OMnet Query

The **`omniapi_query_ex`** routine is used for sending a query request to a central application.

Format

cstatus = omniapi_query_ex (txstat, factyp, qrymsg, retflg, rcvbuf, rcvlen, txidnt, ordidt).

Description

The **omniapi_query_ex()** routine is used for sending queries to the central application. The normal use of the routine is to fetch information stored in the central system.

Returns

cstatus

Type:	Signed 32-bit integer with a C function completion code
Access:	Write only
Mechanism:	By value

See description of the **omniapi_tx_ex()** routine for an explanation.

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

txstat

Type:	Signed 32-bit integer
Access:	Write only
Mechanism:	By reference

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

If cstatus	Txstat contains
Is positive	a secondary status code from the central application. (For information on this type of completion, see the subsystem documentation for the specific central applications.)
Is negative	a code for the cause of the failure, see 5.3.4 omniapi_tx_ex – Issue an OMnet Transaction.

factyp

Type:	Unsigned 32-bit integer
Access:	Read only
Mechanism:	By value

The facility type is described in section *Section 3.3 on page 12*.

qrymsg

Type:	omni_message buffer
Access:	Read only
Mechanism:	By reference

This argument is a reference to an omni_message buffer.

retflg

Type:	Unsigned byte
Access:	Read only
Mechanism:	By value

This Boolean argument states whether an immediate reply is expected or not.

- If the immediate reply is **expected,** the **retflg** argument is set to one and the **rcvbuf** and **rcvlen** arguments must have valid references.
- If the immediate reply is **not expected,** the **retflg** is set to zero. (This is reserved for future use, currently not implemented)

rcvbuf

Type:	User buffer
Access:	Write only
Mechanism:	By reference

This argument is a reference to a buffer about to receive the resulting message.

rcvlen

Type:	Unsigned 32-bit integer
Access:	Read, Write
Mechanism:	By reference

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 o received data.

Note that 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 return.

txidnt

Type:	Eight-byte buffer
Access:	Write only
Mechanism:	By reference

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

ordidt

Type:	Eight-byte buffer
Access:	Write only
Mechanism:	By reference

The **ordidt** argument holds a reference to an eight-byte order identification.

Return Values (refer to the Messages chapter)

OMNIAPI_SUCCESS	Successful completion; the txstat argument may hold a secondary message from the central application.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAPI_NOT_READY	The LOGIN is not yet complete. Wait and retry the function call.
OMNIAPI_FACID_NOT_VALID	Unrecognized facility type given.
OMNIAPI_NOT_LOGGED_IN	The OMnet Gateway considers the user to be logged out.
OMNIAPI_NOBACKEND	The OMnet backbone is absent.
OMNIAPI_TX_ABORTED	The transaction was aborted.
OMNIAPI_TRUNCATED	The provided receive buffer was too small.
OMNIAPI_NO_SUCH_ID	The transaction type of the message data was not recognized.
OMNIAPI_TX_TIMEOUT	The transaction timed out and failed.
OMNIAPI_TX_DECLFAIL	The OMnet Gateway failed accessing a facility during LOGIN.
OMNIAPI_TX_FAILURE	Transaction failure; txstatus holds the ID identifying the problem.
OMNIAPI_DYNMEM	The OMnet Gateway failed to allocate dynamic memory.

OMNIAPI_NO_USR_OR_PASSW	Expected username or password string is missing.
OMNIAPI_INVEVT	Invalid event type.
OMNIAPI_FATAL	The OMnet Gateway encountered a fatal error in the internal OMnet API.
OMNIAPI_OAPI	The internal OMnet interface detected an error; txstat states the cause of the problem.
OMNIAPI_BADNARGS	Bad number of arguments provided.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOGWYSRV	Unable to connect to gateway.
OMNIAPI_NONETWORK	No network available for OMnet Gateway access.
OMNIAPI_OSBADCONFIG	Operating system not properly set up; for example, the network and system configurations are different.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN. The session handle is not logged in.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_INVALID_BEF_CHG_PWD	Password has expired. Restricted access to the Genium INET system until the password has been changed.

Calling Examples

```
/*****************************************************************************
 *                                                                           *
 * Function: APISAMPLE_Query                                                 *
 *                                                                           *
 * Description:                                                              *
 *    This is a wrapper function for the omniapi_query_ex() OMnet function.*
 *    The difference is that this function always expects a reply, and     *
 *    the send buffer and the buffer length is given separately (whereas   *
 *    OMnet expects the first four bytes of the buffer to specify the      *
 *    length of the rest of the buffer).                                   *
 *                                                                           *
 * Input parameters:                                                         *
 *    qryBuffer      I      Query buffer                                    *
 *    qryLength      I      Query buffer length  (size)                     *
 *    facilityType   I      Facility type                                   *
```

```
 *    rcvBuffer      I      Recive buffer                                    *
 *    rcvLenght      I      Recive buffer length (size)                      *
 *                                                                           *
 * Return value: The completion status.                                      *
 *                                                                           *
 * Comment:                                                                  *
 *    Note: The following global variables are updated by this function:     *
 *       glTransactionStatus                                                 *
 *       glTransactionId                                                     *
 *                                                                           *
 ****************************************************************************/
int32_t APISAMPLE_Query( void* qryBuffer, uint32_t qryLength,
                      uint32_t facilityType,void* rcvBuffer, uint32_t*
rcvLength )
{
 static int8_t sendBuffer[MAX_REQUEST_SIZE]; /* not thread safe */
 static omni_message* queryMessage;          /* not thread safe */
 quad_word dummyOrderId;
 int32_t retStatus;
 queryMessage = (omni_message*)sendBuffer;
 memcpy( &sendBuffer[0], &qryLength, sizeof(qryLength) );
 memcpy( &sendBuffer[sizeof(qryLength)], qryBuffer, qryLength );
 retStatus = omniapi_query_ex(hSession,&glTransactionStatus,
             facilityType,queryMessage, 1, rcvBuffer,rcvLength,
             &glTransactionId[0],(uint32_t*)&dummyOrderId);
                                          /* TRUE == yes, I want a reply */
 return retStatus;
}
```

6.3.9 omniapi_set_event_ex- Request Subscription of Broadcasts

The **`omniapi_set_event_ex()`** routine is used to set up a subscription to asynchronous messages events.

Format

cstatus = omniapi_set_event_ex (session, evtype , buffer)

Description

The **`omniapi_set_event_ex()`** routine used for setting up broadcast subscriptions.

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 the SHOW option.

> **Note:**
>
> The exchange may define some broadcasts as 'forced.' The Gateway will set up subscriptions for these broadcasts on behalf of the user, and they will thus be delivered to the API client even if there is no explicit subscription for them.
>
> User cannot set up more than 10,000 subscriptions.

If the API Client Application has already subscribed for a certain broadcast and tries to make another subscription for it, the Gateway will return the error code OMNIAPI_ ALR_SET. The handle of the already existing subscription is also returned.

Returns

cstatus

Type:	Signed 32-bit integer
Access:	Write only
Mechanism:	By value

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

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

evtype

Type:	Unsigned 32-bit integer
Access:	Read only
Mechanism:	By value

The central Genium INET system authorizes what event types a user may subscribe to. The event types can be:

- Central OMEX events (1).
- Network events (OMNI_EVTTYP_NETWORK).

Before reading events for a particular event type, the user must subscribe to the event type.

For finding the allowed event types, the user should call **`omniapi_read_event_ext_ex()`** with OMNI_ EVTTYP_SHOW as event type. The list is returned as a comma-separated, NULL-terminated string (”1,1001\0”). The user application must parse the list and **`call omniapi_set_event_ex()`** for each event type in it. Event type one (1) designates broadcasts from the central Genium INET system. In this case the subscription mechanism can be used (see the buffer argument below). For all other event numbers the buffer argument must be NULL.

To receive Dedicated Broadcasts one should set up subscriptions for event type= 1 and information object type = 4 (dedicated) and a valid such broadcast.

buffer

Type:	Optional pointer to structure (of type set_event_list_t)

Access:	Read, Write
Mechanism:	By reference

The first call to **`omniapi_set_event_ex()`** tells whether the subscription mechanism is to be used.

If the subscription **mechanism is not used** (that is, the *buffer* argument is NULL), the Gateway will automatically request subscription for all authorized information. In this case it is not possible to set up a new subscription using a subscription buffer.

If the subscription **mechanism is used**, the buffer argument is the address of a structure containing the subscription information. The subscription mechanism is only supported for event type one (1). In all other cases than event type one (1) the subscription mechanism will be ignored.

It is possible to request multiple subscriptions in one request. The buffer argument has the following layout:

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

- **Length field (4 (bytes)** – The length of the whole buffer, the length field included.
- **Item (20 bytes)** – An item identifies a subscription 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, whereas the subscription handle and request status will be completed by the API as a result of a call to the **`omniapi_set_event_ex().`** Identifies the selected information. For more information about information objects see *Broadcast Information Objects on page 13*.
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 authorized to subscribe to the requested information.

Return Values

OMNIAPI_SUCCESS	The reception of events is enabled.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN.
OMNIAPI_NOT_LOGGED_IN	Connected to OMnet Gateway, but not logged in (LOGOUT forced from the central OMnet).
OMNIAPI_NOTAUTH	You are not allowed to subscribe on this event type *or* invalid event code.

OMNIAPI_ALR_SET	The event or broadcast subscription has already been requested.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAPI_FATAL	The OMnet Gateway encountered a fatal error in the internal OMnet API.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_INVALID_BEF_CHG_PWD	Password has expired. Restricted access to the Genium INET system until the password has been changed.

Calling Examples

The following program demonstrates the use of the **`omniapi_set_event_ex()`** call setting up subscriptions for event type one (1) (general broadcasts). The first example shows how to subscribe to all authorized information objects. The second example shows how to subscribe to specific broadcasts.

```
/*****************************************************************************
 *                                                                           *
 * Function:      APISAMPLE_SubscribeAll                                     *
 *                                                                           *
 * Description:                                                              *
 *    This function asks OMnet for the available event types, and sets       *
 *    up a subscription for each one. Thus, we will now receive all the      *
 *    broadcasts we are allowed to see.                                      *
 *                                                                           *
 * Input parameters:    None                                                 *
 *                                                                           *
 * Return value:        None                                                 *
 *                                                                           *
 * Comment:             None                                                 *
 *                                                                           *
 *****************************************************************************/
void APITEST_SubscribeAll()
{
   uint32_t  buffSize; /* initial strbuffer size */
   int8_t   *strBuffer;
   int32_t   getEventStatus;
   int8_t   *strPtr;
   uint32_t  eventType;
   int32_t   completionStatus;
```

```
   getEventStatus = APITEST_GetAllEventTypes(&strBuffer, &buffSize);
   if( getEventStatus != OMNIAPI_SUCCESS )
   {
      return;
   }
   strPtr = &strBuffer[0];
   eventType = strtoul(strPtr, &strPtr, 10);
   completionStatus = 0;
   while( *strPtr != '\0'      &&
          eventType != 0       &&
          completionStatus >= 0)
   {
      completionStatus = omniapi_set_event_ex(hSession,eventType, 0 );
      strPtr++; /* skip the comma delimiter */
      if (strPtr != '\0')
      {
         eventType = strtoul(strPtr, &strPtr, 10);
      }
   }
   free(strBuffer);
   return;
}
/*****************************************************************************
 *                                                                           *
 * Function: APISAMPLE_ SubscribeIndividualEvents                            *
 *                                                                           *
 * Description:                                                              *
 *  This function sets up a subscription for a specific broadcasts.          *
 *  This function can easily be enhanced with several more broadcasts.       *
 *                                                                           *
 *                                                                           *
 * Input parameters: None                                                    *
 *                                                                           *
 * Return value: None                                                        *
 *                                                                           *
 * Comment:                                                                  *
 *  Forced broadcasts you'll get even if you not subscribed for them         *
 *                                                                           *
 *****************************************************************************/
int32_t APISAMPLE_SubscribeIndividualEvents()
{
   int32_t status;
   /*
    * Subscribe for Information source=43,
        Informationtype=7, Broadcast=BO14,
    * Commodity=11,  all expiration years, all expiration months
    */
   uint16_t infsrc = 43;
   uint16_t inftyp = 7;
   int8_t   module = 'B';
   int8_t   server_type = 'O';
   uint16_t broadcast_number_n = 14;
   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  = 11;  /* Used for inftyp = 3 */
   uint32_t member_info_n  = 0;        /* Used for inftyp = 4 */
   uint32_t dissemination_u = 0;       /* Used for inftyp = 5 */
   status = 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 status;
}
/*****************************************************************************
 *                                                                           *
 * Function: APISAMPLE_ SetEvent                                             *
 *                                                                           *
 * Description:                                                              *
 *    This function sets up a subscription for a specific broadcast          *
 *                                                                           *
 * Input parameters: None                                                    *
 *                                                                           *
 * Return value: None                                                        *
 *                                                                           *
 * Comment: None                                                             *
 *                                                                           *
 *****************************************************************************/
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;
   case 5:
      bcinfo->attrib_x.dissemination_x.dissemination_u = dissemination_u;
      break;
   default:
      /* Unknown inftyp */
      return -1;
   }

   return omniapi_set_event_ex( hSession, 1, (char*)&list);
}
/*****************************************************************************
 *                                                                           *
 * Function: APISAMPLE_GetAllEventTypes                                      *
 *                                                                           *
 * Description: Fetch the possible event types from the system               *
 *                                                                           *
 * Input parameters:                                                         *
 *    strBuffer      O     buffer with all event types                       *
 *    buffSize       O     Size of the buffer allocated                      *
 *                                                                           *
 * Return value:                                                             *
 *    0  OK                                                                  *
 *   <0  Error                                                               *
 *                                                                           *
 * Comment: The buffer will be allocated untill it's large enough            *
 *                                                                           *
 *****************************************************************************/
int32_t APITEST_GetAllEventTypes(char **strBuffer, uint32_t *buffSize)
{
   int32_t    completionStatus;
   glTransactionStatus = 0; /* Clear it, since we return
```

```
                                a completion status */
   *buffSize = 100;
   *strBuffer = (char *)malloc(*buffSize);
   completionStatus = omniapi_read_event_ext_ex( hSession,
                        OMNI_EVTTYP_SHOW,*strBuffer, buffSize, 0, 0);
   while( completionStatus == OMNIAPI_TRUNCATED )
                           /* if the buffer is too small...*/
   {
      free(*strBuffer);    /* ... double it until it's large enough  */
      *buffSize *= 2;
      *strBuffer = (char *)malloc(*buffSize);
      completionStatus = omniapi_read_event_ext_ex(hSession,
                         OMNI_EVTTYP_SHOW, *strBuffer, buffSize, 0, 0);

   }
   if( completionStatus != OMNIAPI_SUCCESS )
   {
      free(*strBuffer);
      printf("Error: Couldn't send ready to trade.
                         Completion status = %d\n",
                         completionStatus);
   }
   return completionStatus;
}
```

6.3.10 omniapi_read_event_ext_ex – Read Events

This routine must be called regularly to read events (broadcast messages).

Format

cstatus = omniapi_read_event_ext_ex (hSession, evtype, rcvbuf, rcvlen, evtmsk, optmsk)

Description

This routine reads events (that is, broadcasts) from the OMnet Gateway process.

Note: The API client is obliged to read the events it has set up subscriptions for in a timely manner. A good method is to use nested loops, where the outer loop polls with some reasonable time interval, such as 250 milliseconds, and the inner loop just iterates as long as there is data available.

Returns

cstatus

Type:	Signed 32-bit integer with a C function completion code
Access:	Write only
Mechanism:	By value

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 will be **either** the negative OMNIAPI_NOT_FOUND **or** the positive OMNIAPI_ALLEVTS. The OMNIAPI_NOT_FOUND is not to be regarded as an error.

Arguments

hSession

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

evtype

Type:	Unsigned 32-bit integer
Access:	Read only
Mechanism:	By value

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". The valid event types are loaded into the internal programming interface of the OMnet Gateway from the central components of OMnet.

rcvbuf

Type:	Pointer to a structure
Access:	Write only
Mechanism:	By reference

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

rcvlen

Type:	Unsigned 32-bit integer
Access:	Read, Write
Mechanism:	By reference

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 OMnet Gateway.

evtmsk

Type:	Unsigned 32-bit integer

Access:	Reserved for future use
Mechanism:	By reference

This argument is reserved and currently ignored.

optmsk

Type:	Unsigned 32-bit integer
Access:	Read
Mechanism:	By value

The **optmsk** argument is a bit mask value stating optional behavior 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 **rcvbuf** arguments, is completed with chained broadcast messages (events).

Field	Explanation
Lx, 2 bytes	The *Lx* length fields are two-byte little endian unsigned integers. (Big endian platforms such as Sun, HP and IBM AIX 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 bytes) boundary. The preceding length field includes the padding.
SHx, 4 bytes	Added for all “read event”, if the application has specified one or several subscriptions when **`omniapi_set_event_ex()`** was called. No valid information will be stored in this position.

The rcvbuf with the READEV_OPTMSK_MANY option and subscriptions will have the following format (mapping to the definition **subscribed_event_t**, which is 6 bytes in size):

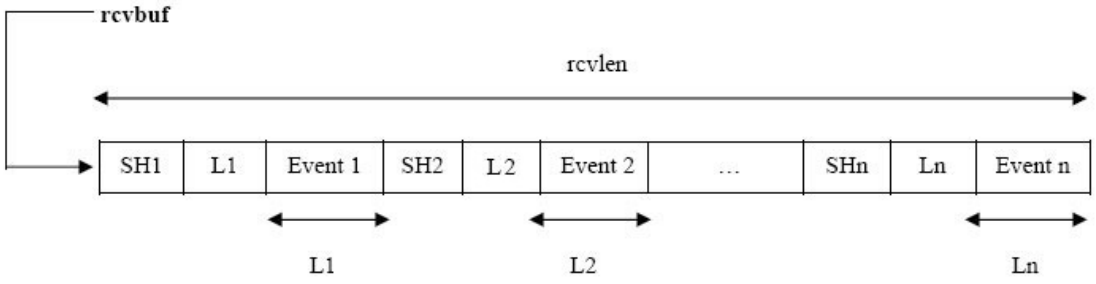

Figure 3: rcvbuf Format

If the application requests to read a single message, that is, the least significant bit is cleared in the **optmsk** parameter and subscriptions were specified in the call to **`omniapi_set_event_ex()`**, the **rcvbuf** will have the following format:

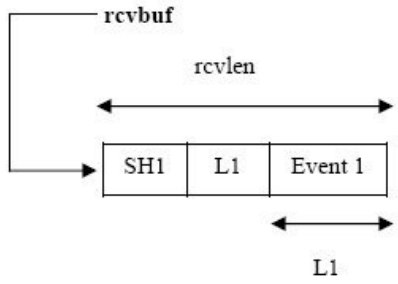

Figure 4: rcvbuf Format

If the subscription is not used [**`omniapi_set_event_ex()`**] was invoked with 0 as parameter), the buffer will look like this:

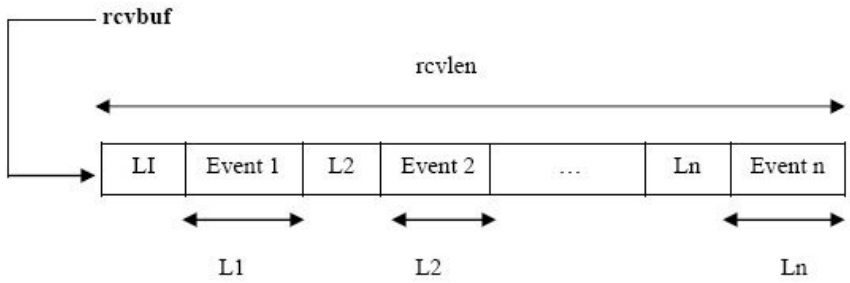

Figure 5: rcvbuf Format

And like this when a single message is requested:

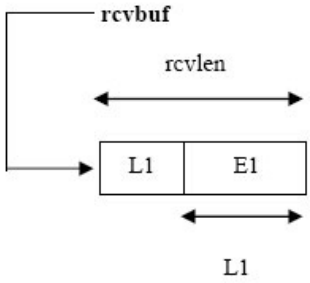

Figure 6: rcvbuf Format

Each event is always padded with zeros to a long word (4 bytes) boundary. The preceding length field includes the padding. The last item of the buffer is empty and filled with zeros.

Setting up and clearing subscriptions

The header of an event is always either six or two bytes as described above. The following rules applies when setting up or clearing subscriptions:

- Only subscriptions concerning event type 1 affect the size of the event header.

Setting up subscriptions

When setting up subscriptions, the following rules applies:

- Only the first subscription set up, when there are no subscriptions, 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 will be six bytes (subscibed_event_t structure)
- If the first subscription is a subscription for event type 1 (i.e. a NULL buffer is provided as third argument to the **`omniapi_set_event_ex routine`**) the event header size will be two bytes (length of event).

Example 6:

If a subscription for event type 1 is set up at first, and then a subscription for a specific event, the event header will still be two bytes in length.

Example 7:

If a subscription for a specific broadcast is set up at first, and then a subscription for event type 1, the event header will still be six bytes in length.

Clearing subscriptions

When clearing subscriptions the following rules applies:

- If the event header size is six bytes, and the subscription for event type 1 is cleared the event header will be two bytes in size from that point on.
- If the event header size is six bytes, and the last specific subscription is cleared, not taking forced subscriptions in account, the event header size will be two bytes from that point on.
- The event header size can not change from two bytes to any other size when unsubscribing for events.

Example 8:

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

Hints

- When fetching broadcasts, if the length is –1 (signed) or 64000 (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 (when there is no subscriptions), the header will always be two bytes. This means that if only specific subscriptions are wanted, there is a possibility to set up a subscription for event type one, then the specific subscriptions are set up and then the subscription for event type one is cleared. This will always give a header of two bytes even if subscriptions are added or removed, until the last specific subscription is removed and new ones set up.

omniapi_read_event_ext_ex event types

The **evtype** argument depicts the type and number of messages to be read or fetched:

Value	Type of result
OMNI_EVTTYP_ALL	Any event message.
OMNI_EVTTYP_SHOW_SUBSCR	List all information objects that the application is authorized to subscribe for.
OMNI_EVTTYP_SHOW	List of event types enabled by central configuration.
OMNI_EVTTYP_NETWORK	Network status message, see Section 4.6.1 on page 22.

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

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

```
"1001, 1, 990"
```

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 authorized information objects (auth_infobj_t) will be returned. The list contains an 32-bit number specifying the number of authorized information objects returned, with the objects following the number:

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

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

Information Object (infobj_t) (*12 bytes*)	Force flag (*4 bytes*)

- 0 = not automatically subscribed by OMnet Gateway
- 1 = automatically subscribed by OMnet Gateway

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

If the *force flag* field has a value 1, the OMnet API will automatically request a subscription for that particular information object. The application is not supposed to clear these subscriptions.

Return Values

OMNIAPI_SUCCESS	Successful completion; more events may exist.
OMNIAPI_ALLEVTS	All events collected.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN.

OMNIAPI_NOT_LOGGED_IN	Connected to OMnet Gateway, but not logged in (LOGOUT forced from the central OMnet).
OMNIAPI_NOT_FOUND	Requested data were not found or there is no broadcast available. If there is no broadcast available, this is not an error.
OMNIAPI_OVERFLOW	Success, but at least one event message was lost.
OMNIAPI_INVEVT	An invalid (not enabled) event type was specified.
OMNIAPI_TRUNCATED	The provided buffer was too small, so the received message was truncated.
OMNAPI_NOT_AUTH	You are not allowed to read this event type.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAPI_FATAL	The OMnet Gateway encountered a fatal error in the internal OMnet API.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.

Calling Example

The example below polls broadcasts and prints the broadcast type on the console.”

```
/*****************************************************************************
*                                                                            *
* Function:      APITEST_PollBroadcasts                                      *
*                                                                            *
* Description:                                                               *
*    This function polls OMnet and fills an internal buffer with all the     *
*    broadcasts currently waiting to be distributed by OMnet.                *
*    The polling is continued for 30 seconds with a polling interval of      *
*    one second.                                                            *

*                                                                            *
* Input parameters:     None                                                 *
*                                                                            *
* Returns the completion status.                                             *
*                                                                            *
* Comment:                                                                   *
*    The buffer size is changed by the read_event_ext_ex call,               *
*    so we need to reset it in the loop                                      *
```

```
 ****************************************************************************/
int32_t APITEST_PollBroadcasts()
{
 static char buffer[MAX_RESPONSE_SIZE];
 int32_t     completionStatus, secsElapsed;
 uint32_t    buffSize;
   completionStatus = OMNIAPI_SUCCESS;
   secsElapsed = 0;
   while( secsElapsed < 30)
   {
      buffSize = sizeof(buffer);
     completionStatus = omniapi_read_event_ext_ex( hSession, OMNI_EVTTYP_ALL,
                                                   buffer,  &buffSize,
                                                   0,
READEV_OPTMSK_MANY);
      printf("omniapi_read_event_ext_ex : PollBroadcasts: status=%d\n",
completionStatus);

      if( completionStatus >= 0 )
      {
         APITEST_ShowBroadcasts( buffer, buffSize );
      }
      else if ( completionStatus != OMNIAPI_SUCCESS && completionStatus !=
OMNIAPI_NOT_FOUND)
      {
          break;
      }
      APITEST_Sleep(1);
      ++secsElapsed;
   }
   return completionStatus;
}
/****************************************************************************
 *                                                                          *
 * Function:      APITEST_ShowBroadcasts                                    *
 *                                                                          *
 * Description:                                                             *
 *                                                                          *
 * This function steps through all broadcasts the specified buffer         *
 * contains.                                                                *
 *                                                                          *
 * Input parameters:                                                        *
 *    buffer      I     Broadcase buffer                                    *
 *    buffsize    I     Broadcast buffer size                               *
 *                                                                          *
 * Return value:   None                                                     *
 *                                                                          *
 * Comment:        None                                                     *
 *                                                                          *
 ****************************************************************************/
void  APITEST_ShowBroadcasts( char *buffer, uint32_t buffsize )
{
typedef struct broadcast_type
{
   char central_module_c;
```

```
   char server_type_c;
   uint16_t transaction_number_n;
} broadcast_type_t;
   char             *current;
   uint16_t          tickLength;
   broadcast_type_t *bc;
   uint16_t          swap_transaction_number_n;
   current = buffer + 4;
   tickLength = *(uint16_t *)current;
   PUTSHORT(tickLength, tickLength);
   while( tickLength > 0 )
   {
      current = current + sizeof( uint16 );       /* step to the start of the
 actual broadcast */

      bc = (broadcast_type_t*)current;
      swap_transaction_number_n = bc->transaction_number_n;
      PUTSHORT(swap_transaction_number_n,swap_transaction_number_n);
      printf("Brodcast received : <%c%c%i>\n",bc->central_module_c,
bc->server_type_c, swap_transaction_number_n);

      current = current + tickLength + 4;           /* step to the beginning
of the next header */
      tickLength = *(uint16 *)current;
      PUTSHORT(tickLength, tickLength);
   }
}
```

6.3.11 omniapi_clear_event_ex – Cancel Event Subscription

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

Format

cstatus = omniapi_clear_event_ex (hSession, evtype, buffer)

Description

This call will cancel reception of events for the event type specified.

> **Note:** If buffer argument is used, all objects in the buffer are cancelled from 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 buffer is filled with the actual objects.

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

> **Note:** Subscription mechanism is only supported for event type 1 (general events).

Returns

cstatus

Type:	Signed 32-bit integer with a C function completion code
Access:	Write only
Mechanism:	By value

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

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

evtype

Type:	Unsigned 32-bit integer
Access:	Read only
Mechanism:	By value

This argument specifies the event type (or "tick") number of the events for which the broadcasts will be cancelled.

buffer

Type:	Pointer to structure
Access:	Read, Write
Mechanism:	By reference

The first four bytes declare the size of 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().`**

L	H1	…	Hn

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

If the value is specified as zero, all active subscriptions will be cleared except Auto subscription (see *Section 6.3.10 on page 63*).

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

Return Values

OMNIAPI_SUCCESS	The reception of events is enabled.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN.
OMNIAPI_NOT_LOGGED_IN	Connected to OMnet Gateway, but not logged in. (LOGOUT forced from the central OMnet).
OMNIAPI_INVEVT	An invalid event code was specified or the event has already been cleared.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAPI_FATAL	The OMnet Gateway encountered a fatal error in the internal OMnet API.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_INVALID_BEF_CHG_PWD	Password has expired. Restricted access to the Genium INET system until the password has been changed.

Calling Examples

The following program demonstrates the use of the **omniapi_clear_event_ex** call to cancel a subscription for events type 1 (general broadcast events). The example will clear an information object earlier subscribed to.

```
/*****************************************************************************
 *                                                                           *
 * Function: APISAMPLE_ClearSubscription                                     *
 *                                                                           *
 * Description: This function clears a subscription                          *
 *                                                                           *
 * Input parameters:                                                         *
 * subscriptionHandle I     Handle to be renewed                             *
 *                                                                           *
 * Return value: clearStatus                                                 *
 *                                                                           *
```

```
 * Comment: None                                                              *
 *                                                                            *
 ******************************************************************************/
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;
}
```

6.3.12 omniapi_get_info_ex – Get OMnet Environmental Information

This routine returns an informational string, associated with the information type requested.

Format

cstatus = omniapi_get_info_ex (session, reason_pi, inftyp_u, inflen_pu, infbuf_ps)

Description

This is a generic routine for retrieving OMnet-related information. The calling interface is somewhat similar to **`omniapi_get_message_ex`**.

For example, this routine may be used for retrieving the numeric value for a facility type. See Section 3.3 on page 12 for more information.

Restriction: this call can only be performed when logged in.

Returns

cstatus

Type:	Signed 32-bit integer with a C function completion code
Access:	Write only
Mechanism:	By value

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

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

reason_pi

Type:	Signed 32-bit integer
Access:	Write
Mechanism:	By reference

Upon completion, this argument contains additional information about the cause of a call failure. The message text can be picked up with the **`omniapi_get_message_ex()`** routine.

inftyp_u

Type:	Unsigned 32-bit integer
Access:	Read
Mechanism:	By value

This argument specifies the type of information requested. The following information types are known:

OMNI_INFTYP_USERCODE	Gets information about the user code assigned to this logged-in user; the information buffer will be assigned a twelve-byte characters array.
OMNI_INFTYP_FACTYP_E0	Retrieves the facility number for the first business facility on this exchange system. The information buffer will be assigned a four-byte integer.
OMNI_INFTYP_FACTYP_I0	Retrieves the facility number for the first internal facility on this exchange system (applicable for internal applications). The information buffer will be assigned a four-byte integer.
OMNI_INFTYP_OMEXVERSION	Gets information about current OMnet version. The version will be presented as a character string. The information buffer should be assigned a 128-byte character array.
OMNI_INFTYP_BANDWIDTH	Gets information about the bandwidth limitation and usage, see chapter Programming Considerations on page 19.
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 if the information about the compression setting for the session. The buffer should be assigned a 4-byte integer. The value returned will denote either OMNIAPI_OPVAL_ENABLE or OMNIAPI_OPVAL_DISABLE, for compression enabled or disabled respectively.
OMNI_INFTYP_CONCURRENT_BDX	Gets the information about the concurrent broadcast support on the gateway. The buffer should be assigned a 4-byte integer. The value returned will denote OMNIAPI_OPVAL_ENABLE if the gateway supports concurrent broadcast feature. For more

	details on concurrent broadcasts, refer to Concurrent Broadcasts and Transactions on page 91.
OMNI_INFTYP_ENCRYPTION	Gets the information about the encryption setting for the session. The buffer should be assigned a 4-byte integer. The value returned will denote either OMNIAPI_OPVAL_ENABLE or OMNIAPI_OPVAL_DISABLE, for encryption enabled or disabled respectively.
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, the password has expired that number of days ago. When the password is set using **omniapi_set_newpwd_ex**, a new expiration date is set within the system. It will take up to ten seconds before the update is finished. Therefore retrieval of password expiration should not be performed immediately after the password has been set. 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. (Either the Gateway or the Central System is older. May appear during an upgrade period.)
OMNI_INFTYP_TIME_UTC	Gets the UTC time of the Central System. Returned data struct is omni_time_info_t.
OMNI_INFTYP_TIME_LOCAL	Gets the local time of the Central System. Returned data struct is omni_time_info_t.

inflen_pu

Type:	Unsigned 32-bit integer
Access:	Read and Write
Mechanism:	By reference

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

Type:	Context dependent
Access:	Write
Mechanism:	By reference

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

Return Values

OMNIAPI_SUCCESS	Successful completion.
OMNIAPI_INVARG	At least one of the arguments was invalid, most likely because a null pointer value was provided.
OMNIAPI_NOT_FOUND	Information about the requested information type was not found; an invalid information type was requested or no data was found.
OMNIAPI_TRUNCATED	The information returned did not fit into the provided buffer.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAPI_FATAL	The OMnet Gateway encountered a fatal error in the internal OMnet API.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN.
OMNIAPI_NOT_LOGGED_IN	The OMnet Gateway considers the user to be logged out.
OMNIAPI_NOT_APPLICABLE	The requested information is not applicable to the user or session.

Example

```
/*****************************************************************************
 *                                                                           *
 * Function: APISAMPLE_GetFacilityTypes                                      *
 *                                                                           *
 * Description:                                                              *
 *    Fetch the facility types from the system and save them                 *
 *                                                                           *
 * Input parameters: None                                                    *
 *                                                                           *
 * Return value:                                                             *
 *    0  OK                                                                  *
```

```
 *   <0  Error                                                              *
 *                                                                          *
 * Comment:                                                                 *
 *    All facility types will not be needed (and not be valid for this     *
 *    system).                                                              *
 *                                                                          *
 ****************************************************************************/
int32_t APISAMPLE_GetFacilityTypes()
{
   uint32_t factypeSize;
   int32_t completionStatus;
   factypeSize = sizeof(glFacilityType_EP0);
   completionStatus = omniapi_get_info_ex(hSession,&glTransactionStatus,
                     OMNI_INFTYP_FACTYP_E0,&factypeSize,&glFacilityType_EP0);
 if( completionStatus != OMNIAPI_SUCCESS )
   {
      printf("Error: Couldn't get Facility type. Completion status = %d\n",
             completionStatus);
      return completionStatus;
   }
 return OMNIAPI_SUCCESS;
}
```

6.3.13 omniapi_get_message_ex – Get an OMnet Exchange Message

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

Format

cstatus = omniapi_get_message_ex (session, msgcod, msgstr, msglen, simple)

Description

All messages in the *OMnet System Error Messages Reference* document are returned, except positive OMnet API routine return codes.

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

Returns

cstatus

Type:	Signed 32-bit integer with a C function completion code
Access:	Write only
Mechanism:	By value

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

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

msgcod

Type:	Signed 32-bit integer
Access:	Read only
Mechanism:	By value

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

msgstr

Type:	String
Access:	Write only
Mechanism:	By reference

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

msglen

Type:	Unsigned 32-bit integer
Access:	Read, Write
Mechanism:	By reference

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

- **On input**: the length of the caller's provided buffer
- **On completion**: the length of the string received from the OMnet Gateway.

simple

Type:	Signed 32-bit integer
Access:	Read
Mechanism:	By value

This variable is not used.

Return Values

OMNIAPI_SUCCESS	Successful completion, more events may exist.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN.
OMNIAPI_NOT_FOUND	Requested data not found.
OMNIAPI_TRUNCATED	The provided buffer was too small, so the received message was truncated.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAPI_FATAL	The OMnet Gateway encountered a fatal error in the internal OMnet API.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_NOT_LOGGED_IN	The OMnet Gateway considers the user to be logged out.

Example

```
/*****************************************************************************
 *                                                                           *
 * Function: APISAMPLE_ErrorTest                                             *
 *                                                                           *
 * Description: Enter a error Message and print out the text                 *
 *                                                                           *
 * Input parameters:                                                         *
 *     errorID I error identity                                              *
 *                                                                           *
 * Return value: None                                                        *
 *                                                                           *
 * Comment: None                                                             *
 *                                                                           *
 *****************************************************************************/
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);
   }
}
```

6.3.14 omniapi_set_option_ex

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

- Compression
- Encryption
- Concurrent Broadcast

Format

cstatus = omniapi_set_option_ex (session, opttype, optval)

Description

This routine is used for setting up network options, such as compression, encryption, and concurrent broadcast. This routine sets options for already created sessions, and the options can only be set before a login transaction has been sent.

Returns

cstatus

Type:	Signed 32-bit integer with a C function completion code
Access:	Write only
Mechanism:	By value

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

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

opttype

Type:	Signed 32-bit integer
Access:	Read only
Mechanism:	By value

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

optvalue

Type:	Signed 32-bit integer
Access:	Read only
Mechanism:	By value

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

Encryption

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

OMNIAPI_OPVAL_ANY	This is the default value. The setting for the Gateway is used.
OMNIAPI_OPVAL_ENABLE	The API client demands encryption, and if encryption is disabled in the Gateway or not implemented in the API on the current platform the login will be rejected.
OMNIAPI_OPVAL_DISABLE	The API client demands that encryption must not be used. If the Gateway has enforced encryption, the login will be rejected.

The following diagram show how the connection will be set up with different settings:

	Gateway encryption setting		
API Client encryption setting	**ANY**	**REQUIRED**	**DISALLOW**
OMNIAPI_OPVAL_ANY	No encryption	Encryption	No encryption
OMNIAPI_OPVAL_ENABLE	Encryption	Encryption	Connection is rejected
OMNIAPI_OPVAL_DISABLE	No encryption	Connection is rejected	No encryption

Compression

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

OMNIAPI_OPVAL_ANY	This is the default value. The setting for the Gateway is used.

OMNIAPI_OPVAL_ENABLE	The API client demands compression. If compression is disabled in the Gateway the login will be rejected.
OMNIAPI_OPVAL_DISABLE	The API client demands that compression must not be used. If the Gateway has enforced compression, the login will be rejected.

The following diagram show how the connection will be set up with different settings:

	Gateway compression setting		
API Client 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

Concurrent Broadcasts

When enabling or disabling concurrent broadcast feature, the OMNIAPI_OPT_CONCURRENT_BDX is used as the opttype parameter.

Valid values for Concurrent Broadcast options are:

OMNIAPI_OPVAL_ENABLE	Enable concurrent broadcast feature for the current session. An error will be returned If the gateway does not support concurrent broadcast feature or if the feature is already enabled for the session.
OMNIAPI_OPVAL_DISABLE	Disables concurrent broadcast option for the session. The function will return an error if concurrent broadcast feature is already disabled.

For more details on using concurrent broadcast feature refer to Concurrent Broadcasts and Transactions on page 91.

Return Values

OMNIAPI_SUCCESS	Successful completion, option set.
OMNIAPI_FATAL	The OMnet Gateway encountered a fatal error in the internal OMnet API.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN.

OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_NOT_LOGGED_IN	The OMnet Gateway considers the user to be logged out.
OMNIAPI_ERROPTENCRYPT	Requested encryption option rejected.

6.3.15 omniapi_set_option_default

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

Format

cstatus = omniapi_set_option_default(opttype, optval)

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 session, which already has been created.

Returns

cstatus

Type:	Signed 32-bit integer with a C function completion code
Access:	Write only
Mechanism:	By value

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

Arguments

opttype

Type:	Signed 32-bit integer
Access:	Read only
Mechanism:	By value

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

optvalue

Type:	Signed 32-bit integer
Access:	Read only
Mechanism:	By value

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

For argument explanation, see omniapi_set_option_ex on page 81.

Return Values

OMNIAPI_SUCCESS	Successful completion, option set.
OMNIAPI_FATAL	The OMnet Gateway encountered a fatal error in the internal OMnet API.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN. The session handle is not logged in.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.
OMNIAPI_NOT_LOGGED_IN	The OMnet Gateway considers the user to be logged out.
OMNIAPI_ERROPTENCRYPT	Requested encryption option rejected.

6.3.16 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.

Format

void omniapi_cvt_int (number, numsiz)

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.

Arguments

number

Type:	Unspecified 32-bit integer
Access:	Read, Write

Mechanism:	By reference

The *number* argument references the integer about to be converted.

numsiz

Type:	Signed 16-bit word
Access:	Read only
Mechanism:	By value

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

6.3.17 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 central application.

Format

cstatus = omniapi_cvt_string (toctrl, ebtstr)

Description

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

Arguments

toctrl

Type:	Signed byte
Access:	Read only
Mechanism:	By value

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

- A Boolean TRUE, normally an integer 1, represents that the conversion is TO the central format.
- A Boolean FALSE, integer 0, represents FROM the central format.

ebtstr

Type:	8-bit NULL-terminated string
Access:	Read, Write
Mechanism:	By reference

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

Example 9:

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

6.3.18 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.

Format

struct tm* omniapi_convert_timestruct(omni_tm_t* omni_tm_p, struct tm* tm_p)

Description

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

Arguments

omni_tm_p

Type:	Pointer to a omni_tm_t
Access:	Read
Mechanism:	By reference

The omni_tm_t struct is in OMnet endian, which is in little endian.

tm_p

Type:	Pointer to a struct tm
Access:	Write
Mechanism:	By reference

The struct tm is in native endian, that is the endian of the API Client Application plattform.

Return Values

The tm_p pointer is returned.

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

6.3.19 omniapi_read_event_block-Blocking Read Event

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

Format

cstatus = omniapi_read_event_block (session, evtype, rcvbuf, rcvlen, timeout, waittype);

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 should be enabled for the session. For more details on using the concurrent broadcast feature, see Concurrent Broadcasts and Transactions on page 91.

Returns

cstatus

Type:	Signed 32-bit integer with a C function completion code
Access:	Write only
Mechanism:	By value

The completion code is used for checking the operation of the call. If **cstatus** is negative, the call has failed. The completion code gives the hint about the problem.

Arguments

session

Type:	omniapi_session_handle
Access:	Read only
Mechanism:	By reference

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

evtyp

Type:	Unsigned 32-bit integer
Access:	Read only
Mechanism:	By value

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

rcvbuf

Type:	Pointer structure
Access:	Write only
Mechanism:	By reference

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

rcvlen

Type:	Unsigned 32-bit interger
Access:	Read, write
Mechanism:	By reference

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 OMnet Gateway.

timeout

Type:	Unsigned 32-bit interger
Access:	Read only
Mechanism:	By value

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 0 and maximum value cannot be greater than 10 seconds (10000 milliseconds).

waittype

Type:	Unsigned 32-bit interger
Access:	Read only
Mechanism:	By value

This argument specifies the condition for the wait. Currently only the following 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 on page 63 under the description of READEV_OPTMSK_MANY event mask.

Return Values

OMNIAPI_SUCCESS	Successful completion; more events may exist.
OMNIAPI_ALLEVTS	All events collected.
OMNIAPI_NOTCONNECTED	Invalid operation before LOGIN.
OMNIAPI_NOT_LOGGED_IN	Connected to OMnet Gateway, but not logged in (LOGOUT forced from the central OMnet).
OMNIAPI_NOT_FOUND	Requested data were not found or there is no broadcast available. If there is no broadcast available, this is not an error.

OMNIAPI_OVERFLOW	Success, but at least one event message was lost.
OMNIAPI_INVEVT	An invalid (not enabled) event type was specified.
OMNIAPI_TRUNCATED	The provided buffer was too small, so the received message was truncated.
OMNAPI_NOT_AUTH	You are not allowed to read this event type.
OMNIAPI_FAILURE	Undefined error in the internal API.
OMNIAPI_FATAL	The OMnet Gateway encountered a fatal error in the internal OMnet API.
OMNIAPI_INTFAILURE	Internal error in OMNIAPI.
OMNIAPI_BADARGVAL	At least one of the arguments holds an erroneous value, such as an illegal pointer value. This code is returned if more than one txmsgs message was specified.
OMNIAPI_NOSESSION	The session is aborted because the network link to the OMnet Gateway was disconnected.
OMNIAPI_ERRSEND	Error appeared during a send operation.
OMNIAPI_SESINUSE	The API session is already in use.

Calling Example

```
/*****************************************************************************
 *                                                                           *
 * Function: PollBdx                                                         *
 *                                                                           *
 * Description:                                                              *
 * Poll for broadcasts. This function uses the omniapi_read_event_block     *
 * function. The concurrent broadcast feature must be enabled for this       *
 * call to succeed.                                                          *
 *                                                                           *
 * Input parameters:                                                         *
 * hSession : API Session handle                                             *
 *                                                                           *
 * Return value:                                                             *
 * : 0 - Failed, 1 - Success                                                 *
 *                                                                           *
 *****************************************************************************/
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;
}
```

6.4 Concurrent Broadcasts and Transactions

This section describes the process for enabling the concurrent broadcast feature for an API session and reading events concurrently in a separate thread.

By default, the OMNet API sessions cannot be used in multiple threads simultaneously without using synchronization mechanism by the applications. However, the functions for setting up subscriptions and reading broadcasts can be called in a different thread concurrently if the concurrent broadcast feature is

enabled on the session. This allows the multi-threaded applications to poll for broadcasts concurrently along with performing transactions and queries using the same session handle.

6.4.1 Enabling Concurrent Broadcast Feature

The application can enable the concurrent broadcast feature by calling the function **`omniapi_set_option_ex()`** function on a valid session handle before login, that is, before calling **`omniapi_login_ex()`**.

The API library will allow some calls to be called from a different thread concurrently. The following API calls cannot be called concurrently from the multiple threads:

- omniapi_logout_ex
- omniapi_set_newpwd_ex
- omniapi_tx_ex
- omniapi_query_ex
- omniapi_get_message_ex
- omniapi_set_option_ex
- omniapi_set_option_default
- omniapi_tx_response_ex
- omniapi_get_info_ex
- omniapi_login_ex

The following API calls will use the concurrent connection and can be called from a different thread concurrently:

- omniapi_set_event_ex
- omniapi_clear_event_ex
- omniapi_read_event_ext_ex
- omniapi_read_event_block

Besides these calls, the API library will ensure that no calls using the same connection can be made from two different threads at the same time. Doing so will result in one of the calls being rejected with the error 'OMNIAPI_SESINUSE.'

The following example shows how to enable concurrent broadcast feature for a session:

Example 10:

```
/*************************************************************************
 *                                                                       *
 * Function: EstablishConcurrent                                         *
 *                                                                       *
 * Description:                                                          *
 *    This function tries to establish a concurrent connection           *
 *    with the connection with thegateway.                               *
 *                                                                       *
 * Parameters:                                                           *
 *     hSession: OAPI Session handle                                     *
 *                                                                       *
```

```
* Return value: Failed        :   NULL                                    *
*                Success       :   Session Handle                          *
*                                                                          *
***************************************************************************/
int EstablishConcurrent(omniapi_session_handle hSession)
{
   int sts, txstatus;

   sts = omniapi_set_option_ex(hSession, OMNIAPI_OPT_CONCURRENT_BDX,
         OMNIAPI_OPVAL_ENABLE);
   if (sts != OMNIAPI_SUCCESS)
       printf("Error %d\n",sts);
   else
       printf("Success\n");

   return sts;
}
```

6.4.2 Handling Network Errors on Concurrent Connection

In the event of any network disruption between the client application and the gateway, the concurrent connection is handled in unison with the logon connection, i.e. either both are available or none. This implies a fail-fast model, where a disconnect on any of the two connections triggers a logout (and implicitly a disconnect of the remaining connection).

6.4.3 Behavior of Read Event and set Event Functions

The behavior of the broadcast related functions will change depending on the situation if the gateway does not support concurrent connections and situation where it supports concurrent connections. The following table lists the behavior of the read event and set event functions under various situations:

Functions	**Gateway not Support Concurrent BDX**	**GatewaySupports Concurrent BDX**		
		Feature Not Enabled	Feature Enabled	Error in Concurrent Connection
read_event_ext_ex	Not concurrent	Not concurrent	Concurrent	Fail
read_event_block	Fail	Fail	Concurrent	Fail
set_event_ex	Not concurrent	Not concurrent	Concurrent	Fail
clear_event_ex	Not concurrent	Not concurrent	Concurrent	Fail

7 Messages

7.1 Information Messages

Number: 00000005 (5) Name: OMNIAPI_TX_IN_PROGRESS

Explanation:	The last blocking transaction or query is still executing.
User Action:	None. Currently this message is handled by the API library itself and is not propagated to the application.ast request.

Number: 00000004 (4) Name: OMNIAPI_ALLEVTS

Message Text:	All events collected.
Explanation:	There are no more messages in the broadcast buffer. All events are collected.
User Action:	None.

Number: 00000003 (3) Name: OMNIAPI_OVERFLOW

Message Text:	Event message buffer overflow.
Explanation:	The API has successfully read a message from the broadcast message buffer. However, the API has also detected a loss of broadcast messages. The OMnet Gateway now has a filled broadcast message buffer and has discarded at least one broadcast message.
User Action:	Increase the **`omniapi_read_event_ex`** call frequency. If a loss of broadcast messages remains, contact Customer Support. If this status code is received from any other call than the **`omniapi_read_event`** call, report the problem to Customer Support.

Number: 00000002 (2) Name: OMNIAPI_ALR_CLR

Message Text:	Event type already cleared.
Explanation:	The requested event number is already cleared, either because it has never been set or because it has already been cleared by a previous **`omniapi_clear_event_ex`** call.
User Action:	Correct the program logic. If this message is returned from a call other than the **`omniapi_clear_event_ex`** call, it should be regarded as a back-end application message.

Number: 00000001 (1) Name: OMNIAPI_ALR_SET

Message Text:	Event type already set.
Explanation:	The API is trying to set an event number or subscribe for a broadcast twice.
User Action:	Correct the program logic. If this message is returned from a call other than the **`omniapi_set_event()`** call, it should be regarded as a back-end application message.

Number: 00000000 (0) Name: OMNIAPI_SUCCESS

Message Text:	Successful completion.
Explanation:	An OMNIAPI call has been completed successfully.
User Action:	If this value is returned from an OMNIAPI call, the call has been completed successfully.

7.2 Error Messages

There are two types of OMnet related error messages:

- Messages triggered by an error when you are not connected to OMN_TAP. These are described below.
- Messages triggered by an error when you are connected to OMN_TAP. These are described in the OMnet System Error Messages Reference manual.

Number: FFFFFFFF (-1) Name: OMNIAPI_FAILURE

Message Text:	Failure completion.
Explanation:	Internal error in the API.
User Action:	Contact the Customer Support.

Number: FFFFFFFE (-2) Name: OMNIAPI_NOT_READY

Message Text:	OAPI not ready.
Explanation:	The OMnet API is not ready for an OMNIAPI call.
User Action:	At least one of the tables defining the user environment in the OMnet Gateway has not been received after a successful LOGIN request. Wait a few seconds and try calling the OMNIAPI routine again. If the problem remains after 60 seconds, contact Customer Support.

Number: FFFFFFFD (-3) Name: OMNIAPI_FACID_NOT_VALID

Message Text:	Facility id out of range.
Explanation:	An invalid facility type, most likely zero, was used during an **`omniapi_tx_ex`** or **`omniapi_query`** call.
User Action:	Correct the program, using facility types defined in the **omnifact.h** file.

Number: FFFFFFFC (-4) Name: OMNIAPI_INVALID_TABLE

Message Text:	Invalid table type.
Explanation:	The OMnet transaction syntax and verification subsystem has received an invalid transaction syntax table.
User Action:	None. The OMnet API should never return this error code.

Number: FFFFFFFB (-5) Name: OMNIAPI_NOT_LOGGED_IN

Message Text:	OAPI not logged in.
Explanation:	The OMnet Gateway considers the user as logged out.
User Action:	None.

Number: FFFFFFF7 (-9) Name: OMNIAPI_NOT_INITIALIZED

Message Text:	OAPI not initialized.
Explanation:	Internal API error code.
User Action:	Contact Customer Support.

Number: FFFFFFF6 (-10) Name: OMNIAPI_NO_INFO_RCVD

Message Text:	No network information received.
Explanation:	Internal API error code.
User Action:	Contact Customer Support.

Number: FFFFFFF5 (-11) Name: OMNIAPI_NOBACKEND

Message Text:	Backend communication failure.
Explanation:	There are communication problems with the central system.
User Action:	Contact customer support.

Number: FFFFFFF4 (-12) Name: OMNIAPI_TX_ABORTED

Message Text:	Transaction aborted.
Explanation:	The transaction was aborted. The following (TXSTAT) code explains the cause.
User Action:	None.

Number: FFFFFFF3 (-13) Name: OMNIAPI_TRUNCATED

Message Text:	Data truncated.
Explanation:	A query message response message was truncated.
User Action:	Either provide a bigger response buffer *or* set the receive buffer (the RCVLEN argument) length correctly before each call to **`omniapi_query.`**

Number: FFFFFFF2 (-14) Name: OMNIAPI_CNV_NO_RANGE

Message Text:	Table conversion range error.

Explanation:	Internal error in the internal API.
User Action:	Contact Customer Support.

Number: FFFFFFF1 (-15) Name: OMNIAPI_CNV_NOT_SORTED

Message Text:	Table conversion sort error.
Explanation:	Internal error in the internal API.
User Action:	Contact Customer Support.

Number: FFFFFFF0 (-16) Name: OMNIAPI_CNV_OFFS_ERROR

Message Text:	Table conversion offset error.
Explanation:	Internal error in the internal API.
User Action:	Contact Customer Support.

Number: FFFFFFEF (-17) Name: OMNIAPI_NO_SUCH_ID

Message Text:	Invalid transaction type.
Explanation:	The transaction type of the transaction is incorrect or unauthorized.
User Action:	Check the transaction type. The first two bytes contain ISO Latin-1 (mostly ASCII) characters, whereas bytes 3 and 4 contain a little-endian two-byte integer. If the transaction type is authorized, check the byte order of the two-byte integer.

Number: FFFFFFEE (-18) Name: OMNIAPI_VER_FIELD_ERROR

Message Text:	Transaction verification error.
Explanation:	The transaction contents (after the transaction type) are incorrect or not recognized. This error message should only be returned in the development phase of an API client application.
User Action:	Check the contents of the transaction and the byte order of possible integers. If the problem remains, request assistance with the "Tracing Utility" from Customer Support.

Number: FFFFFFED (-19) Name: OMNIAPI_VER_INT_ERROR

Message Text:	Message verification internal error.
Explanation:	Internal error in the internal API.
User Action:	Contact Customer Support.

Number: FFFFFFEC (-20) Name: OMNIAPI_VER_TABLE_ERROR

Message Text:	Table verification error.
Explanation:	Internal error in the internal API.

User Action:	Contact Customer Support.

Number: FFFFFFEB (-21) Name: OMNIAPI_TX_TIMEOUT

Message Text:	Transaction timeout.
Explanation:	The transaction has timed out and is incomplete. This may have been caused by: • workload • problems at the marketplace • network problems
User Action:	Resubmit the transaction and contact Customer Support.

Number: FFFFFFEA (-22) Name: OMNIAPI_TX_DECLFAIL

Message Text:	Transaction requester declaration failure.
Explanation:	A connection to a particular facility failed.
User Action:	Contact customer support.

Number: FFFFFFE9 (-23) Name: OMNIAPI_TX_FAILURE

Message Text:	Transaction failure.
Explanation:	The transaction failed. The following (TXSTAT) code explains the cause.
User Action:	None.

Number: FFFFFFE8 (-24) Name: OMNIAPI_DYNMEM

Message Text:	Error obtaining dynamic memory.
Explanation:	Memory allocation error in the OMnet Gateway.
User Action:	Contact customer support.

Number: FFFFFFE7 (-25) Name: OMNIAPI_INVARG

Message Text:	Invalid argument.
Explanation:	An invalid facility type argument was specified to **`omniapi_tx_ex`** or **`omniapi_query`** routines.
User Action:	Check the FACTYP argument.

Number: FFFFFFE6 (-26) Name: OMNIAPI_NOT_FOUND

Message Text:	Requested data not found.

Explanation:	Buffer is empty. No more events or ticks have been received or returned from an **`omniapi_read_event_ex`** call. Message also occurs when **`omniapi_get_info`** is called with an invalid information request.
User Action:	None.

Number: FFFFFFE5 (-27) Name: OMNIAPI_ITV_ERROR

Message Text:	Information table verification error.
Explanation:	Internal error in the internal API.
User Action:	Contact Customer Support.

Number: FFFFFFE4 (-28) Name: OMNIAPI_NO_USR_OR_PASSW

Message Text:	Username and/or password missing.
Explanation:	The username or the password was not provided with the login request.
User Action:	Provide both a username and a password with the login request.

Number: FFFFFFE3 (-29) Name: OMNIAPI_NO_NET_PATH

Message Text:	Net path data missing.
Explanation:	The OMnet Gateway could not identify its network path.
User Action:	Check the DECnet configuration and the system parameters at the host of the OMnet Gateway.

Number: FFFFFFE2 (-30) Name: OMNIAPI_INVEVT

Message Text:	Invalid event type.
Explanation:	An invalid event type was specified in a call to **`omniapi_clear_event.`**
User Action:	Use granted event types. These event types are obtained by calling **`omniapi_read_event_ex`** with the event type set to OMNI_EVTTYP_SHOW.

Number: FFFFFFE0 (-32) Name: OMNIAPI_INVTXTYPE

Message Text:	Invalid transaction type.
Explanation:	A zero was passed as facility type to an **`omniapi_tx_ex`** or **`omniapi_query`** call.
User Action:	Use a business facility type or a session facility type (login, logout, setpassword etc).

Number: FFFFFFDD (-35) Name: OMNIAPI_FATAL

Message Text:	Fatal OAPI error.
Explanation:	Undefined internal error in the internal API.

User Action:	Log out and log in again to verify that the problem persists. If the problem persists, submit a problem report.

Number: FFFFFFDC (-36) Name: OMNIAPI_NOORDERID

Message Text:	No order id.
Explanation:	Obsolete error code.
User Action:	No order ID could not be generated for the transaction.

Number: FFFFFFD7 (-39) Name: OMNIAPI_NOTAUTH

Message Text:	The user is not authorized to perform the request action.
Explanation:	The user is not authorized to perform the request action.
User Action:	None.

Number: FFFFFFD7 (-41) Name: OMNIAPI_INVALID_PASSW

Message Text:	Password is not valid.
Explanation:	The provided new password is not valid. The new password breaches the password restrictions set by the exchange. The reason for this may be (a) that the password is shorter than x characters, where x is a number configurable by the exchange, or (b) that the password has already been used during the last y months, where y is a number configurable by the exchange.
User Action:	Prompt the user for a new valid password. Make a new attempt to change the password.

Number: FFFFFFCE (-50) Name: OMNIAPI_ENDIAN_ERROR

Message Text:	endian/byteswapping error
Explanation:	Error when byteswapping the transaction or query in the OMnet Gateway.
User Action:	Log out and log in again to verify that the problem persists. If the problem persists, submit a problem report.

Number: FFFFFFCB (-53) Name: OMNIAPI_NETERR_LINK

Message Text:	Link failure between Client and Gateway.
Explanation:	Network error occurred between client and gateway.
User Action:	The application should re-login to the gateway.

Number: FFFFFFCA (-54) Name: OMNIAPI_NETERR_TIMEOUT

Message Text:	Timed out during network I/O, connection with gateway is lost.
Explanation:	Timed out while waiting for results from the gateway.

User Action:	The application should re-login to the gateway.

Number: FFFFFFC9 (-55) Name: OMNIAPI_CONCUR_ERROR

Message Text:	Error in concurrent connection.
Explanation:	The API has lost concurrent connection with the gateway.
User Action:	The API client should re-establish concurrent connection with the gateway.

Number: FFFFFFC8 (-56) Name: OMNIAPI_CONCUR_DISABLE

Message Text:	Concurrent connection is disabled.
Explanation:	The concurrent connection is currently disabled.
User Action:	The application should enable concurrent connection.

Number: FFFFF82F (-2001) Name: OMNIAPI_INTFAILURE

Message Text:	Internal OMNIAPI error.
Explanation:	Internal OMnet API failure in the OMnet API library.
User Action:	Restart the application and verify that the problem persists. If the problem persists, submit a problem report.

Number: FFFFF82D (-2003) Name: OMNIAPI_BADNARGS

Message Text:	Bad number of arguments.
Explanation:	An incorrect number of arguments were passed to the OMNIAPI routine.
User Action:	Check the number of provided arguments.

Number: FFFFF82C (-2004) Name: OMNIAPI_BADARGVAL

Message Text:	Bad argument value.
Explanation:	At least one of the arguments contains an incorrect value.
User Action:	Debug the application. Specifically look for null pointers.

Number: FFFFF82B (-2005) Name: OMNIAPI_NONETWORK

Message Text:	No network is present for IPC.
Explanation:	The network software is unavailable for inter process communication.
User Action:	Check the network processes and configuration on the local node.

Number: FFFFF82A (-2006) Name: OMNIAPI_OSBADCONFIG

Message Text:	O/S incorrectly configured.
Explanation:	The operating system is incorrectly configured.
User Action:	Compare the system configuration to the network configuration.

Number: FFFFF829 (-2007) Name: OMNIAPI_OAPI

Message Text:	Problem in the internal OM API.
Explanation:	Obsolete error code.
User Action:	Contact Customer Support.

Number: FFFFF82F8 (-2008) Name: OMNIAPI_NOTCONNECTED

Message Text:	Invalid operation before LOGIN.
Explanation:	The OMnet API application has no network link to the OMnet Gateway.
User Action:	Check the application. No OMNIAPI calls are valid after a logout request or an aborted business transaction with a reason code between (decimal) -1000 and -1999.

Number: FFFFF82F7 (-2009) Name: OMNIAPI_NOGWYSRV

Message Text:	Unable to connect to gateway.
Explanation:	A connection to an OMnet Gateway failed. Some possible causes: • the host or socket name is incorrect • the OMnet Gateway is missing • a previous session with the OMnet Gateway was incorrectly terminated
User Action:	Check the TXSTAT code for more information about the cause. Also check the host and the socket names and the existence of an OMnet Gateway. Finally check the state of the OMnet Gateway using the OGWYLOOK utility.

Number: FFFFF826 (-2010) Name: OMNIAPI_BADHOSTNAME

Message Text:	Host name could not be translated.
Explanation:	An unknown or syntactically incorrect host name for the Network Gateway node was specified in a login request.
User Action:	Check the host name of the login request.

Number: FFFFF825 (-2011) Name: OMNIAPI_ERRSOCKET

Message Text	Socket could not be allocated.

Explanation:	Error assigning a local socket.
User Action:	Check local system configuration.

Number: FFFFF824 (-2012) Name: OMNIAPI_ERRCONNECT

Message Text:	System error occurred on "connect".
Explanation:	An unfamiliar error happened during a login request.
User Action:	Check local system configuration.

Number: FFFFF823 (-2013) Name: OMNIAPI_ERRBIND

Message Text:	System error occurred on "bind".
Explanation:	An unfamiliar error appeared during a socket bind operation.
User Action:	Check local system configuration.

Number: FFFFF822 (-2014) Name: OMNIAPI_NOSESSION

Message Text:	Session is aborted.
Explanation:	The network link to the OMnet Gateway was disconnected.
User Action:	Check the network and the OMnet Gateway.

Number: FFFFF821 (-2015) Name: OMNIAPI_ERRSEND

Message Text:	Error on send().
Explanation:	An error appeared during a send operation.
User Action:	Check the network and the local system configuration.

Number: FFFFF820 (-2016) Name: OMNIAPI_ERRMEM

Message Text:	Error on malloc().
Explanation:	Error allocating OMnet API dynamic memory.
User Action:	Check the local system.

Number: FFFFF81F (-2017) Name: OMNIAPI_APIOLD

Message Text:	OMnet API too old for the Gateway.
Explanation:	The OMnet API application is linked with an old unsupported version of the OMnet API library.
User Action:	Relink the application with the latest version of the OMnet library.

Number: FFFFF81D (-2019) Name: OMNIAPI_SESINUSE

Message Text:	Gateway API session already in use.
Explanation:	There is an ongoing API call for the session, or a previous session has been incorrectly terminated.
User Action:	Check status of different threads in the API client program.

Number: FFFFF81C (-2020) Name: OMNIAPI_ERROPTCMPZ

Message Text:	Compression option rejected.
Explanation:	The requested compression option was rejected by the gateway.
User Action:	Check compression options or contact Customer Support about OMnet Gateway configuration.

Number: FFFFF81B (-2021) Name: OMNIAPI_ERROPTENCRYPT

Message Text:	Encryption option rejected.
Explanation:	The requested encryption option was rejected by the Gateway or API library.
User Action:	Check encryption options; check whether the API library supports encryption (see Section 4.8 on page 26) or contact Customer Support about Gateway configuration.

Number: FFFFF81A (-2022) Name: OMNIAPI_ERR_SECURE

Message Text:	Failed to establish secure link.
Explanation:	A secure link could not be established between the API client application and the gateway.
User Action:	Check the transaction status for error code from SSL libraries.

Number: FFFFF819 (-2023) Name: OMNIAPI_LOGIN_DISABLED

Message Text:	System login state is disabled.
Explanation:	User login has been disabled in the central system.
User Action:	Wait until login is enabled.

Number: FFFFF818 (-2024) Name: OMNIAPI_INVALID_LOGIN_ATTEMPT

Message Text:	Invalid login attempt.
Explanation:	User setup related problem.
User Action:	Contact Customer Support.

Number: FFFFF817 (-2025) Name: OMNIAPI_OMNBE_BUSY

Message Text:	Please wait - OMnet backend is busy.

Explanation:	Central OMnet processes are busy and cannot handle user login requests at the moment.
User Action:	Wait for a short while and try to login again.

Number: FFFFF816 (-2026) Name: OMNIAPI_LOGIN_FAILED

Message Text:	Communications/database problems.
Explanation:	There may have been problems to communicate with central OMnet, or internal database problems caused the login to fail.
User Action:	Try to login again. If this fails, contact Customer Support.

Number: 7EB (2027) Name: OMNIAPI_PWD_CHANGE_REQ

Message Text:	Password has expired. Restricted access until password is changed.
Explanation:	The password has expired. The Application will only get a restricted access to the Genium INET system until the password has successfully been changed. See Section 5.2.5 on page 31 and Section 4.5 on page 21.
User Action:	Change the password.

Number: 7EC (2028) Name: OMNIAPI_PWD_IMPEND_EXP

Message Text:	Password expiration is impending.
Explanation:	The login attempt succeeded, but the password will soon expire.
User Action:	The user may choose to ignore the message. But the recommendation is to change the password before it expires.

Number: FFFFF813 (-2029) Name: OMNIAPI_INVALID_CHRS

Message Text:	The new password contains invalid characters.
Explanation:	Characters 0x01 to 0x19, '%' and '\' are not allowed in passwords.
User Action:	Choose another password.

Number: FFFFF812 (-2030) Name: OMNIAPI_NO_NULL_TERMINATION

Message Text:	The provided string is no null-terminated.
Explanation:	Provided string has to be null-terminated.
User Action:	Contact Customer Support for the Application. Application has to be re-programmed.

Number: FFFFF811 (-2031) Name: OMNIAPI_WRONG_TX_SIZE

Message Text:	The provided buffer size is not correct.
Explanation:	The buffer provided for the call is of the wrong size.

User Action:	Contact Customer Support for the Application. Application has to be re-programmed.

Number: FFFFF810 (-2032) Name: OMNIAPI_CALL_NOT_SUPPORTED

Message Text:	The call that was done is not supported by the exchange.
Explanation:	There are calls that are deprecated and the exchange may choose to configure the OMnet Gateway to block them and force the Applications to use the newer calls.
User Action:	Contact Customer Support.

Number: FFFFF80F (-2033) Name: OMNIAPI_INVALID_LOGIN_LOCKED

Message Text:	Login denied – user is locked.
Explanation:	There has been too many failed login attempts. The user account has been locked for login.
User Action:	Contact Customer Support to unlock the user ID.

Number: FFFFF80E (-2034) Name: OMNIAPI_INVALID_LOGIN_SUSPENDED

Message Text:	Login denied – user is suspended.
Explanation:	The user account has been suspended by the exchange.
User Action:	Contact Customer Support.

Number: FFFFF80D (-2035) Name: OMNIAPI_RELOGIN_NOT_ALLOWED

Message Text:	Login denied – re-login is not allowed.
Explanation:	There is an already on-going session with the user ID. The exchange have configured that re-login is not allowed.
User Action:	Either logout the already on-going session or make a new login attempt but with the “forced”-flag set.

Number: FFFFF80C (-2036) Name: OMNIAPI_NOT_APPLICABLE

Message Text:	The requested information is not applicable.
Explanation:	The requested information is not applicable to the user or the session.
User Action:	None.

Number: FFFFF80B (-2037) Name: OMNIAPI_INVALID_LOGIN_INACTIVE

Message Text:	Login denied – user has been inactivated.
Explanation:	The user account has been unused for a too long time. The user account has been inactivated.
User Action:	Contact Customer Support.

Number: FFFFF80A (-2038) Name: OMNIAPI_INVALID_BEF_CHG_PWD

Message Text:	Invalid request before password is changed.
Explanation:	The user has password that has expired. The user will have restricted access to the Genium INET system until the password has been changed successfully.
User Action:	Change the password.

Number: FFFFF805 (-2043) Name: OMNIAPI_ERR_PARTITION

Message Text:	No partition information available for the transaction or query
Explanation:	The backend partition was not found.
User Action:	Contact customer support.

Number: FFFFF804 (-2044) Name: OMNIAPI_INV_PARTITION

Message Text:	Partition not allowed on the specified transaction or query.
Explanation:	The backend partition was not reachable for the given transaction.
User Action:	Try again and if the problem persist for an extended period of time contact customer support.

Number: FFFFF803 (-2045) Name: OMNIAPI_CONCUR_EXISTS

Message Text:	Concurrent connection already exists for the session.
Explanation:	A concurrent connection is already established for the session.
User Action:	Only one concurrent connection per session is allowed. Do not enable concurrent broadcast feature again if it is already enabled.

Number: FFFFF802 (-2046) Name: OMNIAPI_INVALID_IDENT

Message Text:	Invalid concurrent identifier, session may be lost.
Explanation:	Invalid session was encountered by the gateway when establishing a concurrent connection.
User Action:	The parent session may have been lost of logged out while the concurrent connection was commencing.

A Appendix A – OMnet API Implementation Notes

A.1 General Notice about the Kits

The kits delivered, contains a directory tree structure with files. The following structure applies:

```
omex   ; OMex related files
gen   ; files belonging to the GEN subsys
src   ; required source files:
   ; omex.h placeholder, om_inttypes.h, omn_om_inttypes.h
omn   ; OMnet files
bin   ; object modules and/or executables
com   ; makefile for the apitest program
src   ; OMnet kit source files: apisample.c,
apisamples ; API Sample programs in C
```

A README file in the API-kit explains how to run the sample application.

B Appendix B – OpenSSL Open Source License

```
OpenSSL License
---------------
/* ====================================================================
 * Copyright (c) 1998-2002 The OpenSSL Project. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in
 * the documentation and/or other materials provided with the
 * distribution.
 * 3. All advertising materials mentioning features or use of this
 * software must display the following acknowledgment:
 * "This product includes software developed by the OpenSSL Project
 * for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 * endorse or promote products derived from this software without
 * prior written permission. For written permission, please contact
 * openssl-core@openssl.org.
 * 5. Products derived from this software may not be called "OpenSSL"
 * nor may "OpenSSL" appear in their names without prior written
 * permission of the OpenSSL Project.
 * 6. Redistributions of any form whatsoever must retain the following
 * acknowledgment:
 * "This product includes software developed by the OpenSSL Project
 * for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ‘‘AS IS’’ AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com). This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *<
 */ Original SSLeay License
```

```
-----------------------
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to. The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code. The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.
* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.
** Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in the
* documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
* must display the following acknowledgement:
* "This product includes cryptographic software written by
* Eric Young (eay@cryptsoft.com)"
* The word 'cryptographic' can be left out if the rouines from the library
* being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof) from
* the apps directory (application code) you must include an acknowledgement:
* "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ''AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed. i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/
```