exgarde SDK API

User Manual

UM0100_3

Unit 10 Concept Park
Innovation Close
Poole
Dorset
BH12 4QT

t: +44 (0) 1202 723535
w: www.tdsi.co.uk
e: sales@tdsi.co.uk
e: support@tdsi.co.uk

Foreword

Time and Data Systems International Ltd operate a policy of continuous improvement and reserves the right to change specifications, colours or prices of any of its products without prior notice.

Guarantee

For terms of guarantee, please contact your supplier.

Trademarks

Microsoft and Windows are registered trademarks of Microsoft Corporation.
All other brands and product names are trademarks or registered trademarks of their respective owners.

Cautions and Notes

The following symbols are used in this guide:

CAUTION! This indicates an important operating instruction that should be followed to avoid any potential damage to hardware or property, loss of data, or personal injury.

NOTE. This indicates important information to help you make the best use of this product.

Document Control

Issue	Date Issued	Change Summary	Issued by
1	28/11/16	Initial Release	GFH
2	04/12/16	Added sections 4.9 and 5.9. Added class info	RT
3	14/02/17	Added Door state info and Enumerations	RT

Contents

1 Introduction

1.1 Instructions

This SDK is designed to enable you to interface to EXgarde PRO to obtain certain parameters and control certain aspects of the Access Control System. Some of the commands utilise Fusion to run and are defined within this SDK to provide a common interface. These commands are identified and will require Fusion to be running for them to operate.

There are two libraries provided with this SDK, for server and client functions. The Server library is used if an application has been developed to run on the machine where the SDK has been installed. The client library is to be use if the application is to be run on a remote machine with a network connection to the machine the SDK has been installed on.

EXgarde SDK consists of the following:

- Library
- Configuration file

1.2 System Requirements

The following are the system requirements for the EXgarde SDK

- Operating System – Windows 7 Professional or higher
- Microsoft .Net 4.5.2 or higher
- Development environment – Microsoft Visual Studio 2012 or higher
- IIS 7 or higher enabled in windows

2 Enumerations

```
enum KeyStatus
{ Available, Issued,Lost,Damaged,Suspended,ReIssued };

enum DoorState
{ Released, Locked, LockedBySchedule, Unlocked,
UnlockBySchedule, ControlledAccess };

enum DoorStatus
{ Unknown, DoorOpened, DoorClosed, DoorForced,
DoorLocalAlarm, DoorRemotealarm, DoorEgressOn,
DoorEgressOff };


enum DoorSenseState
{ Open, Close };

enum EgressState
{ On, Off };

enum ACUState
{ online, offline };
```

3 Classes

General Info

```
public class Info
{
    public string Manufacturer; // TDSi
    public string Software;     // EXgarde PRO
    public string EXgardeVer;
    public string DBVer;
 };
```

System Info

```
public class SystemInfo
{
    public int NumberACUs;
    public int NumberDoors;
    public int NumberKeyolders;
};
```

ACU Info

```
public class ACUInfo
{
    public int ID;
    public string ACUName;
    public string Model;
    public string Configuration;
    public string UID;
    public string PortType;
    public int ReaderCount;
```

```
    public ReaderInfo[] reader;
    public DoorInfo[] doorInfo;
    public string FirmwareVer;
    public int CommStatus;
 };
```

Door Info

```
public class DoorInfo
{
    public int ID;
    public string DoorName;
    public int ACUID;
    public int DoorNumber;
    public int EntryReaderID;
    public int ExitReaderID;
    public string AreaToName;
    public string AreaFromName;
    public int DoorSense;
    public int EgressState;
    public DoorState DoorProgrammedState;
    public DoorStatus DoorState;
  };
```

Alarm Procedure

```
public class AlarmProcedure
{
    public int ID;
    public string Name;
    public string Comment;
    public int Priority;
    public string Actions;
 };
```

Alarm Info

```
public class AlarmInfo
{
    public int ID;
    public int ProcedureID;
    public string Name;
    public string Operator;
    public int Priority;
    public DateTime Created;
    public DateTime Updated;
    public int AckStatus;
    public int ACUID;
    public int DoorID;
    public int ReaderID;
 };
```

Event

```
public class Event
```

```
{
    public int ID;
    public DateTime SystemTime;
    public DateTime LocalTime;
    public long EventID;
    public string EventType;
    public string Message;
    public int AlarmID;
    public string AlarmName;
    public long ID1;
    public long ID2;
    public long ID3;
    public long ID4;
    public string AlarmProcedureID;
    public string Text;
 };
```

Keyholder Info

```
public class KeyholderInfo
{
    public int ID;
    public string Name;
    public string LongName;
    public string Comment;
    public string Tenant;
    public string Info1;
    public string Info2;
    public string Info3;
    public string Info4;
    public string Info5;
    public string Info6;
    public string Info7;
    public string Info8;
 };
```

Reader Info

```
public class ReaderInfo
{
    public int ID;
    public string Name;
    public string LongName;
    public int ACUID;
    public int ReaderNumber;
    public string ReaderType;
 };
```

Input Info

```
public class InputInfo
{
    public int ID;
    public string Name;
    public int ACUID;
```

```
};
```

Areas

```
public class Areas
{
    public int ID;
    public string Name;
    public string ParentArea;
 };
```

Groups

```
public class Groups
{
    public int ID;
    public string Name;
 };
```

Keys

```
public class Keys
{
    public int ID;
    public string Type;
    public int Number;
    public int Status;
    public string Keyholder;
    public DateTime ValidityStart;
    public DateTime ValidityEnd;
 };
```

Key Boxes

```
public class KeyBox
{
    public int ID;
    public string Name;
 };
```

4 Server Functions

4.1 Constructors

public **EXGardeServer**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Creates a new instance of the EXgarde SDK

public **EXGardeServer**(**EXgarde.TCM.ITCMClient** *TcmClient*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Creates a new instance of the EXgarde SDK

Parameters:
TcmClient: The TCM client library use to connect to the EXgarde TCM

public **EXGardeServer**(**string** *ConfigurationPath*, **EXgarde.TCM.ITCMClient** *TcmClient*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Creates a new instance of the EXgarde SDK

Parameters:
ConfigurationPath: The path in which the configuration files reside. If not specified the current folder is used.
TcmClient: The TCM client library use to connect to the EXgarde TCM

public **EXGardeServer**(**string** *ConfigurationPath*, **EXgarde.TCM.ITCMClient** *TcmClient*, **EXgarde.Data.IRepository** *Repository*, [**EXgarde.Data.IIntrudersRepository** *IntruderRepository* = null])

Member of **EXgarde.Server.EXGardeServer**

Summary:
Creates a new instance of the EXgarde SDK

Parameters:
ConfigurationPath: The path in which the configuration files reside. If not specified the current folder is used.
TcmClient: The TCM client library use to connect to the EXgarde TCM
Repository: The database repository to use
IntruderRepository: The intruder repository to use

4.2 Connection Functions

public **EXgarde.ConnectionStatus** **Connect**(**string** *DatabaseUserName*, **string** *DatabasePassword*,**string** *OperatorName*)
Member of **EXgarde.Server**.**EXGardeServer**

Summary:
Connects to EXgarde

Parameters:
DatabaseUserName: EXgarde database user name
DatabasePassword: EXgarde database user password
OperatorName: EXgarde operator name

Returns:
The status of the connection. One of the EXgarde.ConnectionStatus values

public **EXgarde.ConnectionStatus** **ConnectServer**(**string** *DatabaseUserName*, **string** *DatabasePassword*, **string** *OperatorName*)
Member of **EXgarde.Server**.**EXGardeServer**
Summary:
Connects to EXgarde from a server instance

Parameters:
DatabaseUserName: EXgarde database user name
DatabasePassword: EXgarde database user password
OperatorName: EXgarde operator name

Returns:
The status of the connection. One of the EXgarde.ConnectionStatus values

Remarks:
ConnectServer is intended for a permanent server connection, such as that initiated by a server. It creates a connection but also starts server based processes such as Active Directory and Intruder monitoring if those are licensed options. If ServerConnect is issued more than once and Active Directory is enabled you could see Active Directory events being raised twice. Similarly, you would experience problems with intruder systems as ConnectServer connects to intruder systems, some of which may not allow multiple connections.

public **void Disconnect**()

Member of **EXgarde.Server**.**EXGardeServer**

Summary:
Disconnects from EXgarde

4.3 Dispose Function

public **void Dispose**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Disposes of the EXgarde server instance

4.4 Keyholder Functions

public **int CreateKeyholder**(**string** *Name*, **string** *LongName*, **string** *Comment*, **System.Collections.Generic.List<string>** *Infos*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Create a new keyholder

Parameters:
Name: Keyholder name
LongName: Keyholder longname
Comment: Keyholder comment
Infos: List of strings for the keyholder info

Returns:
Fusion response from Create keyholder

public **System.Collections.Generic.IEnumerable<EXgarde.Data.KeyholderInfo> GetKeyholders**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Returns information on the Keyholders in the system

Returns:
An enumerable list of Keyholders Exgarde.Data.KeyholderInfo

public **System.Collections.Generic.IEnumerable<EXgarde.Data.KeyholderInfo> GetKeyholdersInArea**(**int** *AreaID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get a list of keyholders within a specific area

Parameters:
AreaID: The ID of the area to check

Returns:
An enumerable list of Keyholders Exgarde.Data.KeyholderInfo

public **EXgarde.Data.KeyholderInfo** **GetKeyholder**(**int** *KeyholderID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get information for a specific keyholder

Parameters:
KeyholderID:

Returns:
Keyholder info Exgarde.Data.KeyholderInfo

public **byte[]** **GetKeyholderInfoData**(**int** *KeyholderID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get the keyholder's info data

Parameters:
KeyholderID: ID of the keyholder to get

Returns:
byte array of the info data

public **System.Drawing.Image** **GetKeyholderImage**(**int** *KeyholderID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get the image for a particular keyholder

Parameters:
KeyholderID:

Returns:
An image

public **int** **ModifyKeyholder**(**string** *Name*, **string** *NewName*, **string** *LongName*, **string** *Comment*, **System.Collections.Generic.List<string>** *Infos*, **string** *PicturePath*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Modify a keyholder

Parameters:
Name: Keyholder existing name
NewName: Keyholder new name
LongName: Keyholder longname
Comment: Keyholder comment
Infos: List of strings for keyholder info
PicturePath: path to the keyholder picture

Returns:
Fusion response from Modifykeyholder

public **int** **DeleteKeyholder**(**string** *Name*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Delete a keyholder

Parameters:
Name: Keyholder name to delete

Returns:
Fusion response from Delete Keyholder

public **int** **AddKeyholderToGroup**(**string** *KeyholderName*, **string** *GroupName*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Add keyholder to group

Parameters:
KeyholderName: Keyholder name to add
GroupName: Group name to add to

Returns:
Response from fusion command

public **int** **RemoveKeyholderFromGroup**(**string** *KeyholderName*, **string** *GroupName*)

Member of **EXgarde.Server.EXGardeServer**
Summary:
Remove keyholder from group

Parameters:
KeyholderName: Name of keyholder to remove
GroupName: Group to remove keyholder from

Returns:
Response from fusion command

public **System.Collections.Generic.IEnumerable<EXgarde.Data.Group>** **GetGroups**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get a list of groups in the system

Returns:
Enumerable list of groups, Exgarde.Data.Group

public **System.Collections.Generic.IEnumerable<EXgarde.Data.Group>** **GetGroups**(**int** *KeyholderID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get groups a keyholder belongs to

Parameters:
KeyholderID: ID of keyholder to check

Returns:
Enumerable list of groups, Exgarde.Data.Group

public **EXgarde.Data.Group** **GetGroup**(**int** *GroupID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get information for a group

Parameters:
GroupID: ID of group to interrogate

Returns:
Group information, Exgarde.Data.Group

public **bool** **IsKeyholderInGroup**(**int** *KeyholderID*, **int** *GroupID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
See if a keyholder is in a particular group

Parameters:
KeyholderID: ID of keyholder to check
GroupID: ID of group to check

Returns:
True if keyholder in group

public System.Collections.Generic.IEnumerable<EXgarde.Data.Group> **GetIntruderGroups**()

Member of EXgarde.Server.EXGardeServer

Summary:
Get a list of intruder groups in the system

Returns:
Enumerable list of groups, Exgarde.Data.Group

4.5 Key Functions

public int **CreateKeyBox**(string *Name*)

Member of EXgarde.Server.EXGardeServer

Summary:
Create a keybox

Parameters:
Name: Name of the keybox

Returns:
Response from fusion command

public System.Collections.Generic.IEnumerable<EXgarde.Data.KeyBox> **GetKeyBoxes**()

Member of EXgarde.Server.EXGardeServer

Summary:
Get a list of Keyboxes in the system

Returns:
Enumerable list of key boxes Exgarde.Data.Keybox

public EXgarde.Data.KeyBox **GetKeyBox**(int *BoxID*)

Member of EXgarde.Server.EXGardeServer

Summary:
Get information for a specific keybox

Parameters:
BoxID: Keybox ID

Returns:
Keybox information Exgarde.Data.KeyBox

public int **ModifyKeyBox**(string *Name*, string *NewName*)

Member of EXgarde.Server.EXGardeServer

Summary:

Modify a keybox

Parameters:
Name: Current name of the keybox
NewName: New name of the keybox

Returns:
Response from fusion command

public **int** **DeleteKeyBox**(**string** *Name*)

Member of **EXgarde.Server**.**EXGardeServer**

Summary:
Delete a keybox

Parameters:
Name: Keybox to delete

Returns:
Response from fusion command

public **int** **CreateKey**(**int** *Number*, **string** *KeyholderName*, **string** *BoxName*, **EXgarde.KeyType** *Technology*)

Member of **EXgarde.Server**.**EXGardeServer**

Summary:
Create a key

Parameters:
Number: Key number
KeyholderName: Keyholder name to associated key with
BoxName: Name of keybox to put key in
Technology: Technology EXgarde.KeyType

Returns:
Response from Fusion command

public **System.Collections.Generic.IEnumerable<EXgarde.Data.Key>** **GetKeys**()

Member of **EXgarde.Server**.**EXGardeServer**

Summary:
Gets a list of keys within the system

Returns:
An enumerable list of keys Exgarde.Data.Key

public **EXgarde.Data.Key** **GetKey**(**int** *KeyID*)

Member of **EXgarde.Server**.**EXGardeServer**

Summary:
Get information on a specific key

Parameters:
KeyID: ID of key to get

Returns:
Key information. Exgarde.Data.Key

public **int IssueKey**(**int** *Number*, **EXgarde.KeyType** *Technology*, **string** *KeyholderName*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Issue key

Parameters:
Number: Key number to issue
Technology: Key technology EXgarde.KeyType
KeyholderName: Keyholder name to issue key to

Returns:
Response from Fusion command

public **void ReissueKey**(**int** *KeyID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Re-issue a key

Parameters:
KeyID: Key to re-issue

public **void KeyAvailable**(**int** *KeyID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Change a key to be available

Parameters:
KeyID: key to change

public **void KeyDamaged**(**int** *KeyID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Change a key to damaged

Parameters:
KeyID: The key that has been damaged

public **void KeyLost**(**int** *KeyID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Change a key to be lost

Parameters:
KeyID: The key that has been lost

public **void SuspendKey**(**int** *KeyID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Suspend a key

Parameters:
KeyID: The key to suspend

public **int ModifyKey**(**int** *Number*, **string** *KeyholderName*, **string** *BoxName*, **EXgarde.KeyType** *Technology*, **EXgarde.KeyStatus?** *Status*, **string** *Pin*, **System.DateTime?** *StartDate*, **System.DateTime?** *EndDate*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Modify a key

Parameters:
Number: Key number
KeyholderName: Keyholder name
BoxName: Keybox name
Technology: Key technology EXgarde.KeyType
Status: Key Status EXgarde.KeyStatus
Pin: PIN
StartDate: Valid start date
EndDate: Valid end date

Returns:
Response from fusion command

public **int DeleteKey**(**int** *Number*, **EXgarde.KeyType** *Technology*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Delete a key

Parameters:
Number: key number to delete
Technology: Key technology EXgarde.KeyType

Returns:
Response from Fusion command

4.6 ACU Functions

public **System.Collections.Generic.IEnumerable<EXgarde.Data.ACUInfo>** **GetACUs**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Requests a list of ACU details

Returns:
A enumerable list of ACUs

public **EXgarde.Data.ACUInfo** **GetACU**(**int** *ACUID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Requests a single ACU

Parameters:
ACUID: The ID of the ACU to fetch

Returns:
The requested ACU

public **EXgarde.ACUState** **GetACUCommsState**(**int** *ACUID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Requests the status of an ACU

Parameters:
ACUID: The ID of the ACU

Returns:
The ACU state

4.7 Door Functions

public **System.Collections.Generic.IEnumerable<EXgarde.Data.DoorInfo>**
GetDoors(**EXgarde.DoorFetch** *Fetch*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get information on the doors in the system

Parameters:
Fetch: Can be one of DoorFetch.IncludeReader to also fetch the reader details associated with the door or DoorFetch.ExcludeReader to only fetch the reader and area details

Returns:
An enumerable list of Doors

public **EXgarde.Data.DoorInfo** **GetDoor**(**int** *DoorID*, **EXgarde.DoorFetch** *Fetch*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get information on a specific door

Parameters:
DoorID:
Fetch: Can be one of DooFetch.IncludeReader to also fetch the reader details associated with the door or DoorFetch.ExcludeReader to only fetch the reader and area details

Returns:
Door info

public **void AuthoriseAccess**(**int** *DoorID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Authorise access through a door. This unlocks the door remotely

Parameters:
DoorID: ID of door to unlock

public **void SetDoorState**(**int** *DoorID*, **EXgarde.DoorState** *NewState*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Set the state of a door, i.e. lock and unlock a door

Parameters:
DoorID:
NewState:

public System.Collections.Generic.IEnumerable<EXgarde.Data.DoorInfo> **GetFireDoors**()

Member of EXgarde.Server.EXGardeServer

Summary:
Get a list of Fire doors in the system

Returns:
An enumerable list of Fire Doors

public **void ResetFireDoors**()

Member of EXgarde.Server.EXGardeServer

Summary:
Reset Fire doors

4.8 Reader Functions

public System.Collections.Generic.IEnumerable<EXgarde.Data.ReaderInfo> **GetReaders**()

Member of EXgarde.Server.EXGardeServer

Summary:
Obtain a list of all readers in the system

Returns:
Enumerable list of ReadersEXgarde.Data.ReaderInfo

public EXgarde.Data.ReaderInfo **GetReader**(int *ReaderID*)

Member of EXgarde.Server.EXGardeServer

Summary:
Obtain information on a single reader in the system

Parameters:
ReaderID:

Returns:
ReaderInfo

4.9 Input Functions

public System.Collections.Generic.IEnumerable<EXgarde.Data.InputInfo> **GetInputs**()

Member of EXgarde.Server.EXGardeServer

Summary:

Requests a list of inputs

Returns:

The list of inputs

public **EXgarde.Input.InputInfo** **GetInput**(**int** *InputID)*

Member of EXgarde.Server.EXGardeServer

Summary:
Get info in a single input

Parameters:
InputID;

Returns:
Information on a particular input

4.10 Relay Functions

public System.Collections.Generic.List<EXgarde.Data.Relay> **GetRelays**()

Member of EXgarde.Server.EXGardeServer

Summary:
Requests a list of all relays

Returns:

public EXgarde.Data.Relay **GetRelay**(int *RelayID*)

Member of EXgarde.Server.EXGardeServer

Summary:
Requests a single relay

Parameters:
RelayID: The ID of the relay to fetch

Returns:

public **void SetRelayStatus**(**int** *RelayID*, **EXgarde.RelayStatus** *status*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Sets the state of a relay

Parameters:
RelayID: The ID of the relay
status: The new state of the relay. Will be one of the EXgarde.RelayStatus enumeration values

4.11 Area Functions

public **System.Collections.Generic.IEnumerable<EXgarde.Data.Area>** **GetAreas**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get a list of area information

Returns:
Enumerable list of area information. Exgarde.Data.Area

public **EXgarde.Data.Area** **GetArea**(**int** *AreaID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get information for an Area

Parameters:
AreaID: ID of area

Returns:
Area info, Exgarde.Data.Area

public **int** **GetAreaCount**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get a count of all areas in the system

Returns:
Count of areas

4.12 Occupancy Functions

public **void ResetOccupancyAll**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Reset all occupancy counts

public **void ResetOccupancyForArea**(**int** *AreaID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Reset the occupancy counter for an area

Parameters:
AreaID: ID of area to reset

public **void ResetOccupancyForKeyholder**(**int** *KeyholderID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Reset the occupancy count for a keyholder

Parameters:
KeyholderID: ID of keyholder to reset

4.13 Event Functions

public **System.Collections.Generic.IEnumerable<EXgarde.Data.Event>** **GetEvents**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get all events logged within the system

Returns:
An enumerable list of events, Exgarde.Data.Event

public **System.Collections.Generic.IEnumerable<EXgarde.Data.Event>** **GetEvents**(**EXgarde.EventNumber** *EventType*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get events by event type

Parameters:
EventType: Event type, EXgarde.EventNumber

Returns:
Enumerable list of events, Exgarde.Data.Event

public **System.Collections.Generic.IEnumerable<EXgarde.Data.Event>**
GetEvents(**EXgarde.EventNumber** *EventNumber*, **System.DateTime** *From*, **System.DateTime** *To*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get events by event type and within a time frame

Parameters:
EventNumber: Event type, EXgarde.EventNumber
From: Start date
To: End date

Returns:
Enumerable list of events, Exgarde.Data.Event

public System.Collections.Generic.IEnumerable<EXgarde.Data.Event> **GetEvents**(System.DateTime *From*, System.DateTime *To*)

Member of EXgarde.Server.EXGardeServer

Summary:
Get events within a time frame

Parameters:
From: Start date
To: End date

Returns:
Enumerable list of events, Exgarde.Data.Event

public EXgarde.Data.Event **GetEvent**(int *EventID*)

Member of EXgarde.Server.EXGardeServer

Summary:
Get event by ID

Parameters:
EventID: Event ID to get

Returns:
Event info, Exgarde.Data.Event

public **void InsertEvent**(EXgarde.EventNumber *Event*, int? *ID1*, int? *ID2*, int? *ID3*, int? *ID4*, int? *ID5*, int? *AlarmProcedureID*, string *Text*, System.DateTime *LocalTime*)

Member of EXgarde.Server.EXGardeServer

Summary:
Insert an event into EXgarde

Parameters:
Event: Event to insert, EXgarde.EventNumber
ID1: ID1
ID2: ID2
ID3: ID3
ID4: ID4
ID5: ID5
AlarmProcedureID: Alarm procedure ID
Text: Event text
LocalTime: local time for the event

public **void InsertEvent**(**EXgarde.EventNumber** *Event*, **int?** *ID1*, **int?** *ID2*, **int?** *ID3*, **int?** *ID4*, **int?** *AlarmProcedureID*, **string** *Text*, **System.DateTime** *LocalTime*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Insert an event into EXgarde

Parameters:
Event: Event to insert, EXgarde.EventNumber
ID1: ID1
ID2: ID2
ID3: ID3
ID4: ID4
AlarmProcedureID: Alarm procedure ID
Text: Event text
LocalTime: local time for the event

public **void InsertEvent**(**int** *EventNumber*, **int?** *ID1*, **int?** *ID2*, **int?** *ID3*, **int?** *ID4*, **int?** *ID5*, **int?** *AlarmProcedureID*, **string** *Text*, **System.DateTime** *LocalTime*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Insert an event into EXgarde

Parameters:
EventNumber: Event to insert
ID1: ID1
ID2: ID2
ID3: ID3
ID4: ID4
ID5: ID5
AlarmProcedureID: Alarm procedure ID
Text: Event text
LocalTime: local time for the event

public **void InsertEvent**(**int** *EventNumber*, **int?** *ID1*, **int?** *ID2*, **int?** *ID3*, **int?** *ID4*, **int?** *AlarmProcedureID*, **string** *Text*, **System.DateTime** *LocalTime*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Insert an event into EXgarde

Parameters:
EventNumber: Event to insert
ID1: ID1
ID2: ID2
ID3: ID3
ID4: ID4
AlarmProcedureID: Alarm procedure ID
Text: Event text
LocalTime: local time for the event

public **void PauseEvents**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Pauses listening for events

public **void ResumeEvents**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Resumes listening for events

public **void ProcessEvent**(**int** *siteID*, **int** *eventDetailsID*, **System.DateTime** *eventTime*, **int** *sourceObject*, **int** *ID1*, **int** *ID2*, **int** *ID3*, **int** *ID4*, **int** *ID5*, **string** *eventText*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Process an event and then raise TCM events based upon the response

Parameters:
siteID: siteID - should always be set to 1 at present
eventDetailsID: eventDetailsID
eventTime: event Date and Time
sourceObject: source object raising the event
ID1: ID1
ID2: ID2
ID3: ID3
ID4: ID4
ID5: ID5
eventText: event text to insert

public **EXgarde.Data.AuthorisationEvent** **GetAuthorisationEvent**(**int** *UniqueID*)
Member of **EXgarde.Server.EXGardeServer**

Summary:
Get details of the authorisation event

Parameters:
UniqueID: ID of the event to get

Returns:
Authorisation Event, Exgarde.Data.AuthorisationEvent

public **EXgarde.Data.AuthorisationEvent** **GetAuthorisationEventDetails**(**int** *UniqueID*, **int** *MultiCardPrimedID*)
Member of **EXgarde.Server**.**EXGardeServer**

Summary:
Get details of the authorisation event

Parameters:
UniqueID: Unique event ID
MultiCardPrimedID: Keyholder ID of keyholder who primed the system

Returns:
Authorisation Event, Exgarde.Data.AuthorisationEvent

4.14 Alarm Functions

public **System.Collections.Generic.IEnumerable<EXgarde.Data.AlarmInfo>** **GetAlarms**()

Member of **EXgarde.Server**.**EXGardeServer**

Summary:
Get Alarm information

Returns:
Enumerable list of Alarms, Exgarde.Data.AlarmInfo

public **System.Collections.Generic.IEnumerable<EXgarde.Data.AlarmInfo>** **GetAlarms**(**bool** *Acknowledged*)

Member of **EXgarde.Server**.**EXGardeServer**

Summary:
Get all alarms in the system

Parameters:
Acknowledged: True - Get acknowledged alarms, False - Non acknowledged alarms

Returns:
Enumerable list of Alarms, Exgarde.Data.AlarmInfo

public System.Collections.Generic.IEnumerable<EXgarde.Data.AlarmInfo> **GetAlarms**(System.DateTime *From*, System.DateTime *To*)

Member of EXgarde.Server.EXGardeServer

Summary:
Get alarms within a date range

Parameters:
From: Start date
To: End date

Returns:
Enumerable list of Alarms, Exgarde.Data.AlarmInfo

public System.Collections.Generic.IEnumerable<EXgarde.Data.AlarmInfo> **GetAlarms**(System.DateTime *From*, System.DateTime *To*, bool *Acknowledged*)

Member of EXgarde.Server.EXGardeServer

Summary:
Get alarms within a date range with acknowledge option

Parameters:
From: Start date
To: End date
Acknowledged: True - Get acknowledged alarms, False - Non acknowledged alarms

Returns:
Enumerable list of Alarms, Exgarde.Data.AlarmInfo

public System.Collections.Generic.IEnumerable<EXgarde.Data.AlarmProcedure> **GetAlarmProcedures**()

Member of EXgarde.Server.EXGardeServer

Summary:
Get alarm procedures

Returns:
Enumerable list of alarm procedures, Exgarde.Data.AlarmProcedure

public EXgarde.Data.AlarmProcedure **GetAlarmProcedure**(int *AlarmProcdureID*)

Member of EXgarde.Server.EXGardeServer

Summary:
Get alarm procedure

Parameters:
AlarmProcdureID: ID of Alarm procedure

Returns:
Alarm Procedure, Exgarde.Data.AlarmProcedure

public **int** **GetAlarmCount**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get the number of alarms present

Returns:
Number of alarms

public **EXgarde.Data.AlarmInfo** **GetAlarm**(**int** *AlarmID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get alarm information

Parameters:
AlarmID: ID of alarm

Returns:
Alarm information, Exgarde.Data.AlarmInfo

public **EXgarde.Data.AlarmEvent** **GetAlarmEvent**(**int** *AlarmID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get event associated with alarm

Parameters:
AlarmID: ID of alarm

Returns:
Alarm event, Exgarde.Data.AlarmEvent

public **EXgarde.AlarmAcknowledgementStatus** **AcknowledgeAlarm**(**int** *AlarmID*, **int** *OperatorID*, **EXgarde.AlarmClear** *Clear*, **string** *DeletionText*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Acknowledge an alarm that is present

Parameters:
AlarmID: ID of alarm to acknowledge
OperatorID: ID of the operator to acknowledge the alarm
Clear: Indicates whether or not the alarm should also be cleared if no alarm procedure is associated with the alarm. This only comes into effect if the alarm procedure is set to NONE (Procedure ID = 1). The values can be ClearIfNoProcedure to clear the alarm or DoNotClear to leave the alarm uncleared. Note: if there is an alarm procedure with this alarm then setting Clear to ClearIfNoProcedure has no effect. EXGarde.AlarmClear

DeletionText: The comment to use if the alarm is also to be cleared. This will only be used if the alarm procedure is also being cleared. The deletion text must be between 5 and 20 characters, otherwise an ArgumentException will be thrown.

Returns:
Acknowledgement status, Exgarde.AlarmAcknowledgementStatus

public **EXgarde.AlarmControlStatus TakeAlarmControl**(**int** *AlarmID*, **int** *OperatorID*, **bool** *Force*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Takes control of an alarm

Parameters:
AlarmID: ID of the alarm to take control of
OperatorID: ID of operator taking control of the alarm
Force: Indicates whether or not the alarm should be take control even if another operator already has control. • NowInControl, to indicate that the operator ID specified is now in control. This will be returned if no operator has control, or if an operator has already has control and Force is set to true. • AlreadyUnderControl, to indicate that another operator already has control of this alarm. This will be returned if an operator has already has control and Force is set to false.

Returns:
Status response, EXgarde.AlarmControlStatus

public **void UpdateAlarmToStep**(**int** *AlarmID*, **int** *StepNumber*, **int** *OperatorID*, **string** *Comment*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Updates an alarm to a given step number

Parameters:
AlarmID: ID of alarm to process
StepNumber: Step number to update alarm to
OperatorID: ID of operator updating the alarm
Comment: The comment to add to the event

public **void ClearAlarm**(**int** *AlarmID*, **int** *OperatorID*, **string** *DeletionText*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Clears an alarm from the alarm queue

Parameters:
AlarmID: ID of alarm to clear
OperatorID: ID of operator clearing the alarm
DeletionText: The comment to use if the alarm is also to be cleared. This will only be used if the alarm procedure is also being cleared. The deletion text must be between 5 and 20 characters, otherwise an ArgumentException will be thrown.

4.15 Tenant Functions

public **System.Collections.Generic.IEnumerable<EXgarde.Data.Tenant> GetTenants**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get a list of tenants on the system

Returns:
An enumerable list of tenants, Exgarde.Data.Tenant

public **EXgarde.Data.Tenant GetTenant**(**int** *TenantID*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get tenant information

Parameters:
TenantID: ID of tenant

Returns:
Tenant information, Exgarde.Data.Tenant

public **EXgarde.Data.Tenant GetTenantByName**(**string** *TenantName*)

Member of **EXgarde.Server.EXGardeServer**

Summary:
Get tenant information by name

Parameters:
TenantName: Name of the tenant

Returns:
Tenant information, Exgarde.Data.Tenant

4.16 System Functions

public **EXgarde.Data.Info GetInfo**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Gets system information from EXgarde such as version number

Returns:

public **EXgarde.Data.SystemInfo GetSystemInfo**()

Member of **EXgarde.Server.EXGardeServer**

Summary:
Gets system information detailing number of controllers, readers and doors in the system

Returns:
SystemInfo structure. Exgarde.Data.SystemInfo

public **bool IsModuleEnabled**(**EXgarde.Module** *module*)

Member of **Exgarde.Server.EXGardeServer**

public **bool IsModuleEnabled**(**EXgarde.Module** *module*, ref **double** *NumberOfVisitors*)

Member of **EXgarde.Server.EXGardeServer**

4.17 Fusion Command Functions

public **System.Collections.Generic.List<EXgarde.Data.UGCommand> GetAllCommands**()

Member of **Exgarde.Server.EXGardeServer**

public **System.Collections.Generic.List<EXgarde.Data.UGCommand> GetCommandsInError**()

Member of **Exgarde.Server.EXGardeServer**

public **EXgarde.Data.UGCommand GetCommand**(**int** *CommandSequence*)

Member of **Exgarde.Server.EXGardeServer**

public **int DeleteCommand**(**int** *CommandSequence*)

Member of **Exgarde.Server.EXGardeServer**

public **int DeleteSuccessfullyProcessedCommands**()

Member of **EXgarde.Server.EXGardeServer**

5 Client Functions

5.1 Constructors

public **EXgardeClient**(**EXgarde.Network.Client.INetworkClient** *NetworkClient*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Creates a new instance of the EXgarde SDK client

Parameters:
NetworkClient: The network client to use

Exceptions:
System.ArgumentNullException: Thrown when NetworkClient is null

public **EXgardeClient**(**string** *Address*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Creates a new instance of the EXgarde SDK client

Parameters:
Address: The network address of the server component (eg

http://{MachineName}/Exgarde/sdk)

5.2 Connection Commands

public **void Connect**(**string** *DatabaseUserName*, **string** *DatabasePassword*, **string** *OperatorName*, **System.Action<EXgarde.ConnectionStatus>** *OnConnection*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Connects the network client to the SDK

Parameters:
DatabaseUserName: The name of the database user
DatabasePassword: The database password associated with the database user
OperatorName: The EXgarde operator name
OnConnection: A delegate used to notify when the connection has taken place. The delegate

argument indicates the status of the connection.

public **void Disconnect**()

Member of **EXgarde.Client.EXgardeClient**

Summary:
Disconnects the client from EXgarde

5.3 Keyholder Functions

public **void CreateKeyholder**(**string** *Name*, **string** *LongName*, **string** *Comment*, **System.Collections.Generic.List<string>** *Infos*, **System.Action<int>** *OnCreated*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Creates a new keyholder

Parameters:
Name: The name of the new keyholder
LongName: The long name
Comment: A descriptive comment
Infos: A list of information fields, containing additional keyholder details
OnCreated: The delegate that is called when the keyholder has been created. The value returned will be the ID of the new keyholder.

Remarks:
This command is based upon Fusion so is dependent upon the EXgarde Fusion process running.

Exceptions:
System.ArgumentNullException: If Name is null or

empty**System.ArgumentNullException**: Thrown when Name is null or empty

Public **void GetKeyholders**(**System.Action<System.Collections.Generic.List<EXgarde.Data.KeyholderInfo>>** *OnKeyholders*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a list of keyholders

Parameters:
OnKeyholders: The delegate that is called when the keyholders are available

public **void GetKeyholdersInArea**(**int** *AreaID*, **System.Action<System.Collections.Generic.List<EXgarde.Data.KeyholderInfo>>** *OnKeyholders*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a list of keyholders is an area

Parameters:
AreaID: The ID of the area
OnKeyholders: The delegate that is called when the keyholders are available

Remarks:
Keyholders in an area are detected by occupancy. If occupancy for a keyholder or area has been reset, keyholders may not show up in expected areas until they access those areas.

public **void GetKeyholder**(**int** *KeyholderID*, **System.Action<EXgarde.Data.KeyholderInfo>** *OnKeyholder*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a single keyholder

Parameters:
KeyholderID: The ID of the keyholder
OnKeyholder: The delegate that is called when the keyholder is available

public **void GetKeyholderImageData**(**int** *KeyholderID*, **System.Action<string>** *OnKeyholderImage*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests the keyholder image data.

Parameters:
KeyholderID: The ID of the keyholder
OnKeyholderImage: The delegate that is called when the keyholder image is available

Remarks:
Keyholder image data is used for the badge image and is returned as an array of bytes. You can use the standard classes in System.Drawing to load the bytes into an image; the sample application contains an ImageHandler class that shows how to do this.

public **void SaveKeyholder**(**int** *ID*, **string** *NewName*, **string** *LongName*, **string** *Comment*, **System.Collections.Generic.List<string>** *Infos*, **string** *PicturePath*, **System.Action<bool>** *OnSaved*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Saves changes to a keyholder

Parameters:
ID: The ID of the keyholder
NewName: The new name of the keyholder
LongName: The long name
Comment: A descriptive comment
Infos: A list of information fields, containing additional keyholder details
PicturePath: The path to a new picture
OnSaved: The delegate that is called when the keyholder has been saved. The value returned will be true if the keyholder command has been saved, false otherwise

Remarks:
This command is based upon Fusion so is dependent upon the EXgarde Fusion process running.

Exceptions:
System.ArgumentNullException: If NewName is null or empty

public **void DeleteKeyholder**(**string** *Name*, **System.Action<bool>** *OnDeleted*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Deletes a keyholder

Parameters:
Name: The name of the keyholder to delete
OnDeleted: The delegate that is called when the keyholder deleted command has been

created

Remarks:
This command is based upon Fusion so is dependent upon the EXgarde Fusion process

running.

Exceptions:
System.ArgumentNullException: If Name is null or empty

public **void AddKeyholderToGroup**(**string** *KeyholderName*, **string** *GroupName*)
Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Adds a keyholder to a group

Parameters:
KeyholderName: The name of the keyholder
GroupName: The name of the group

Exceptions:
System.ArgumentNullException: If KeyholderName is null or empty**System.ArgumentNullException**: If GroupName is null or empty

public **void RemoveKeyholderFromGroup**(**string** *KeyholderName*, **string** *GroupName*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Removes a keyholder from a group

Parameters:
KeyholderName: The name of the keyholder
GroupName: The name of the group

Exceptions:
System.ArgumentNullException: If KeyholderName is null or empty**System.ArgumentNullException**: If GroupName is null or empty

public **void GetGroups**(**System.Action<System.Collections.Generic.List<EXgarde.Data.Group>>** *OnGroups*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests a list of groups

Parameters:
OnGroups: The delegate that will be called when the groups are available

public **void GetGroups**(**int** *KeyholderID*, **System.Action<System.Collections.Generic.List<EXgarde.Data.Group>>** *OnGroups*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests a list of groups for a given keyholder

Parameters:
KeyholderID: The ID of the keyholder
OnGroups: The delegate that will be called when the groups are available

public **void GetGroup**(**int** *GroupID*, **System.Action<EXgarde.Data.Group>** *OnGroup*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Gets a single group

Parameters:
GroupID: The ID of the group
OnGroup: The delegate that will be called when the group is available

public **void IsKeyholderInGroup**(**int** *KeyholderID*, **int** *GroupID*, **System.Action<bool>** *OnInGroup*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Identifies whether or not a keyholder belongs to a group

Parameters:
KeyholderID: The Id of the keyholder
GroupID: The ID of the group
OnInGroup: The delegate that is called when the result is available

5.4 Key Functions

public **void CreateKeyBox**(**string** *Name*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Creates a new key box

Parameters:
Name: The name of the new key box

Exceptions:
System.ArgumentNullException: If Name is null or empty

public **void GetKeyBoxes**(**System.Action<System.Collections.Generic.List<EXgarde.Data.KeyBox>>** *OnKeyBoxes*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a list of key boxes

Parameters:
OnKeyBoxes: The delegate that will be called when the keys boxes are available

public **void GetKeyBox**(**int** *BoxID*, **System.Action<EXgarde.Data.KeyBox>** *OnKeybox*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Gets a single key box

Parameters:
BoxID: The ID of the key box
OnKeybox: The delegate that will be called when the key box is available

public **void ModifyKeyBox**(**string** *Name*, **string** *NewName*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Modifies a key box

Parameters:
Name: The existing name of the key box
NewName: The new name of the key box

Exceptions:
System.ArgumentNullException: If Name is null or empty**System.ArgumentNullException**: If *NewName* is null or empty

public **void DeleteKeyBox**(**string** *Name*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Deletes a key box

Parameters:
Name: The name of the key box to delete

Exceptions:
System.ArgumentNullException: If Name is null or empty

public **void CreateKey**(**int** *Number*, **string** *KeyholderName*, **string** *BoxName*, **EXgarde.KeyType** *Technology*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Creates a new key

Parameters:
Number: The number of the new key
KeyholderName: The name of an existing keyholder associated with the key. If the keyholder name is empty or null, the key is still created, but not assigned to a keyholder
BoxName: The name of an existing box into which the key is put. If the box is null or empty, the key is still created, but not placed in a box.
Technology: The technology type of the key. This will be one of the EXgarde.KeyType values.

public **void GetKeys**(**System.Action<System.Collections.Generic.List<EXgarde.Data.Key>>** *OnKeys*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests a list of keys

Parameters:
OnKeys: The delegate that will be called when the keys are available

public **void GetKey**(**int** *KeyID*, **System.Action<EXgarde.Data.Key>** *OnKey*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests a single key

Parameters:
KeyID: The ID of the key
OnKey: The delegate that will be called when the key is available

public **void IssueKey**(**int** *Number*, **EXgarde.KeyType** *Technology*, **string** *KeyholderName*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Issues a key to a keyholder

Parameters:
Number: The number of the key to issue
Technology: The technology type of the key This will be one of the EXgarde.KeyType values.
KeyholderName: The name of the keyholder

Exceptions:
System.ArgumentNullException: If KeyholderName is null or empty

public **void ReissueKey**(**int** *KeyId*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Reissues a key

Parameters:
KeyId: The ID of the key

public **void MarkKeyAsDamaged**(**int** *KeyId*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Marks a key as damaged

Parameters:
KeyId: The ID of the key

public **void MarkKeyAsLost**(**int** *KeyId*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Marks a key as lost

Parameters:
KeyId: The ID of the key

public **void SuspendKey**(**int** *KeyId*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Suspends a key

Parameters:
KeyId: The ID of the key

public **void ModifyKey**(**int** *Number*, **string** *KeyholderName*, **string** *BoxName*, **EXgarde.KeyType** *Technology*, **EXgarde.KeyStatus?** *Status*, **string** *Pin*, **System.DateTime?** *StartDate*, **System.DateTime?** *EndDate*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Modifies an existing key

Parameters:
Number: The number of the key
KeyholderName: The name of the keyholder assigned to the key
BoxName: The box name into which the key should be put
Technology: The technology type of the key This will be one of the EXgarde.KeyType values.
Status: The status of the new key. This will be one of the EXgarde.KeyStatus values, or null of no status is to be assigned.
Pin: The new PIN for the key
StartDate: The date from which the key is applicable. If null, the start date is not modified, if set to DateTime.MinValue the startdate for the key is cleared.
EndDate: The date to which the key is applicable. If null, the end date is not modified, if set to DateTime.MinValue the enddate for the key is cleared.

public **void DeleteKey**(**int** *Number*, **EXgarde.KeyType** *Technology*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Deletes a key

Parameters:
Number: The number of the key to delete
Technology: The technology type of the key, this will be one of the EXgarde.KeyType values.

5.5 ACU Functions

public **void GetACUs**(**System.Action<System.Collections.Generic.List<EXgarde.Data.ACUInfo>>** *OnACUs*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests a list of ACU details

Parameters:
OnACUs: The delegate that will be called when the details have been fetched. The list of ACUS are returned within the delegate parameter.

public **void GetACU**(**int** *ACUID*, **System.Action<EXgarde.Data.ACUInfo>** *OnACU*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Request details of a single ACU

Parameters:
ACUID: The ID of the ACU to fetch
OnACU: The delegate that will be called when the details have been fetched. The list of ACUS are returned within the delegate parameter.

public **void GetACUCommsState**(**int** *ACUID*, **System.Action<EXgarde.ACUState>** *OnACUCommandState*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests the communications status of an ACU

Parameters:
ACUID: The ID of the ACU
OnACUCommandState: The delegate that will be called when the status has been fetched. The ACU state is returned within the delegate parameter

5.6 Door Functions

public **void GetDoors**(**EXgarde.DoorFetch** *Fetch*, **System.Action<System.Collections.Generic.List<EXgarde.Data.DoorInfo>>** *OnDoors*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a list of all doors

Parameters:
Fetch: Inidicates whether or not to fetch the readers along with the doors. This will be one of the EXgarde.DoorFetch values
OnDoors: The delegate that will be called when the doors are available

Remarks:
When Fetch is set to IncludeReader, the reasers associated with doors will also be returned. This may cause a performance hit if you have a large number of doors.

public **void GetDoor**(**int** *DoorID*, **EXgarde.DoorFetch** *Fetch*, **System.Action<EXgarde.Data.DoorInfo>** *OnDoor*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a single door

Parameters:
DoorID: The ID of the door
Fetch: Inidicates whether or not to fetch the readers along with the doors. This will be one of the EXgarde.DoorFetch values
OnDoor: The delegate that will be called when the door is available

public **void AuthoriseAccess**(**int** *DoorID*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Authorises access through a door

Parameters:
DoorID: The ID of the door to authorise access through

public **void SetDoorState**(**int** *DoorID*, **EXgarde.DoorState** *NewState*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Sets the door to a given state

Parameters:
DoorID: The ID of the door
NewState: The new state. This will be one of the EXgarde.DoorState values.

public **void**
GetFireDoors(System.Action<System.Collections.Generic.List<EXgarde.Data.DoorInfo>>
OnDoors)

Member of EXgarde.Client.EXgardeClient

Summary:
Requests a list of fire doors

Parameters:
OnDoors: The delegate that will be called when the fire doors are available

public **void ResetFireDoors**()

Member of EXgarde.Client.EXgardeClient

Summary:
Resets all fire doors to their default state

5.7 Reader Functions

public **void**
GetReaders(System.Action<System.Collections.Generic.List<EXgarde.Data.ReaderInfo>>
OnReaders)

Member of EXgarde.Client.EXgardeClient

Summary:
Gets a list of readers

Parameters:
OnReaders: The delegate that will be called when the readers are available

public **void GetReader**(int *ReaderID*, System.Action<EXgarde.Data.ReaderInfo> *OnReader*)

Member of EXgarde.Client.EXgardeClient

Summary:
Gets a single reader

Parameters:
ReaderID: The ID of the reader to get
OnReader: The delegate that will be called when the read is available

5.8 Relay Functions

public **void**
GetRelays(System.Action<System.Collections.Generic.List<EXgarde.Data.Relay>> *OnRelays*)

Member of EXgarde.Client.EXgardeClient

Summary:
Requests a list of all relays2

Parameters:
OnRelays: The delegate that will be called when the relays are available.

public **void GetRelay**(int *RelayID*, System.Action<EXgarde.Data.Relay> *OnRelay*)

Member of EXgarde.Client.EXgardeClient

Summary:
Requests a single relay

Parameters:
RelayID: The ID of the relay to fetch
OnRelay: The delegate that will be called when the relay is available.

public **void SetRelayStatus**(int *RelayID*, EXgarde.RelayStatus *status*)

Member of EXgarde.Client.EXgardeClient

Summary:
Sets the state of a relay

Parameters:
RelayID: The ID of the relay
status: The new state of the relay. Will be one of the EXgarde.RelayStatus enumeration values

5.9 Input Functions

public **void GetInputs**(**System.Action<System.Collection.Generic,List<EXgarde.Data>InputInfo>>** *OnInputs*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a list of inputs

Parameters:
OnInputs: The list on inputs

public **void GetInput**(**int InputId, System.Action<EXgarde.Data.InputInfo>** *OnInputs)*

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a single input

Parameters:
OnInputs: The single input

5.10 Area Functions

public **void GetAreas**(**System.Action<System.Collections.Generic.List<EXgarde.Data.Area>>** *OnAreas*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a ist of all areas

Parameters:
OnAreas: The delegate that will be called when the areas are available.

public **void GetArea**(**int** *AreaID*, **System.Action<EXgarde.Data.Area>** *OnArea*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a single area

Parameters:
AreaID: The ID of the area to fetch
OnArea: The delegate that will be called when the area is available.

public **void GetAreaCount**(**System.Action<int>** *OnAreaCount*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests the count of areas

Parameters:
OnAreaCount: The delegate that will be called when the number of areas is available.

Occupancy Functions

public **void ResetOccupancyAll**()

Member of **EXgarde.Client.EXgardeClient**

Summary:
Resets the occupancy for all areas and keyholders

Remarks:
Occupancy tracks the areas in which keyholders last entered. This can become out of sync with the physical locations of keyholders, for example in fire drills. Resetting the entire occupancy moves all keyholders to the default occupancy area.

public **void ResetOccupancyForArea**(**int** *AreaID*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Resets occupancy for an entire area.

Parameters:
AreaID: The ID of the area

Remarks:
Occupancy for an area is defined as the keyholders that entered into the area but that have not left. This can become out of sync with the physical locations of keyholders, for example in fire drills. Resetting the occupancy for an area places the location of all of the keyholder in the area at the default occupancy area.
The default occupancy area is defined in the EXgarde configuration.

public **void ResetOccupancyForKeyholder**(**int** *KeyholderID*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Resets the occupance area for a given keyholder

Parameters:
KeyholderID: The ID of the keyholder

Remarks:
Occupancy for a keyholder is defined as the area into which they last entered. This can become out of sync with the physical location of the keyholder, for example if they tailgate. Resetting the occupancy places the keyholder's location at the default occupancy area.

The default occupancy area is defined in the EXgarde configuration.

5.11 Event Functions

public **void GetEvents**(**System.Action<System.Collections.Generic.List<EXgarde.Data.Event>>** *OnEvents*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Gets a list of all events

Parameters:
OnEvents: The delegate that will be called when the events are ready

Remarks:
Caution should be used in calling this method as the EXgarde event log could be very large. This call does not stream the events, so they will all be returned, which will impact performance of the database, the network and the client machine.

public **void GetEvents**(**EXgarde.EventNumber** *EventType*, **System.Action<System.Collections.Generic.List<EXgarde.Data.Event>>** *OnEvents*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests a list of events of a specific type

Parameters:
EventType: The type of event to return. This will be one of the EXgarde.EventNumber values
OnEvents: The delegate that will be called when the events are ready

public **void GetEvents**(**EXgarde.EventNumber** *EventType*, **System.DateTime** *From*, **System.DateTime** *To*, **System.Action<System.Collections.Generic.List<EXgarde.Data.Event>>** *OnEvents*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a list of events, filtered by time and time

Parameters:
EventType: The type of event to return. This will be one of the EXgarde.EventNumber values
From: The start date of the event filter
To: The end date of the event filter
OnEvents: The delegate that will be called when the events are ready

public **void GetEvents**(**System.DateTime** *From*, **System.DateTime** *To*, **System.Action<System.Collections.Generic.List<EXgarde.Data.Event>>** *OnEvents*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a list of events filtered by time

Parameters:
From: The start date of the event filter
To: The end date of the event filter
OnEvents: The delegate that will be called when the events are ready

public **void GetEvent**(**int** *EventID*, **System.Action<EXgarde.Data.Event>** *OnEvent*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a single event

Parameters:
EventID: The ID of the event
OnEvent: The delegate that will be called when the event is ready

public **void InsertEvent**(**EXgarde.EventNumber** *Event*, **int?** *ID1*, **int?** *ID2*, **int?** *ID3*, **int?** *ID4*, **int?** *ID5*, **int?** *AlarmProcedureID*, **string** *Text*, **System.DateTime** *LocalTime*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Inserts an event

Parameters:
Event: The event number. This will be one of the EXgarde.EventNumber values
ID1: The value for the ID1 field
ID2: The value for the ID2 field
ID3: The value for the ID3 field
ID4: The value for the ID4 field
ID5: The value for the ID5 field
AlarmProcedureID: The ID of any alarm procedure associated with the event
Text: Descriptive text for the event
LocalTime: The local time the event occurred

Remarks:
Inserting an event not only inserts the event details into the EXgarde database, but also raises the event as part of the standard EXgarde eventing system.

public **void InsertEvent**(**EXgarde.EventNumber** *Event*, **int?** *ID1*, **int?** *ID2*, **int?** *ID3*, **int?** *ID4*, **int?** *AlarmProcedureID*, **string** *Text*, **System.DateTime** *LocalTime*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Inserts an event

Parameters:
Event: The event number. This will be one of the EXgarde.EventNumber values
ID1: The value for the ID1 field
ID2: The value for the ID2 field
ID3: The value for the ID3 field
ID4: The value for the ID4 field
AlarmProcedureID: The ID of any alarm procedure associated with the event
Text: Descriptive text for the event
LocalTime: The local time the event occurred

Remarks:
Inserting an event not only inserts the event details into the EXgarde database, but also raises the event as part of the standard EXgarde eventing system.

public **void PauseEvents**()

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Pauses the client from receiving EXgarde events. The client remains connected to EXgarde and commands will continue to be executed, but no events will be received.

public **void ResumeEvents**()

Member of **EXgarde.Client.EXgardeClient**

Summary:
Resumes listening to EXgarde events.

public **void GetAuthorisationEvent**(**int** *UniqueID*, **System.Action<EXgarde.Data.AuthorisationEvent>** *OnAuthorisationEvent*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests an authorisation event

Parameters:
UniqueID: The unique ID of the event
OnAuthorisationEvent: The delegate that will be called when the event is available

public **void GetAuthorisationEventDetails**(**int** *UniqueID*, **int** *MultiCardPrimedID*, **System.Action<EXgarde.Data.AuthorisationEvent>** *OnAuthorisationEvent*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests details of the authorisation event

Parameters:
UniqueID: The unique ID of the event
MultiCardPrimedID: The ID of the card that initiated the authorisation
OnAuthorisationEvent: The delegate that will be called when the authorisation details are available

5.12 Alarm Functions

public **void GetAlarms**(**System.Action<System.Collections.Generic.List<EXgarde.Data.AlarmInfo>>** *OnAlarms*)

Member of **EXgarde.Client.EXgardeClient**

Summary:
Requests a list of alarms

Parameters:
OnAlarms: The delegate that will be called when alarms are available

public **void GetAlarms**(**bool** *Acknowledged*, **System.Action<System.Collections.Generic.List<EXgarde.Data.AlarmInfo>>** *OnAlarms*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests a list of alarms filtered by their acknowledged status

Parameters:
Acknowledged: Indicates the acknowledged state of alarms to return. If true, only ackowledged alarms will be returned, if false only un-acknowledged alarms will be returned.
OnAlarms: The delegate that will be called when the alarms are available

public **void GetAlarms**(**System.DateTime** *From*, **System.DateTime** *To*, **System.Action<System.Collections.Generic.List<EXgarde.Data.AlarmInfo>>** *OnAlarms*)

Member of **EXgarde.Client**.**EXgardeClient**
Summary:
Requests a list of alarms filtered by date
Parameters:
From: The start date of the filter
To: The end date of the filter
OnAlarms: The delegate that will be called when the alarms are available

public **void GetAlarms**(**System.DateTime** *From*, **System.DateTime** *To*, **bool** *Acknowledged*, **System.Action<System.Collections.Generic.List<EXgarde.Data.AlarmInfo>>** *OnAlarms*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests a list of alarms filtered by date and acknowledged status

Parameters:
From: The start date of the filter
To: The end date of the filter
Acknowledged: Indicates the acknowledged state of alarms to return
OnAlarms: The delegate that will be called when the alarms are available

public **void GetAlarmProcedures**(**System.Action<System.Collections.Generic.List<EXgarde.Data.AlarmProcedure>>** *OnAlarmProcedures*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests all alarm procedures

Parameters:
OnAlarmProcedures: The delegate that will be called when the procedures are ready

public **void GetAlarmProcedure**(**int** *AlarmProcedureID*, **System.Action<EXgarde.Data.AlarmProcedure>** *OnAlarmProcedure*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests a single alarm procedure

Parameters:
AlarmProcedureID: The ID of the alarm procedure
OnAlarmProcedure: The delegate that will be called when the procedure is ready

public **void GetAlarmCount**(**System.Action<int>** *OnAlarmCount*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests the count of alarms

Parameters:
OnAlarmCount: The delegate that will be called when the count is established

public **void GetAlarm**(**int** *AlarmID*, **System.Action<EXgarde.Data.AlarmInfo>** *OnAlarm*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Request an individual alarm

Parameters:
AlarmID: The ID of the alarm
OnAlarm: The delegate that will be called when the alarm is available

public **void GetAlarmEvent**(**int** *AlarmID*, **System.Action<EXgarde.Data.AlarmEvent>** *OnAlarmEvent*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests details of an alarm event

Parameters:
AlarmID: The ID of the alarm
OnAlarmEvent: The delegate that will be called when the alarm event details are available

public **void AcknowledgeAlarm**(**int** *AlarmID*, **int** *OperatorID*, **EXgarde.AlarmClear** *Clear*, **string** *DeletionText*, **System.Action<EXgarde.AlarmAcknowledgementStatus>** *OnAcknowledged*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Acknowledges an alarm

Parameters:
AlarmID: The ID of the alarm
OperatorID: The ID of the operator acknowledging the alarm
Clear: Indicates whether or not the alarm should be cleared as it is acknowledged
DeletionText: A description for the deletion text to be added during the acknowledgement; this is only applicable if the alarm is also being cleared.
OnAcknowledged: The delegate that will be called when the acknowledgement has taken place. The parameter value indicating the acknowledgement status will be one of the values from EXgarde.AlarmAcknowledgementStatus.

Remarks:
If the alarm doesn't have a procedure associated with it and it is to be cleared after acknowledgement, the deletion text must be between 5 and 20 characters inclusive, otherwise the alarm will not be acknowledged and the status will be set to Exgarde.AlarmAcknowledgementStatus.AcknowledgementFailedDueToTextLength

public **void TakeAlarmControl**(**int** *AlarmID*, **int** *OperatorID*, **EXgarde.Force** *force*, **System.Action<EXgarde.AlarmControlStatus>** *OnTakenControl*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Takes control of an alarm

Parameters:
AlarmID: The ID of the alarm
OperatorID: The ID of the operator taking control of the alarm
force: Indicates whether or not to force control of the alarm. Can be one of the EXgarde.Force values.
OnTakenControl: The delegate that will be called when control has been taken The parameter value indicating the acknowledgement status will be one of the values from Exgarde.AlarmControlStatus.

public **void UpdateAlarmToStep**(**int** *AlarmID*, **int** *StepNumber*, **int** *OperatorID*, **string** *Comment*, **System.Action<bool>** *OnUpdated*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Updates an alarm to a specific step number

Parameters:
AlarmID: The ID of the alarm
StepNumber: The step number to update the alarm to
OperatorID: The ID of the operator updating the alarm step
Comment: Descriptive text to be added as a comment for the update
OnUpdated: The delegate that will be called when the alarm has been updated

public **void ClearAlarm**(**int** *AlarmID*, **int** *OperatorID*, **string** *DeletionText*, **System.Action<bool>** *OnCleared*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Clears an alarm

Parameters:
AlarmID: The ID of the alarm
OperatorID: The ID of the operator clearing the alarm
DeletionText: A description for the deletion text to be added during the clearing. The deletion text must be between 5 and 20 characters inclusive.
OnCleared: The delegate that will be called once the alarm has been cleared

5.13 Tenant Functions

public **void GetTenants**(**System.Action<System.Collections.Generic.List<EXgarde.Data.Tenant>>** *OnTenants*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Gets a list of tenants

Parameters:
OnTenants: The delegate that will be called when the tenants are available

public **void GetTenant**(**int** *TenantID*, **System.Action<EXgarde.Data.Tenant>** *OnTenant*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Gets a single tenant

Parameters:
TenantID: The ID of the tenant to fetch
OnTenant: The delegate that will be called when the relay is available

public **void GetTenant**(**string** *TenantName*, **System.Action<EXgarde.Data.Tenant>** *OnTenant*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Gets a single tenant

Parameters:
TenantName: The name of the tenant to fetch
OnTenant: The delegate that will be called when the relay is available

5.14 System Functions

public **void GetInfo**(**System.Action<EXgarde.Data.Info>** *OnInfo*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests the product version details

Parameters:
OnInfo: The delegate that will be called when the details are available

public **void GetSystemInfo**(**System.Action<EXgarde.Data.SystemInfo>** *OnSystemInfo*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Requests the system information

Parameters:
OnSystemInfo: The delegate that will be called when the information is available

public **void IsModuleEnabled**(**EXgarde.Module** *module*, **System.Action<bool>** *OnIsEnabled*)

Member of **EXgarde.Client**.**EXgardeClient**

Summary:
Identifies whether or not a module is enabled and licensed

Parameters:
module: The module to check. One of the EXgarde.Module enumeration values
OnIsEnabled: The delegate that will be called when the information is available

5.15 Fusion Command Functions

public **void GetAllCommands**(System.Action<System.Collections.Generic.List<EXgarde.Data.UGCommand>> *OnCommands*)

Member of EXgarde.Client.EXgardeClient

Summary:
Requests a list of all Fusion commands

Parameters:
OnCommands: The delegate that will be called when the commands are available

public **void GetCommandsInError**(System.Action<System.Collections.Generic.List<EXgarde.Data.UGCommand>> *OnCommands*)

Member of EXgarde.Client.EXgardeClient

Summary:
Requests a list of all Fusion commands that are in error

Parameters:
OnCommands: The delegate that will be called when the commands are available

public **void GetCommand**(int *CommandSequence*, System.Action<EXgarde.Data.UGCommand> *OnCommand*)

Member of EXgarde.Client.EXgardeClient

Summary:
Requests a single Fusion command

Parameters:
CommandSequence: The unique ID of the command to fetch
OnCommand: The delegate that will be called when the command is available

public **void DeleteCommand**(int *CommandSequence*)

Member of EXgarde.Client.EXgardeClient

Summary:
Deletes a single Fusion command

Parameters:
CommandSequence: The unique ID of the command to delete

public **void DeleteSuccessfullyProcessedCommands**()

Member of EXgarde.Client.EXgardeClient

Summary:
Deletes all successfully processed Fusion commands

Page intentionally left blank

Page intentionally left blank

Page intentionally left blank

Page intentionally left blank

TDSi

ACCESS EVERYWHERE

Because everyone deserves peace of mind

TDSi UK

Unit 10 Concept Park, Innovation Close, Poole, Dorset BH12 4QT United Kingdom

t: +44 (0) 1202 723535 f: +44 (0) 1202 724975 e: sales@tdsi.co.uk

TDSi France

Immeuble ATRIA, 2 rue du Centre, 93160 NOISY LE GRAND France

t: +33 (0) 1 58 84 20 90 f: +33 (0) 1 58 84 20 91 e: tdsif@vanadoo.fr

bsia british security industry association

Beacon company