



Intel® Active Management Technology Redirection Library Design Guide

Version 3.0.5, March 2007

Information in this document is provided in connection with Intel® products. No license, express or implied, by estoppels or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

The API and software may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com>.

Copyright © 2006, 2007 Intel Corporation.

* Third party other names and brands may be claimed as the property of others.

1 Introduction

1.1 About this Guide

This document specifies the software functionality of the Intel® Active Management Technology (Intel® AMT) Redirection Library.

The software supports Serial Over LAN (SOL – text/keyboard redirection) and IDE Redirection (IDER – floppy/CD redirection) over TCP. This functionality is made available through a C interface for integration into third-party management consoles. In addition, accompanying sample code demonstrates this capability.

1.2 Terms and Abbreviations

The following terms and abbreviations are used throughout this document:

- **IDE:** Integrated Drive Electronics. A common interface between a storage device (in this case, a floppy disk or CD) and a computer.
- **IDER:** IDE redirection.
- **SOL:** Serial Over LAN (serial redirection).
- **MC / Management Console:** An application using the Redirection Library to interact with an Intel AMT machine.
- **IMR:** Intel Management Redirection. A common abbreviation used in the names of data types and functions in the library.
- **Host/Client:** A remotely managed Intel AMT-enabled machine.
- **FW:** The firmware embedded in Intel AMT.
- **UTF-8:** Unicode Transformation Format-8. A Unicode character encoding which is backward compatible with 7-bit US-ASCII encoding.
- **MFC:** Microsoft Foundation Class. An application framework for programming in Microsoft Windows.
- **PEM:** Privacy-enhanced Electronic Mail. An encoding standard specified in the Internet Engineering Task Force (IETF) RFC 1421.
- **SOAP:** Simple Object Access Protocol. SOAP is a lightweight XML based protocol used for invoking web services and exchanging structured data and type information on the Web. Used to communicate with most of the Intel AMT services.
- **WSDL:** Web Services Description Language. An XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. A WSDL encapsulates all the message formats available to communicate with an Intel AMT service.

1.3 Intel Active Management Technology Redirection Overview

Intel Active Management Technology makes it possible to redirect serial and IDE communications from a managed client to a management console regardless of the boot and power state of the managed client. The client need only have the Intel AMT capability, a connection to a power source and a network connection.

All redirection traffic is sent through the following IANA assigned ports:

- IDER/SOL over TCP = 16994
- IDER/SOL over TLS = 16995

1.3.1 Enabling SOL and IDER

For security purposes, Intel AMT requires that a series of steps be performed before SOL and IDER can operate. The first step is required in every instance, while the latter two steps are required only when Intel AMT is configured in Enterprise mode. These steps are:

- Enabling SOL and IDE-R in the BIOS Extension
- Enabling the SOL or IDER interface(s) using the Security Administration SOAP Interface
- Enabling the redirection listener sockets using the Redirection SOAP interface.

1.3.1.1 Enabling SOL/IDE-R in the BIOS Extension

The Intel AMT BIOS Extension Menu SOL/IDE-R menu item must be configured to enable the SOL /IDE-R capabilities. If SOL and IDE-R are not enabled here, these capabilities cannot be enabled remotely. Selecting the menu item displays three enable/disable selections. The first, to enable/disable username and password, determines whether the redirection interface can use a username and password to authenticate a remote SOL/IDE-R session. "Disable" limits the redirection interface to Kerberos authentication. The remaining two choices allow enabling/disabling SOL and IDE-R independently.

1.3.1.2 Enabling the SOL and IDE-R Interfaces Remotely

Once the BIOS extension settings are correct, a user with administrative privileges must execute the SetEnabledInterfaces SOAP command remotely, with "SerialOverLAN" or "IdeRedirection" or both as parameters. See the *Network Interface Guide* for a description of this command.

1.3.1.3 Enabling the Redirection Listener

The Redirection SOAP interface is used to enable the Intel AMT Redirection listener sockets. See the *Network Interface Guide* for a description of the SetRedirectionListenerState command. The [RedirectionConfig Application](#) is a sample program in the SDK that invokes this command.

1.3.2 Serial Over LAN Overview

Serial Over LAN (SOL) is the ability to emulate serial port communication over a standard network connection. SOL can be used for most management applications where a local serial port connection would normally be required.

When an active SOL session is established between an Intel AMT enabled client and a management console using the Intel AMT Redirection Library, the client's serial traffic is redirected through Intel AMT over the LAN connection and made available to the management console. Similarly, the management console may send serial data over the LAN connection that appears to have come through the client's serial port.

1.3.3 IDE Redirection Overview

IDE Redirection (IDER) emulates an IDE floppy or CD drive over a standard network connection. IDER enables a management machine to attach one of its floppy or CD drives to a managed client over the network. Once an IDER session is established, the managed client can use the IDE device as if it were directly attached to one of its own IDE channels. This can be useful, for example, for remotely booting an otherwise unresponsive system. IDER can transfer data from an internal IDE CD-ROM drive, from a standard internal 1.44MByte floppy drive, or from an LS-120 high capacity drive with either a 120 MByte disk or a standard 1.44 MByte disk. IDER does not support DVD format.

1.3.4 Typical SOL Scenario

In this example, SOL is used to diagnose or modify the BIOS settings of a managed client. The example assumes that the managed client supports a BIOS serial console that is auto-enabled when it detects a terminal connection to a serial port.

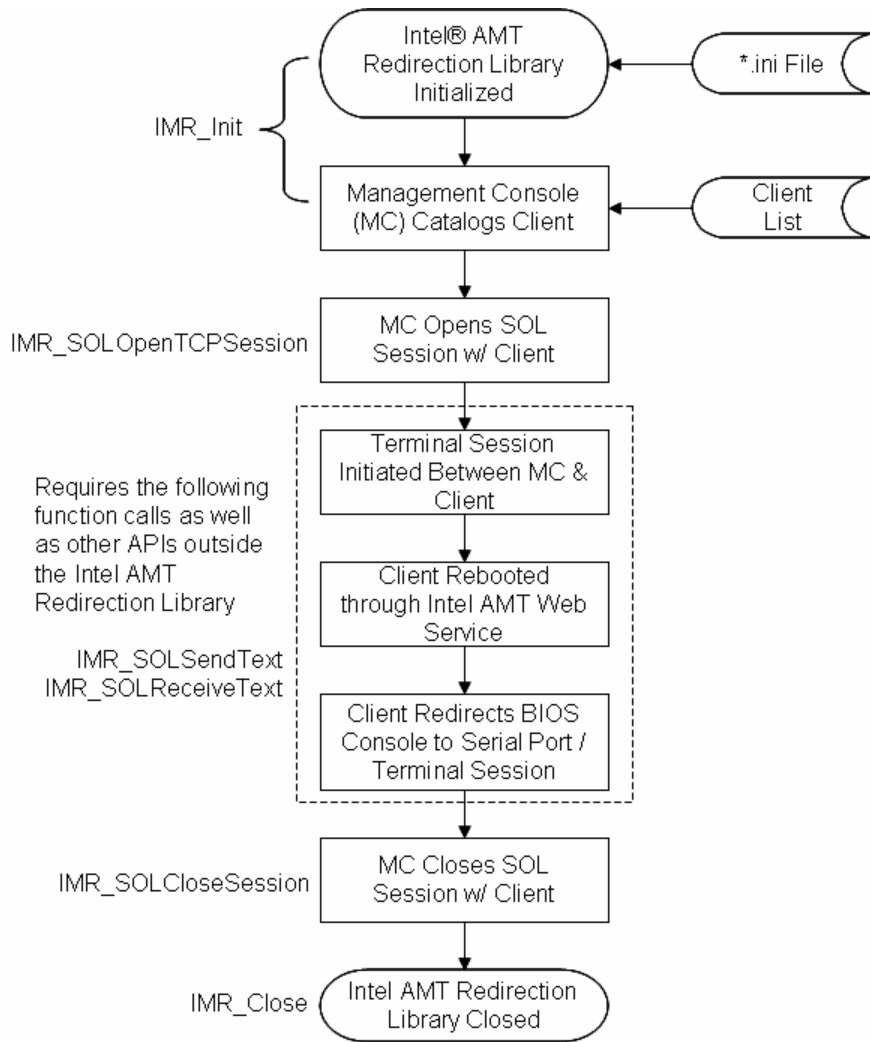


Figure 1: Serial Over LAN Scenario

1.3.5 Typical IDER Scenario

In this example, IDER is used to boot a client with a corrupt operating system. First, a valid boot disk is loaded into the management console disk drive. This drive is then passed as an argument when the management console opens the IDER TCP session. Intel AMT registers the device as a virtual IDE device on the client, regardless of its power or boot state. Both SOL and IDER may be used together since the client BIOS may need to be configured to boot from the virtual IDE device. The SOL steps are described in the previous example.

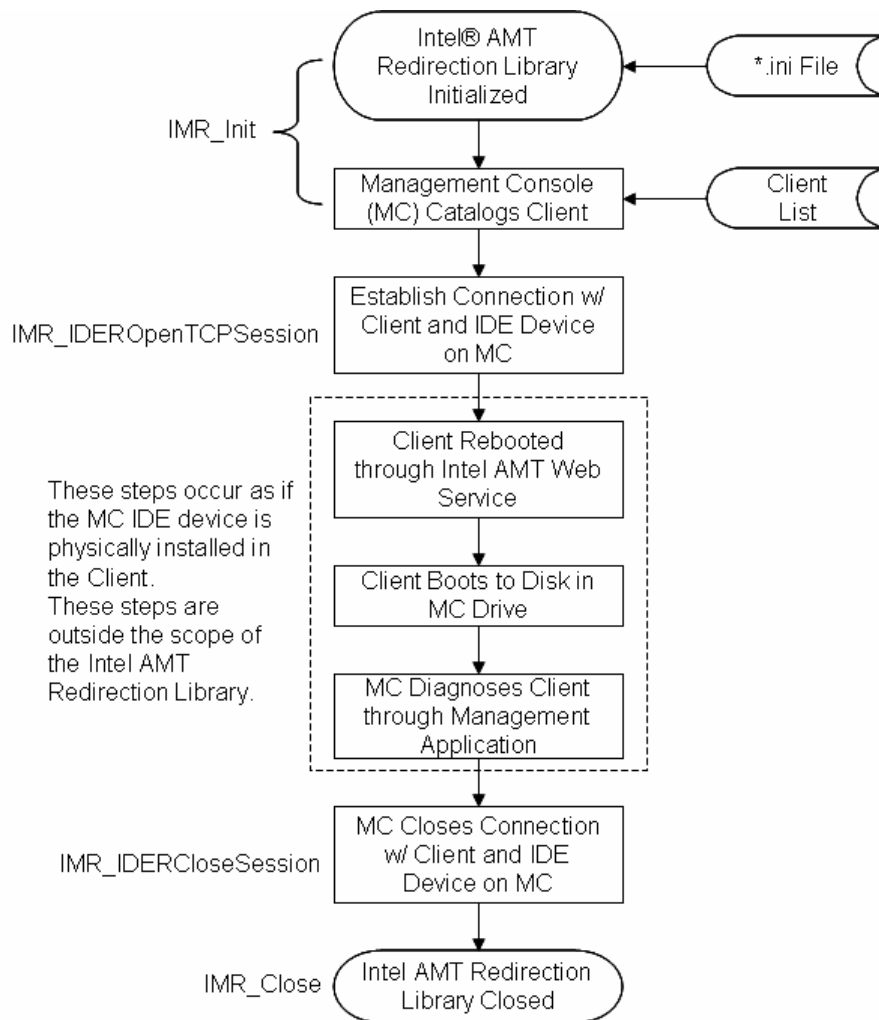


Figure 2: IDE Redirection Scenario

2 Package Overview

2.1 Components

The Redirection Package contains the following components:

- A Redirection Library: a C dynamic library (for Windows) and a C static library (for Linux), that provide support for:
 - Managing clients – maintaining a list of managed clients, adding and removing clients.
 - Performing SOL operations on a client (opening an SOL session, transferring text, etc.).
 - Performing IDER operations on a client (opening an IDER session, enabling/disabling IDE devices, etc.).
 - Storing Client states into a file for later retrieval (encrypting sensitive properties).
 - Translating error codes into error strings.
- Sample code for Windows of an MFC-based Management Console, which uses the Redirection Library to provide GUI support for:
 - Client management – showing existing clients' properties as well as adding and removing clients.
 - A terminal-like interface for SOL.
- Sample code for Linux, which uses the Redirection Library to provide support for:

- Client management – showing existing clients' properties as well as adding and removing clients.
 - Redirecting SOL data from/to the standard input/output.
 - Header files that define the library API to external applications.
- See the [Intel AMT SDK User Guide](#) for steps required to link to the Redirection Library.

2.2 Localization

The Redirection Library provides an interface for translating the library error codes into English strings. The strings are also provided outside of the library in a source file. Applications linking to the library can use the C interface to get the matching error string from the library itself, or use the source file to find the corresponding error strings independently. This allows for the replacement of the error strings source file with a translation file that contains messages in other languages.

2.3 Quantity Limitations

The Redirection Library supports the following:

- Simultaneous SOL/IDER sessions to multiple (different) clients.
- At most one SOL and one IDER session simultaneously to a single client.

2.4 Environment

For system requirements, refer to the System Requirements section of the Intel AMT User's Guide.

All redirection traffic is sent through the following IANA assigned ports:

- IDE-R/SOL over TCP = 16994
- IDE-R/SOL over TLS = 16995

2.5 Performance

IDE-R performance was measured over a range of environments. With 533 MHz memory and a TCP connection over a 1 GB switch, the transfer rate is over 1.4 MB per second, which is equivalent to a CD drive operating at a 9.6X transfer rate. TLS reduces the rate to 804 KB. Using faster memory improves performance.

The tests were performed using DMA (direct memory access). The alternative approach is to use PIO (Programmed Input/Output). For improved performance, it is recommended to use the BIOS setting, if available, to configure SOL/IDE-R to use DMA instead of PIO.

2.6 Additional Support: Error Logging

The library provides a logging mechanism for all components. At least three logging levels are supported (silent, errors only and verbose logging). The user can configure the logging level using the library ini file.

3 Redirection Library API

3.1 Functionality Overview

3.1.1 Client Management

The Redirection Library maintains an internal list of managed clients. Clients should be explicitly added to and deleted from the list by using the appropriate API functions. Any operation on a client that does not exist in the list will fail. When adding a new client, the library user must specify the client type: TCP (non-secure client) or TLS (secure client). Once a new client has been added to the library list, its type cannot be changed.

3.1.2 Client Storage

The internal client list maintained by the library can be stored in a file. This allows the library to save information about the clients even after the library (or the application using it) has shutdown.

The library's client storage feature is configurable by the user through the library ini file. If the client storage feature is disabled, the application using the Redirection Library can still save any desired client information in a storage repository of its choice (for example, in a file or in an external database). When the application restarts at a later time (and initializes the Redirection Library) it can use the repository it chose to restore the client information and add those clients to the library using the library API functions.

3.1.3 SOL

The library allows the user to open an SOL session with a client. Opening a session causes the client to redirect its serial I/O over the LAN. The user can use the library to send keyboard input and receive text output through the SOL protocol, over a TCP connection.

3.1.4 IDER

The Redirection Library allows the user to open an IDE Redirection (IDER) session with the client, which causes the client to redirect its IDE read & write operations over the LAN.

The library implements the IDER protocol and provides the client with the ability to read and write from a floppy or CD drive on the MC machine. The Windows version of the Redirection Library allows the user to specify a CD image file (*.iso) or a floppy image file (*.img) instead of a physical disk. This functionality is performed automatically by the library, without user intervention.

If you use an image file, make sure that it is available to the library for the whole IDER session. If the media containing the image (e.g. a disk-on-key or a CD or DVD containing the file) is removed during the session, the results are unpredictable.

3.1.5 Secure Session Support

Security is important for many Intel AMT features, especially for redirection. The usage model of SOL and IDER includes remote troubleshooting, including remote diagnostics, boot, and OS installation. These procedures usually involve authentication steps, so usernames and passwords will be sent over the LAN as part of the redirection session. To prevent sniffing of network passwords, redirection should be used only over a secure connection.

If the Intel AMT device supports TLS, the Redirection Library will establish a TLS session with it before opening SOL or IDER sessions, thus ensuring that all relevant network communications are secure.

From the TLS protocol point of view, the Intel AMT device is an SSL server and the Redirection Library is an SSL client. When establishing a TLS session, the library attempts to verify the validity of the SSL certificate it receives from the Intel AMT device. In order to perform the verification, the library must be provided with trusted Certificate Authority (CA) certificates that were used to sign the SSL server-provided certificate. The location of the trusted CA certificates is passed to the library using the `IMR_SetCertificateInfo()` function. If this file name is not provided, the library may not be able to verify SSL certificates, and thus will not be able to establish TLS sessions.

The specified file must contain the trusted root certificate in PEM format. The file can also contain subsequent subordinate certificates in the authorization path, identified by sequences of:

```
-----BEGIN CERTIFICATE-----  
... (CA certificate in base64 encoding) ...  
-----END CERTIFICATE-----
```

Before, between, and after the certificates, text is allowed that can be used for purposes such as descriptions of the certificates.

In addition to authenticating the SSL server certificate that is sent from the Intel AMT device, Intel AMT provides a mechanism for TLS mutual authentication, which means that the Redirection Library will send its own SSL client certificate. This feature can increase the security level of the redirection session. When

this capability is used, the SSL client certificate should be supplied to the Redirection Library (using the `IMR_SetCertificateInfo()` function). The file supplied to the library should contain the complete certificate chain and the private key in PEM format.

There are special requirements for the Intel AMT client certificates:

- The certificate needs to contain the OID 1.3.6.1.5.5.7.3.2, which marks the certificate as a TLS client certificate.
- Intel AMT mandates that the "Enhanced Key Usage" OID list field of the leaf certificate contains the OID 2.16.840.1.113741.1.2.1 (this OID is used by the Intel AMT device to authenticate the Redirection Library).

Note: To use the mutual authentication capability, the Intel AMT device should have as part of its trust list the CA certificate that has signed the SSL client certificate. This CA certificate should be provided to the Intel AMT device during the setup and configuration process.

3.1.5.1 An Example of Creating a Client Certificate Using Windows 2003 CA

This section contains detailed instructions on how to create a new Intel AMT-compatible client certificate using an existing root CA on Windows 2003 (e.g. corporate CA).

1. Go to the Windows 2003 machine containing the corporate CA.
2. Open Internet Explorer on the URL <http://localhost/certsrv>
3. Select "Request a certificate".
4. Select "Advanced certificate request".
5. Select "Create and submit a request to this CA".
6. In the "Name" field, type the fully qualified name of your host (host and domain name).
7. Fill the relevant fields of the certificate request (company, department, etc.).
8. Type of Certificate Needed: chose "Other..."
9. OID: the complete OID value must be "1.3.6.1.5.5.7.3.2,2.16.840.1.113741.1.2.1" without spaces.
10. Select "Mark keys as exportable"
11. Press the submit button.
12. Sign the certificate request.
13. Open Control Panel → Administrative Tools → Certificate Authority
14. Go to "Pending Requests" under your corporate root CA.
15. Right click your request and choose All Tasks → Issue.
16. Go to "Issued Certificates" under your corporate root CA.
17. Double click on your new certificate.
18. Choose Details → Copy to File... and save the certificate on your hard drive.
19. Double click on the certificate file you saved and choose "Install certificate".
20. From the start menu choose run and enter "mmc".
21. Choose File → Add/Remove Snap-in...
22. Press Add...
23. Choose Certificates and press Add and then Finish.
24. Press Close and then OK.
25. Expand the Personal folder under "Certificates - Current User".
26. Click on Certificates.
27. Right click your new client certificate and choose "All Tasks → Export..."
28. Choose to export the private key.
29. Enter a passphrase to protect the private key in the exported file.
30. Choose a filename (.pfx) and finish exporting the file.

The output of this section is a file with a .pfx extension that contains a TLS client certificate in the PKCS#12 format, and a private key. The pkcs12 OpenSSL utility can be used to convert the .pfx file to the .pem format (which is expected by the Redirection Library). This is done using the command:

```
openssl pkcs12 -in ClientCertIn.pfx -out ClientCertOut.pem
```

When issuing this command, you will be prompted for the passphrase which protects the .pfx file. You will then be prompted to enter a new passphrase that will protect the .pem file.

3.2 Error Codes

All of the library's API functions return error codes as their return values. These error codes (which are simply integer numbers) can be translated into human readable error strings by using the appropriate library API function. The error codes are enumerated in *IMRSDK.h*. *IMRerrors.cpp* has an example of a set of error strings.

Code	Value or Description
IMR_RES_OK	The requested operation was successfully executed.
IMR_RES_ERROR	Some unspecified error has occurred while executing the operation.
IMR_RES_INVALID_PARAMETER	An invalid parameter was specified to the function.
IMR_RES_NOT_INITIALIZED	The Redirection Library was not yet initialized.
IMR_RES_ALREADY_INITIALIZED	The Redirection Library has already been initialized.
IMR_RES_MEMALLOC_FAILED	Memory allocation has failed; not enough memory.
IMR_RES_UNSUPPORTED	The requested operation is unsupported by the library.
IMR_RES_CLIENT_NOT_FOUND	The specified client ID does not exist in the library's list.
IMR_RES_DUPLICATE_CLIENT	A client with the specified properties (e.g., IP address) already exists in the library's list.
IMR_RES_CLIENT_NOT_ACTIVE	The specified client has not been initialized by the library, although it exists in its list. This indicates an internal error in the library.
IMR_RES_CLIENT_ACTIVE	The specified client has already been initialized by the library. This indicates an internal error in the library.
IMR_RES_SESSION_ALREADY_OPEN	A session of the requested type (either SOL or IDER) is already opened with the specified client.
IMR_RES_SESSION_CLOSED	There is no open session of the requested type (either SOL or IDER) with the specified client.
IMR_RES_SOCKET_ERROR	A socket error has occurred.
IMR_RES_UNKNOWN_PROTOCOL	An internal error has occurred in the protocol handlers in the library.
IMR_RES_PROTOCOL_ALREADY_REGISTERED	An internal error has occurred in the protocol handlers in the library.
IMR_RES_PENDING	An internal error has occurred while trying to send a packet.
IMR_RES_UNEXPECTED_PACKET	An unexpected protocol packet has been received.
IMR_RES_TIMEOUT	A timeout has occurred while trying to execute the operation.
IMR_RES_CORRUPT_PACKET	A corrupted or malformed protocol packet has been received.
IMR_RES_OS_ERROR	An operating system error has occurred.
IMR_RES_IDER_VERSION_NOT_SUPPORTED	The client uses an IDER protocol of a version incompatible with the library's version.
IMR_RES_IDER_COMMAND_RUNNING	An IDER command is in progress.
IMR_RES_STORAGE_FAILURE	An error occurred while trying to store client information.
IMR_RES_UNKNOWN	An unknown library result code was specified.
IMR_RES_AUTH_FAILED	Authentication with the client has failed.

Code	Value or Description
IMR_RES_CLIENT_BUSY	The client already has an open session of the relevant function (either SOL or IDER) with some other application.
IMR_RES_CLIENT_UNSUPPORTED	The library attempted to connect to a host capability that does not support IDER.
IMR_RES_CLIENT_ERROR	General error return when trying to establish an IDER session.
IMR_RES_NOT_ENOUGH_SPACE	The function was provided with a buffer with insufficient space to successfully complete the operation.
IMR_RES_SESSION_LOOPBACK	The requested operation cannot be executed because the SOL session is in loopback mode.
IMR_RES_TLS_CONNECTION_FAILED	The library has failed to establish a TLS connection with the client. There might be a problem verifying the client's certificate.

3.3 ini file

Some of the library behavior can be configured using an *.ini file. The file is a text file divided into sections; each section contains lines of the following format:

```
<key> = <value>
```

A section name must be placed in brackets and appear before any of the key-value pairs belonging to that section. Section names and key names are case-insensitive, but key names may not contain white-space.

The number of white-space characters (and whether they are tabs or real spaces) is unimportant. The order of the keys in the file is irrelevant, and not all defined keys must be specified. If a key is not specified in the file, the library will use a default value. The path to the ini file may be passed to the library when it is initialized.

The following ini file keys are defined for the COMMON section (must be preceded by a line with the string [COMMON]):

Debug_Level Indicates the level of debug message logging. The default value is '0'. The possible values are:

- 0: silent – no messages will be logged.
- 1: errors – only error messages will be logged.
- 2: verbose – errors, warnings and debug messages will be logged.

Storage_Enabled A value of '0' means the library will not use a file for storing client information between library initializations. A non-zero value will cause the library to use the client storage feature. The default value is '1'.

The following ini file key is defined for the SECURITY section (must be preceded by a line with the string [SECURITY]):

CA_File The path to a file containing Certificate Authority certificates. See [Secure Session Support](#), above. The path name of the file can be either absolute or relative to the directory from which the application is run. ***This ini file key is deprecated and may be removed in future versions of the library. Use the IMR_SetCertificateInfo() library function instead.***

3.4 Data Types and Structures

3.4.1 Definitions

3.4.1.1 GUID_LEN

This value is used to indicate the length of a GUID.

```
#define GUID_LEN 16
```

3.4.1.2 INVALID_CLIENT_ID

This value is used to indicate an invalid client ID handle. It is not used as the client ID of any client managed by the library.

```
#define INVALID_CLIENT_ID (unsigned long) -1
```

3.4.1.3 MAX_IP_LEN

```
#define MAX_IP_LEN 128
```

3.4.1.4 MAX_NAME_LEN

This value is used to indicate the maximum length of a username.

```
#define MAX_NAME_LEN 128
```

3.4.1.5 MAX_PSWD_LEN

This value is used to indicate the maximum length of a password.

```
#define MAX_PSWD_LEN 128
```

3.4.2 Enumerations

3.4.2.1 ClientType

These enumerated values are defined to represent the client types. Each client has a type (according to its capabilities), which must be explicitly provided to the library when the client is first added.

```
typedef enum {
    CLI_TCP = 1,
    CLI_TLS = 2,
} ClientType;
```

Field	Value or Description
CLI_TCP	An Intel AMT (non-secure) client
CLI_TLS	An Intel AMT secure client

3.4.2.2 DeviceState

This enumeration defines possible IDER device states at the client. It is returned by the client when querying it about current IDER device states.

```
typedef enum {
    IDER_DISABLED,
    IDER_ENABLED
} DeviceState;
```

Field	Value or Description
IDER_DISABLED	The device is disabled.
IDER_ENABLED	The device is enabled.

3.4.2.3 SetOperation

This enumeration defines possible commands sent by the console to the client to change the state of the IDER devices.

```
typedef enum {
    IDER_ENABLE,
```

```

    IDER_DISABLE,
    IDER_NOP
} SetOperation;

```

Field	Value or Description
IDER_ENABLE	Request from the client to enable IDER devices.
IDER_DISABLE	Request from the client to disable IDER devices.
IDER_NOP	Request from the client to not change the IDER device state. Currently not supported. If specified, the library will return an error.

3.4.2.4 SetOption

This enumeration defines possible timing options for the set command defined in section 3.5.4.5.

```

typedef enum {
    IDER_SET_ONRESET,
    IDER_SET_GRACEFULLY,
    IDER_SET_IMMEDIATELY,
} SetOption;

```

Field	Value or Description
IDER_SET_ONRESET	Change of device at PCI reset. This is the preferred option to synchronize IDE device enable/disable. At PCI reset, the host typically restarts and then enumerates its devices and IDE cables for devices. The client must accept the “at reset” timing and respond with a result status of IDER_DONE. Otherwise, it will respond with IDER_REJECTED. Upon PCI reset, the host applies both primary and secondary cable settings.
IDER_SET_GRACEFULLY	Change the specified device settings in a graceful manner: If the client Intel AMT FW detects that there is a pending IDE command from its host, it will reject this request with a response of IDER_REJECTED. Otherwise, the client will respond with IDER_DONE.
IDER_SET_IMMEDIATELY	Change the specified cable register settings immediately, whether there is a pending IDE command from its host or not. The client should respond with IDER_DONE.

3.4.2.5 SetResult

This enumeration defines possible SetDevice result values returned by the client for each IDER device.

```

typedef enum {
    IDER_REJECTED,
    IDER_DONE
} SetResult;

```

Field	Value or Description
IDER_REJECTED	This value is returned by the client FW when the client receives a set command when an IDE or ATAPI command is pending. In such a case, disabling the IDE interface may result in unpredictable host behavior and should be avoided. This result informs the console that the IDE interface is in active use.
IDER_DONE	Disable/Enable successfully done.

3.4.3 Types

3.4.3.1 ClientID

Each client managed by the library has a ClientID handle assigned to it. A ClientID handle value is assigned to a client when the client is added to the library. The handle is used to reference the client in all subsequent operations. The handle remains permanently associated with that client, as long as the library has not shut down.

```
typedef unsigned long ClientID;
```

3.4.3.2 GUIDType

This type is used to hold the Globally Unique ID of a client. This value should be unique for each client, and it can be optionally provided to the library when a client is added.

```
typedef char GUIDType[GUID_LEN];
```

3.4.4 Structures

3.4.4.1 ClientInfo

This structure is used for providing information for a specific client.

```
typedef struct {
    ClientType type;
    char ip[MAX_IP_LEN];
    GUIDType guid;
} ClientInfo;
```

Field	Value or Description
type	The client's type (CLI_TCP, CLI_TLS)
ip	A NULL-terminated string holding either the client IP address in numbers and dots notation (nnn.nnn.nnn.nnn) or the valid hostname of the client.
guid	The Globally Unique ID of the client

3.4.4.2 FeaturesSupported

This is the response structure when querying the supported IDER features of the client with an IMR_IDERClientFeatureSupported command.

```
typedef struct {
    BOOL ider_dev_pri;
    BOOL ider_dev_sec;
    BOOL reserved[30];
} FeaturesSupported;
```

Field	Value or Description
ider_dev_pri	TRUE if client FW supports enable/disable of primary IDE devices.
ider_dev_sec	TRUE if client FW supports enable/disable of secondary IDE devices.
reserved	This field is reserved for future use.

3.4.4.3 IDERDeviceCmd

This structure is used as an input parameter when sending a "set IDE devices state" command to the client.

```
typedef struct {
```

```

        SetOperation    pri_op;
        SetOption      pri_timing;
        SetOperation    sec_op;
        SetOption      sec_timing;
    } IDERDeviceCmd;

```

Field	Value or Description
pri_op	Primary device operation (enable or disable).
pri_timing	Primary device operation timing.
sec_op	Secondary device operation (enable or disable). (Not supported by the IMR library)
sec_timing	Secondary device timing. (Not supported by the IMR library)

3.4.4.4 IDERDeviceResult

This structure is used as an output parameter when setting client IDE devices state. The IMR library will overwrite the structure fields with command results.

```

typedef struct {
    SetResult    pri_res;
    SetResult    sec_res;
} IDERDeviceResult;

```

Field	Value or Description
pri_res	Set command result for primary devices.
sec_res	Set command result for secondary devices. The SDK does not support enabling/disabling secondary IDE devices. This field should be ignored.

3.4.4.5 IDERDeviceState

This structure is used as an output parameter when querying client IDE devices state. The IMR library will overwrite structure fields with query results.

```

typedef struct{
    DeviceState pri_default;
    DeviceState pri_current;
    DeviceState sec_default;
    DeviceState sec_current;
} IDERDeviceState;

```

Field	Value or Description
pri_default	Default primary devices state at the Client FW.
pri_current	Current primary devices state at the Client FW.
sec_default	Default secondary devices state at the Client FW.
sec_current	Default secondary devices state at the Client FW.

3.4.4.6 IDERStatistics

This structure is used as an output parameter when getting the IDER session statistics. The IMR library will overwrite structure fields with appropriate values.

```

typedef struct {
    BOOL            error_state;
    BOOL            data_transfer;
    unsigned short  num_reopen;
    unsigned long   num_error;
}

```

```

    unsigned long    num_reset;
    unsigned long    last_cmd_length;
    unsigned long    data_sent;
    unsigned long    data_received;
    unsigned long    packets_sent;
    unsigned long    packets_received;
} IDERStatistics;

```

Field	Value or Description
error_state	TRUE if session is in ERROR state.
data_transfer	TRUE if there is a read/write command in progress.
num_reopen	Number of session re-opens due to error recovery.
num_error	Number of ErrorOccured messages received.
num_reset	Number of ResetOccured messages received.
last_cmd_length	Last data transfer (read/write) length in bytes.
data_sent	Bytes of data sent to the client.
data_received	Bytes of data received from the client.
packets_sent	Messages sent during the session.
packets_received	Messages received during the session.

3.4.4.7 IDERTout

This structure is used to pass time-out parameters when establishing an IDER session.

```

typedef struct {
    unsigned short    rx_timeout;
    unsigned short    tx_timeout;
    unsigned short    hb_timeout;
} IDERTout;

```

Field	Value or Description
rx_timeout	<p>The client receive timeout, in milliseconds. If this much time passes before receiving any messages from the library, the client shuts down the IDER session. When an IDER session is open, the library continually sends out messages to make sure that the client's receive timeout does not expire (the library heartbeat interval is based on the client receive timeout setting).</p> <p>Minimum value: 10000 Maximum value: 65535 Default value: 10000</p>
hb_interval	<p>The client heartbeat interval. This is the number of milliseconds the client waits before sending a heartbeat message to the library. A value of 0 means that no heartbeat messages are sent. In this case, the library will periodically send IDER keep-alive ping messages to the client when there is no activity, to determine if it is still alive.</p> <p>Minimum value: 0 Maximum value: 65535 Default value: 5000</p>

Field	Value or Description
tx_timeout	The client's command transmit timeout. This is the number of milliseconds the clients waits when sending out an IDE command. If the client does not receive a response from the library to the command within the specified amount of milliseconds, the client will close the IDER session. A value of 0 means that no command transmit timeout is used. Minimum value: 0 Maximum value: 65535 Default value: 0

3.4.4.8 IMRVersion

This structure is used to return the library API version.

```
typedef struct {
    unsigned short major;
    unsigned short minor;
} IMRVersion;
```

Field	Value or Description
major	The major version number. This number is updated only if the API is changed in a way that is incompatible with previous versions.
minor	The minor version number. This number is updated if the API is extended in a way that does not affect backward compatibility.

3.4.4.9 TCPSessionParams

This structure is used to pass the parameters required to establish an SOL or IDER session.

Note: The lengths of the user name and user password are limited by the Redirection SDK to 32 characters. If either the user name or password exceeds this limit, IMR_RES_INVALID_PARAMETER, will be returned.

```
typedef struct {
    char user_name[MAX_NAME_LEN];
    char user_pswd[MAX_PSWD_LEN];
} TCPSessionParams;
```

Field	Value or Description
user_name	The user's name, a NULL-terminated string, UTF-8 encoded.
user_pswd	The user's password, a NULL-terminated string, UTF-8 encoded.

3.4.4.10 SOLLoopbackMode

These values are defined to select a loopback mode for SOL sessions. If an SOL session is opened in loopback mode, the client will not send its outgoing serial data over the LAN, but instead will loop it back to itself as incoming serial data.

```
typedef enum {
    SOL_LOOPBACK_NONE,
    SOL_LOOPBACK_RS232C,
} SOLLoopbackMode;
```

Field	Value or Description
SOL_LOOPBACK_NONE	No loopback is used in the SOL session.

Field	Value or Description
SOL_LOOPBACK_RS232C	The session uses loopback as defined by the RS232C specification in Microsoft Windows Hardware Compatibility Test (HCT) Kit, in the Serial Port Stress Test section.

3.4.4.11 SOLTout

This structure is used to pass time-out parameters when establishing an SOL session.

```
typedef struct {
    unsigned short    tx_over_timeout;
    unsigned short    tx_buf_timeout;
    unsigned short    hb_interval;
    unsigned short    fifo_rx_flush_timeout;
    unsigned short    rx_timeout;
} SOLTout;
```

Field	Value or Description
tx_over_timeout	The client transmit overflow timeout. This is the number of milliseconds the client waits when its transmit buffer is full before starting to drop transmit bytes. A value of 0 means no timeout. Minimum value: 0 Maximum value: 65535 Default value: 0
tx_buf_timeout	The client transmit buffering timeout. This is the number of milliseconds the client waits for its transmit buffer to become full before sending its buffered transmit bytes. A value of 0 means that the client will transmit its data only when its buffer becomes full. Minimum value: 0 Maximum value: 65535 Default value: 100
hb_interval	The client heartbeat interval. This is the number of milliseconds the client waits between sending heartbeat messages to the library, indicating that the client is active. A value of 0 means that no heartbeats are sent. In this case, the library will not monitor the receive activity from the client to determine if it is active. Minimum value: 0 Maximum value: 65535 Default value: 5000
fifo_rx_flush_timeout	The client's FIFO receive flush timeout. This is the number of milliseconds the client waits when its receive FIFO is full before flushing its received data. A value of 0 means that the client never flushes its received data when it is not read by the operating system. Minimum value: 0 Maximum value: 65535 Default value: 100

Field	Value or Description
rx_timeout	<p>The client's receive timeout. If this number of milliseconds passes before receiving any messages from the library, the client shuts down the SOL session. When an SOL session is open, the library periodically sends heartbeat messages to make sure that the client receive timeout does not expire (the interval between library heartbeat messages is based on the client receive timeout).</p> <p>Minimum value: 10000 Maximum value: 65535 Default value: 10000</p>

3.5 Functions

3.5.1 Functions for General Library Management

3.5.1.1 IMR_Init

This function initializes the library. It must be called before calling any other API function of this library.

Function Header

```
IMRResult IMR_Init(
    IMRVersion *version,
    char *ini_file
);
```

Function Parameters

Field	Input/Output	Value or Description
version	Output	A pointer to a structure that will be filled with the library's version. This parameter may be NULL if no version information is desired.
ini_file	Input	A NULL-terminated string specifying the name of an INI file, which can be either absolute or relative to the directory from which the application is run. This parameter may be NULL, in which case the library will use its default settings.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_ALREADY_INITIALIZED	The library has already been initialized.

3.5.1.2 IMR_Close

This function shuts down the library, releasing all system resources. After calling this function, no other API function of this library may be called, except for IMR_Init().

Function Header

```
IMRResult IMR_Close();
```

Function Parameters

Field	Input/Output	Value or Description
None	NA	NA

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_NOT_INITIALIZED	The library has not been previously initialized.

3.5.1.3 IMR_GetErrorStringLength

This function returns the length of the error string associated with a specified error code.

Function Header

```
IMRResult IMR_GetErrorStringLength(
    IMRResult res,
    int *str_len
);
```

Function Parameters

Field	Input/Output	Value or Description
res	Input	One of the library error codes.
str_len	Output	A pointer to an integer that will be filled with the length of the error string matching the specified error code (including the terminating NULL). This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_UNKNOWN	The specified error code 'res' is not a known result code of the library.

3.5.1.4 IMR_GetErrorString

This function returns the error string associated with the specified error code.

Function Header

```
IMRResult IMR_GetErrorString(
    IMRResult res,
    char *str
);
```

Function Parameters

Field	Input/Output	Value or Description
res	Input	One of the library error codes.
str	Output	A pointer to a buffer that will be filled with the error string matching the specified error code (including the terminating NULL). The buffer must long enough for the string. This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_UNKNOWN	The specified error code 'res' is not a known result code of the library.

3.5.1.5 IMR_SetCertificateInfo

This function sets information that is necessary for certificate verification when establishing a TLS session.

Function Header

```
IMRResult IMR_SetCertificateInfo(
    const char *root_cert,
    const char *private_cert,
    const char *cert_pass
);
```

Function Parameters

Field	Input/Output	Value or Description
root_cert	Input	A NULL terminated string specifying the path to a file containing Certificate Authority certificates as described in the Secure Session Support section. When establishing a TLS session with a client, these root certificates will be used to authenticate the SSL certificates sent by the client. The file path name can be either absolute or relative to the directory from which the application is run. This parameter must not be NULL.
private_cert	Input	A NULL terminated string specifying the path to a file containing a private certificate as described in the Secure Session Support section. When establishing a TLS session with a client, this certificate will be sent to the client if it requires the console to provide its own certificate for authentication. The file path name can be either absolute or relative to the directory from which the application is run. This parameter may be NULL, in which case the library will not be able to provide a certificate if one is requested by the client.
cert_pass	Input	A NULL terminated string specifying the password protecting the certificate file specified in 'private_cert'. This parameter may be NULL, in which case 'private_cert' should not be password protected. If 'private_cert' is NULL, then this parameter must also be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.

3.5.2 Functions for Client Management

3.5.2.1 IMR_AddClient

This function adds a client to the library internal client list.

Function Header

```
IMRResult IMR_AddClient(
    ClientType type,
    char *ip,
    GUIDType guid,
    ClientID *id
);
```

Function Parameters

Field	Input/Output	Value or Description
type	Input	The type of client to be added. Can be one of: CLI_TCP (for a non-secure client) or CLI_TLS (for a secure client).
ip	Input	A pointer to a NULL-terminated string holding either the client's IP address in numbers and dots notation or a valid hostname. The IP address must not be used by another client that was already added to the library. This parameter must not be NULL.
guid	Input	The client's Global Unique ID. This parameter may be NULL, in which case the default GUID value (all bytes set to 0) will be stored for the client.
id	Output	A pointer to a ClientID variable that will be filled by the library with a handle for the newly added client. This handle should be used in all future interactions with the client via the library. This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_DUPLICATE_CLIENT	A client with the specified properties (e.g. IP address) already exists in the library's list.

3.5.2.2 IMR_RemoveClient

This function removes a client from the library's internal client list.

Function Header

```
IMRResult IMR_RemoveClient(
    ClientID id
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the client that is to be removed from the list.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_CLIENT_NOT_FOUND	The specified client ID does not exist in the library's list.

3.5.2.3 IMR_RemoveAllClients

This function removes all clients from the library's internal client list.

Function Header

```
IMRResult IMR_RemoveAllClients();
```

Function Parameters

Field	Input/Output	Value or Description
None	NA	NA

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.

3.5.2.4 IMR_GetAllClients

This function returns the handles of all the clients in the library's internal list.

Function Header

```
IMRResult IMR_GetAllClients(
    ClientID *client_list,
    int *list_size
);
```

Function Parameters

Field	Input/Output	Value or Description
client_list	Output	An array of ClientIDs to be filled by the library. This parameter may be NULL, in which case only the size of the library's client list is returned, via the 'list_size' parameter.
list_size	Input/Output	Input: A pointer to an integer. When calling the function, the integer's value must be the size of the buffer pointed to by 'client_list', in ClientID units (that is, how many ClientIDs can fit in the 'client_list' buffer). Output: If there is enough space in the 'client_list' buffer, the buffer will be filled with the ClientIDs of all of the clients managed by the library, and the 'list_size' argument will be updated to be the number of ClientIDs that were actually written into the buffer. If there are more clients managed by the library than there is space in the buffer, then the library will return in the 'client_list' buffer as many as can fit, the 'list_size' argument will be updated to be the number of clients managed by the library and the call will fail with the IMR_RES_NOT_ENOUGH_SPACE error code. This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_MEMALLOC_FAILED	Not enough memory.

Status	Value or Description
IMR_RES_NOT_ENOUGH_SPACE	Not enough space is available in the 'client_list' buffer to hold the entire client list. The library will return as many as can fit, and return the total number of clients available in 'list_size'.

3.5.2.5 IMR_GetClientInfo

This function returns the properties of the specified client.

Function Header

```
IMRResult IMR_GetClientInfo(
    ClientID id,
    ClientInfo *info
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.
info	Output	A pointer to a struct that will be filled by the library with the client's properties. This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.

3.5.3 Functions for SOL Handling

3.5.3.1 IMR_SOLOpenTCPSession

This function opens an SOL session with the specified client over a new TCP connection. If the client uses secure communications, a TLS session will be negotiated with it. The function returns only when the session is established or if an error has occurred.

Function Header

```
IMRResult IMR_SOLOpenTCPSession(
    ClientID id,
    TCPSessionParams *params,
    SOLTouts *touts,
    SOLLoopbackMode *loopback
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.
params	Input	A pointer to a struct with session parameters. This parameter must not be NULL.
touts	Input	A pointer to a struct with timeout parameters. This parameter may be NULL, in which case the default values of the parameters will be used. See the struct definition for the default values.

Field	Input/Output	Value or Description
loopback	Input	A pointer to a variable specifying the requested serial loopback mode for the SOL session. Opening an SOL session in loopback mode will cause the remote client to loopback all serial data to itself, without sending it over the LAN. This is useful for testing and diagnostics use cases. If a loopback is requested but such a mode is not supported by the client, the variable pointed to by 'loopback' will be updated to be SOL_LOOPBACK_NONE. This parameter may be NULL, in which case no loopback is used.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_SESSION_ALREADY_OPEN	The library has already opened an SOL session with the specified client.
IMR_RES_CLIENT_BUSY	The specified client already has an open SOL session with another application.

3.5.3.2 IMR_SOLCloseSession

This function closes an open SOL session with the specified client. After the function returns no other operations can be performed until a new session is initialized.

Function Header

```
IMRResult IMR_SOLCloseSession(
    ClientID id
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_SESSION_CLOSED	The specified client does not have an open SOL session.

3.5.3.3 IMR_SOLSendText

This function sends text (keyboard input) to the client, where it will be received as incoming data from the serial controller. This function should not be called if the SOL session was opened in loopback mode (since in loopback mode the client will loopback all data to itself, and should not receive data from the library).

Function Header

```
IMRResult IMR_SOLSendText(
    ClientID id,
    unsigned char *data,
    int len
);
```

```
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.
data	Input	A pointer to a buffer of data to be sent. This parameter must not be NULL.
len	Input	The length of the data to be sent. This value must be greater than zero.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_SESSION_LOOPBACK	The session is in loopback mode.

3.5.3.4 IMR_SOLReceiveText

Data sent by the client on the serial controller is received by the library and stored in an internal buffer. This function retrieves SOL data that has been stored. The user should poll for new data periodically and then loop over IMR_SOLReceiveText until the len parameter is zero to avoid data loss due to the buffer overflowing. This is similar to having to receive data from a socket to avoid it being lost if the socket buffer becomes full. This function should not be called if the SOL session was opened in loopback mode, since in loopback mode the client will loopback all its data to itself, and would not send data to the library.

Function Header

```
IMRResult IMR_SOLReceiveText(
    ClientID id,
    unsigned char *data,
    int *len
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.
data	Output	A pointer to a buffer into which received data will be copied. This parameter must not be NULL.
len	Input/Output	Input: A pointer to an integer holding the size available in the specified data buffer. Output: The library updates this variable to the actual number of bytes that were copied into the buffer. If after returning, the value is 0, this means no data is pending to be received. This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_SESSION_LOOPBACK	The session is in loopback mode.

3.5.4 Functions for IDER Handling

3.5.4.1 IMR_IDEROpenTCPSession

This function opens an IDER session with the specified client over a new TCP connection. If the client uses secure communications, a TLS session will be negotiated with it. The function returns only when the session is established or if an error has occurred. After the session is open the client can initiate data transfers.

Function Header

```
IMRResult IMR_IDEROpenTCPSession(
    ClientID id,
    TCPSessionParams *params,
    IDERTouts *touts,
    char *drive0,
    char *drive1
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.
params	Input	A pointer to a struct with session parameters. This parameter must not be NULL.
touts	Input	A pointer to a struct with timeout parameters. This parameter may be NULL, in which case the default values of the parameters will be used. See the struct definition for the default values.
drive0	Input	A pointer to a NULL-terminated string holding the name of the floppy disk drive on the Management Console platform that will be used for IDER (e.g. "A:"). For the Windows dynamic library, the string can also hold the path to a binary floppy image file. This parameter must not be NULL.
drive1	Input	A pointer to a NULL-terminated string holding the name of the CD drive on the Management Console platform that will be used for IDER (e.g. "E:"). For the Windows dynamic library, the string can also hold a path to an ISO CD image file. This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_SESSION_ALREADY_OPEN	An IDER session is already open with the specified client.
IMR_RES_CLIENT_BUSY	The specified client already has an open IDER session with another application.

3.5.4.2 IMR_IDERCloseSession

This function closes an open IDER session with the specified client. After the function returns no other operations can be performed. Closing a session during a write data operation can cause data corruption.

Function Header

```
IMRResult IMR_IDERCloseSession(
    ClientID id
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.
IMR_RES_SESSION_CLOSED	The specified client does not have an open IDER session.

3.5.4.3 IMR_IDERClientFeatureSupported

This function queries the client about the special features that it supports. Currently the only special feature defined is an ability to disable/enable host IDE devices.

Function Header

```
IMRResult IMR_IDERClientFeatureSupported(
    ClientID id,
    FeaturesSupported *sup
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.
sup	Output	A pointer to a struct that will be filled with the features supported by the client. This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.

3.5.4.4 IMR_IDERGetDeviceState

This function queries the state of client IDE devices.

Function Header

```
IMRResult IMR_IDERGetDeviceState(
    ClientID id,
    IDERDeviceState *state
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.

Field	Input/Output	Value or Description
state	Output	A pointer to a struct that will be filled by the library with the client's IDER device states. This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.

3.5.4.5 IMR_IDERSetDeviceState

This function controls the client IDE device(s) state. Devices can be disabled and enabled through this function.

Function Header

```
IMRResult IMR_IDERSetDeviceState(
    ClientID id,
    IDERDeviceCmd *cmd,
    IDERDeviceResult *result
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.
cmd	Input	A pointer to a struct that holds disable/enable command and options. This parameter must not be NULL.
result	Output	A pointer to a struct that will be filled by the library with the result of the command. This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.

3.5.4.6 IMR_IDERGetSessionStatistics

This function polls the active IDER session. If the IDER session for the corresponding client was closed due to a timeout or protocol error the function will fill the stat structure with the latest data, but will return IMR_RES_SESSION_CLOSED.

Function Header

```
IMRResult IMR_IDERGetSessionStatistics(
    ClientID id,
    IDERStatistics *stat
);
```

Function Parameters

Field	Input/Output	Value or Description
id	Input	The ID of the relevant client.

Field	Input/Output	Value or Description
stat	Output	A pointer to a struct that will be filled by the library with the IDER session statistics. This parameter must not be NULL.

Function Return Status

See section 3.2 for a complete list of possible error codes.

Status	Value or Description
IMR_RES_OK	The command finished successfully.

4 Windows Sample Applications

The SDK include two sample programs. The IMRGUI program has a graphics interface and demonstrates the capabilities of the redirection library. The RedirectionConfig application uses the SOAP/HTML interface to configure the Intel AMT device for redirection operations and to display the current configuration.

4.1 RedirectionConfig Application

The Redirection listener ports are disabled by default on Intel AMT devices. The ports must be enabled for the IMRGUI sample program to interact with an Intel AMT-managed workstation. Until the ports are enabled, a management console cannot communicate with Intel AMT using the Redirection Library.

The RedirectionConfig application operates outside the framework of the Redirection Library. It communicates with the target Intel AMT device with using SOAP/HTTP communications with a special WSDL. The command has four options:

- **-g** : Get the current RedirectionListenerState
- **-e** : Enable the Redirection service listener
- **-d** : Disable the Redirection service listener
- **-a** : perform API test

Use the **-e** option to enable redirection before executing the Redirection GUI Console.

The following example enables redirection on a host named “hostname” without using TLS.

```
redirectionconfig -e http://hostname:16992/RedirectionService
```

The following example performs the same function using TLS and provides a certificate to be used for authentication with the Intel AMT device.

```
redirectionconfig -e -certName MyCert  
https://hostname:16993/RedirectionService
```

Entering the command without parameters displays the all of the command options.

4.2 GUI Application Functionality Overview

The GUI sample application consists of an MFC-based simple GUI Console. Using the GUI, the user can add clients to the Console or delete them. Double-clicking on a selected client in the main window will open the Client Dialog. In the Client Dialog, the user has the ability to open or close SOL and IDER sessions with the Intel AMT client and to configure session parameters. The source code of the sample application demonstrates how to build a GUI console on top of the Redirection Library.

4.2.1 Windows GUI Sample Project Tree Overview

The sample application files are placed under Samples\Redirection\IMRGUI folder.

Significant files in this folder:

1. IMRGUISample.vcproj – Visual Studio project file. Use it to compile the project from Visual Studio.
2. .\Include, .\src and .\res subdirectories of the folder contain the remaining source files.

Other files in the SDK that support the redirection sample:

1. IMRGUI.exe – Pre-built Sample Application Console, ready to run, located in the \Bin Folder.
2. MC.ini – Sample INI file passed to the Redirection Library at startup, located in the \Bin Folder.
3. imrsdk.dll – Copy of INTEL Management Redirection Library DLL, located in the \Bin Folder.
4. imrsdk.lib – Copy of the Export Library for the above IMRSDK.DLL, which is required to compile the application successfully, located in the \Lib Folder.

IMTSDK.h in the global SDK Include folder contains function prototypes and other definitions for applications linking to the Redirection Library.

Developers using version 2.7 of the Intel AMT SDK in support of Intel AMT 1.0 can link with the Intel AMT 2.0 version of imrsdk.dll if IMRGUI is compiled with Visual Studio 6 Service Pack 5. Compiling with an earlier version of Visual studio 6 will result in compile errors.

4.2.2 Windows Sample GUI Overview

4.2.2.1 Application Main Window

The GUI Main window consists of two sub-panels. In the upper panel, active clients are displayed. Events for each client will appear in the lower panel.

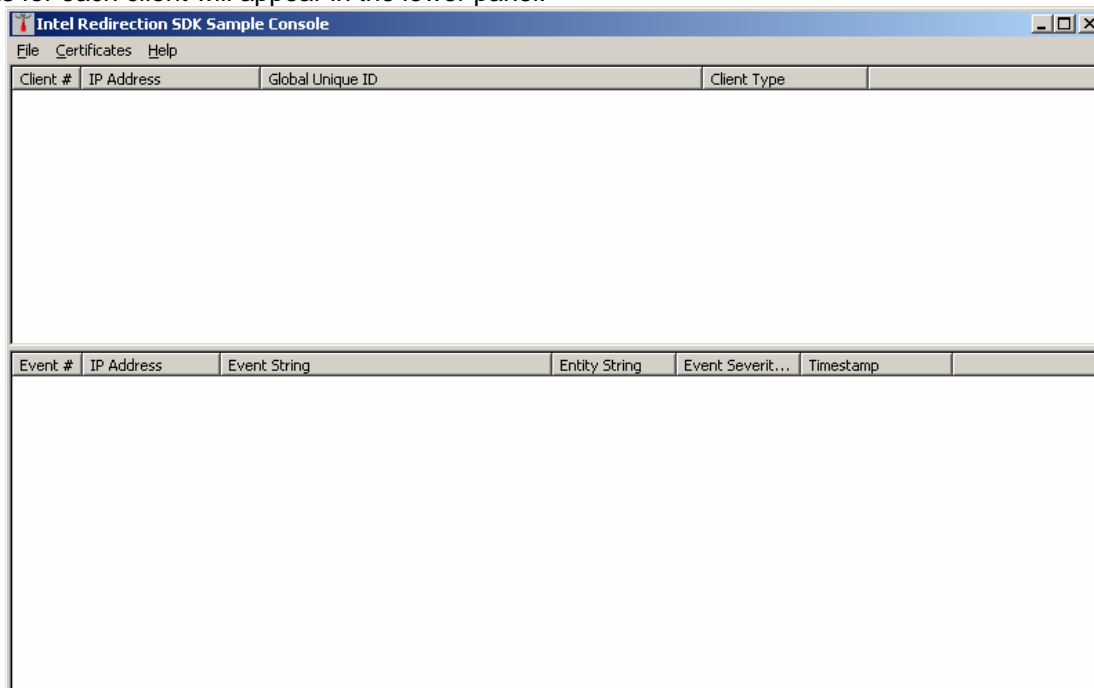


Figure 3: Sample Application Main Window

4.2.2.2 Add Client Dialog

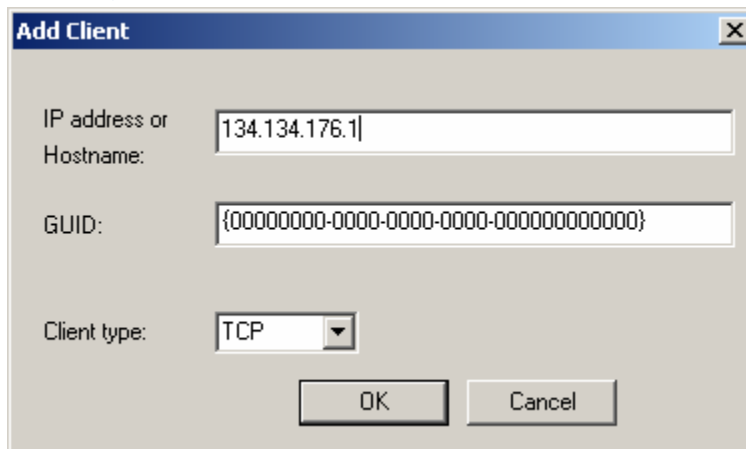


Figure 4: Add Client Dialog

Use “File|Add Client” to add a new client. Each client added must have a unique IP address. “Client type” should be properly selected when adding the client but “GUID” is optional.

4.2.2.3 Certificates Dialog

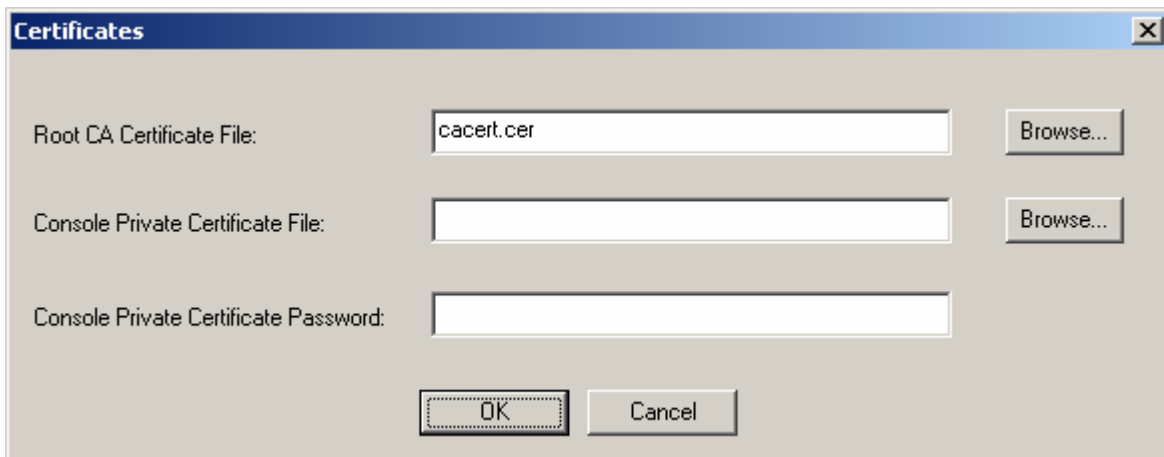


Figure 5: Certificates Dialog

Use “File|Certificates” to set certificate information. See the Secure Session Support section for details.

4.2.2.4 Client Dialog

Double clicking on the selected client in the Main Window upper sub-panel brings up the following dialog:

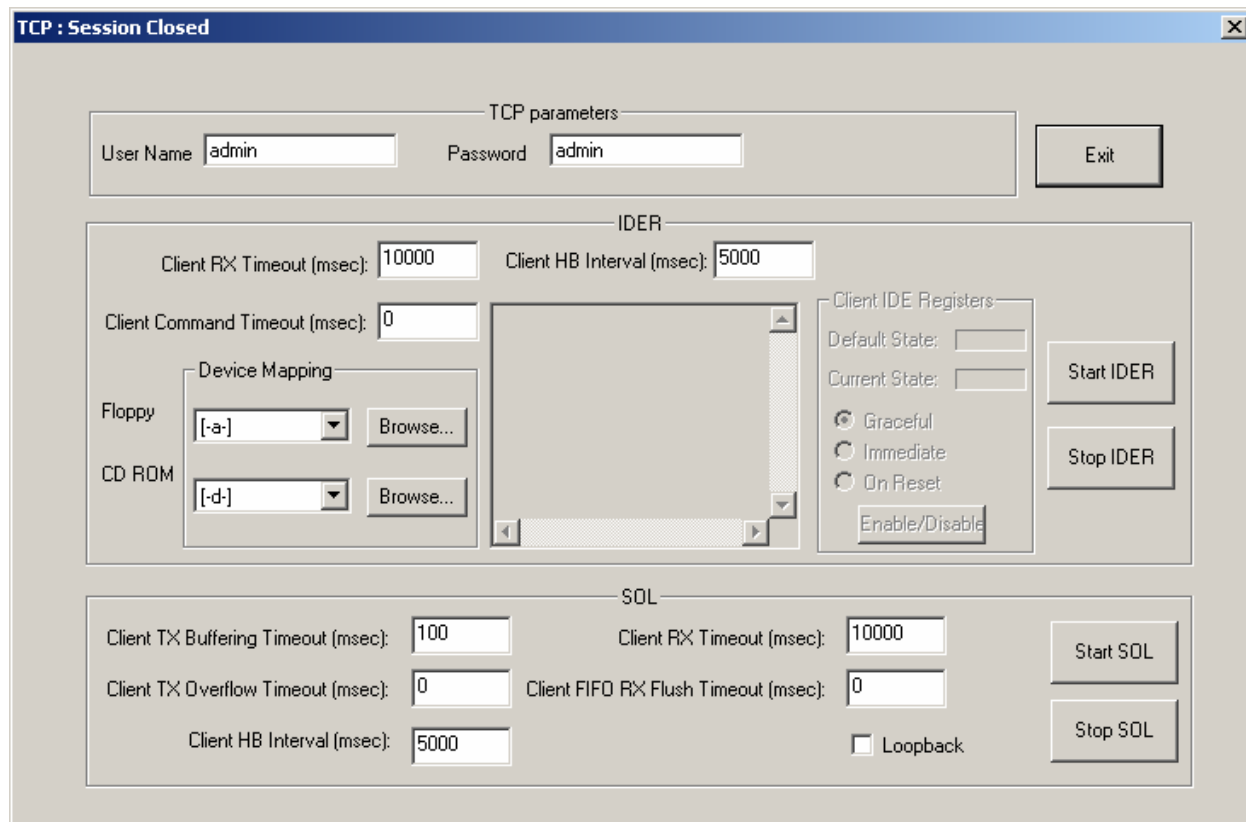


Figure 6: Client Dialog

This dialog has three groups of controls: TCP parameters, IDER and SOL.

TCP Parameters:

Parameter	Value or Description
User Name	User name for the Intel AMT Client user authentication.
User Password	Password for the Intel AMT Client user authentication.

Note: The lengths of the user name and user password are limited by the Redirection SDK to 32 characters. If either the user name or password supplied exceeds this limit the error code, IMR_RES_INVALID_PARAMETER, will be returned.

IDER:

Parameter	Value or Description
Client RX Timeout	The client's receive timeout in milliseconds for this session (see definition of the IDERTout struct).
Client Command Timeout	The client's IDE command transmit timeout in milliseconds for this session (see definition of the IDERTout struct).
Client HB Interval	The interval between two consecutive client Heartbeat messages in milliseconds for this session (see definition of the IDERTout struct).
Device Mapping	Use the controls to map the devices for the IDER session. "Floppy" combo box corresponds to the Floppy device used during the session. Select a 1.44 MB/LS 120 Floppy drive or binary image of a Floppy Disk here. "CD ROM" combo box corresponds to the CD ROM Device used during the session. Select a valid CD ROM drive or an ISO CD image here.
Start IDER	Press to open the session.

Parameter	Value or Description
Stop IDER	Press to close the session.
Client IDE Registers (group of controls)	Will be enabled once the session is established and if the client FW supports the Disable/Enable of IDER devices. Use Enable/Disable button to send the appropriate command to the client. One of the "Graceful", "Immediate" or "On Reset" timing options for the command can be selected.
The Statistics Edit Box	Will display the current IDER session statistics once the session is established.

Important Note: Any new files written to a floppy locally on the management console platform while it is being read by the Intel AMT device will not be detected by the device unless the disk is removed and re-inserted or the platform with Intel AMT is power cycled.

SOL:

The IMRGUI Windows sample application launches the PuTTY* Terminal Emulator when opening an SOL session. To ensure that the application can find the PuTTY executable, either place the putty.exe file in the same directory as the application or set the PATH environment variable to point to its location. The putty.exe is included in the SDK in the Bin directory.

After the application launches PuTTY, configure its settings as follows:

1. Right-click on the PuTTY window title bar, and select **Change Settings...**
2. Select **Terminal** and change both **Local Echo** and **Local Line Editing** to **Force Off**. Select **Terminal-Keyboard** and change the sequence sent by the Function keys and keypad to be **VT100+** and the sequence sent by the Backspace key to be **"Control-H"**.
3. Optionally, select **Session** and **Save** to save these settings for future sessions.
4. Click **Apply** for the settings to take effect.

To use the Windows Telnet client instead of PuTTY, set the variable "TERM_CMD" at the top of the MCSOL.cpp file to be "TELNET_CMD", instead of "PUTTY_CMD". In this case, make sure that the Windows Telnet client is installed on your system.

Note that this application **does not** implement a Telnet server. As a result, the behavior of the Telnet client, which expects to operate opposite a server, may not be what is expected (for example, display issues relating to local echo).

Parameter	Value or Description
Client TX Buffering Timeout	The client transmit buffering timeout in milliseconds for this session (see definition of the SOLTout struct).
Client TX Overflow Timeout	The client transmit overflow timeout in milliseconds for this session (see definition of the SOLTout struct).
Client HB Timeout	The interval between two consecutive client Heartbeat messages in milliseconds for this session (see definition of the SOLTout struct).
Client RX Timeout	The client receive timeout in milliseconds for this session (see definition of the SOLTout struct).
Client FIFO RX Flush Timeout	The client FIFO receive flush timeout in milliseconds for this session (see definition of the SOLTout struct).
Loopback	Check this box to enable the SOL loopback.

Parameter	Value or Description
Start SOL	Press to open the session. A Telnet window will open when the session is established. Use it to transfer the SOL data. The sample application communicates via a TCP socket using a pre-defined port number specified in the sample application. Because of this, only a single instance of the sample application can open an SOL session at a time.
Stop SOL	Press to close the session.

5 Linux Sample Application

5.1 Functionality Overview

When running the application from a command shell, the user will be presented with a list of all available clients that are managed by the redirection library, as well as a menu of available options. By selecting options from the menu, the user will be able to add clients to and delete clients from the console. Selecting a specific client from the list of available clients will display a menu with available options for the client, such as opening/closing SOL and IDER sessions.

The imrcli application provides only a basic redirection library example. Most parameters (like session timeout values) are hard-coded in the application.

5.2 Linux Sample Project Tree Overview.

The sample application files are placed under the RedirectionSDK folder.

Significant files in this folder:

1. imrcli.cpp – The source code of the sample application.
2. IMRerrors.cpp – A source file containing the strings of the various error codes of the library.
3. Makefile – The Makefile for compiling the sample application, using the 'make' command.
4. MC.ini – Sample INI file passed to the Redirection Library at startup.

5.3 SOL

The SOL functionality is demonstrated by redirecting text from/to the standard input/output. While an SOL session is open, all text from the standard input (keyboard) is redirected to the Intel AMT client (Ctrl-D is used to terminate the SOL session). This means that while an SOL session is open, it is impossible to issue other commands to the imrcli sample application, such as opening an IDER session. To have both an SOL session and an IDER session open simultaneously, open the IDER session before opening the SOL session.

When an SOL session is open, text from the keyboard is sent to the remote Intel AMT client without echoing it locally to the standard output. The characters will only be displayed if the Intel AMT client echoes them back to the console. This behavior can be changed by modifying the imrcli sample application's source code.

5.4 IDER

When opening an IDER session, the sample application will request the names of the floppy and CDROM devices that are to be redirected (these devices are usually /dev/fd0 and /dev/cdrom, respectively). To redirect these devices, the user must make sure that they are mounted to a file system, using the 'mount' command (e.g. /dev/cdrom is usually mounted to /mnt/cdrom).

To change the floppy or CD during an IDER session, the user must first unmount the device's file system (using the 'umount' command), and then mount the file system again after the media has been changed.

IMPORTANT: *changing floppy diskettes during an IDER session without first unmounting the original floppy's file system may cause data corruption of the new floppy if it is not write-protected.*

Any new files written to a floppy locally on the management console platform while it is being read by the Intel AMT device will not be detected by the device unless the disk is removed and re-inserted or the platform with Intel AMT is power cycled.