



ArgusConnect Pty Ltd
ABN 63107558387
Suite 9, GreenHill Enterprise Centre
University Drive, Ballarat Victoria 3353
Tel: 0415 645 291
Email: andrew.s@argusconnect.com.au
Website: <http://www.argusconnect.com.au>

Using the Argus API

Last Updated: August 2007

Table of Contents

1	Introduction	1
1.1	Purpose of Document	1
1.2	Terms and Abbreviations	1
2	Using the Argus API	2
2.1	What is the Argus API?	2
2.2	How do I install the Argus API?	2
2.2.1	Installing the Development Kit	2
2.2.2	Argus API Versions	2
2.3	How do I use the Argus API?	2
2.3.1	Calling the Argus API via COM	3
2.3.2	Calling the Argus API via DLL	3
2.3.3	Calling the Argus API via Java	4
2.4	Argus API Functions	4
2.4.1	Initial / Required Functions	4
2.4.2	Frequently Used Functions	4
2.4.3	Adding a Message	5
2.4.4	Complete Function List	5
2.5	Code Examples	5
2.5.1	Calling Common Functions	5
2.5.2	Other Sample code/applications	7
2.6	Troubleshooting	7
Appendix A:	COM Object Function Calls	9
2.7	Introduction	9
2.8	Functions for connecting to Argus	9
2.9	Functions for logging into Argus database	9
2.10	Functions for producing debug information	9
2.11	Functions for dropping messages into the ArgusMail outbox	10
2.12	Functions for initiating sending and receiving of messages	11
2.13	Functions for retrieving messages from the Argus database	11
2.14	Functions for address book handling	12
2.15	Miscellaneous functions	13

1 Introduction

1.1 Purpose of Document

This is a detailed guide outlining the features and use of the *Argus API*. This guide presupposes a high-level understanding of Argus, as detailed on the ArgusConnect website.

1.2 Terms and Abbreviations

Term	Definition
Argus	The suite of programs that provides for the secure mail-exchange between health care providers (GPs, pathology labs, specialists etc.)
ArgusMessenger	This is a program that interacts with POP3/SMTP, and IMAP servers. It implements advanced HL7 handling and encryption via PKI. One instance of ArgusMessenger runs at each location.
ArgusMail	This is the 'client', a program that allows users to view and send email. An instance of ArgusMail runs on each user's desktop.
ArgusAgent	This is a program that can be run in background mode to fire defined events at designated intervals. Tasks include the automatic sending of documents dropped into a directory, and polling for files to be printed automatically.
ArgusAPI	This is an API module that allows other applications to interact with Argus without using the Argus GUI. Non-windows applications can interact directly with a Java class in Argus itself, and Windows applications can use COM.
Message	A mail item, downloaded from the mail server and stored in an Interbase database. As with any email, messages often include attachments.
Argus Users Directory	An LDAP Server hosted by ArgusConnect, whose purpose is to be a national repository of the details of all current Argus users. From version 4.1 of Argus onwards, all new users of Argus will be added to the Argus Users Directory with their clinical email address and public certificate. Access to the Argus Users Directory is restricted to Argus users.
Argus Address Book	Addresses which are stored in the Argus database (locally) for an installation of Argus. These entries can be updated automatically from the Argus Users Directory on a regular basis.
Folder	This does not refer to a file directory. It is used here to denote a mail folder in ArgusMail, such as the Inbox.

2 Using the Argus API

2.1 What is the Argus API?

The *Argus API* is an Application Programming Interface which allows other applications to directly call Argus and utilise Argus functions for sending and receiving messages. Using the API has a number of benefits over other methods of interacting with Argus (such as dropping files for *ArgusAgent*). For example, programs using the Argus API can receive real-time confirmation that a particular message has been accepted by Argus and is ready to be sent.

2.2 How do I install the Argus API?

The COM object which interfaces to the Argus API is registered by default by the Argus installer. As of Argus 4.2, 'Client' installations involve linking to the Argus install on a central 'server' machine via file sharing across an internal network. Running the 'Client Setup' program will register the Argus API COM objects (registry entries are created in HKEY_CLASSES_ROOT for 'ArgusAPI_4_2_apiwrapper') and thus allow access to the API from that particular machine.

2.2.1 Installing the Development Kit

The Argus API and ArgusWord Development Kit contain various resources for developers intending to use the API, including example projects written in Delphi and VB. The Developer Kit it is available as a feature in the Argus installer for Windows however it is not installed by default, so choose 'Custom' during the install to select the Developer Kit feature for install.

Users not wishing to download the Argus Installer or not running Windows can download the Development Kit from the ArgusConnect subversion repository, accessible at [http://www.argusconnect.com.au/subversion/trunk/projects/DelphiProjects/Argus API/](http://www.argusconnect.com.au/subversion/trunk/projects/DelphiProjects/Argus%20API/).

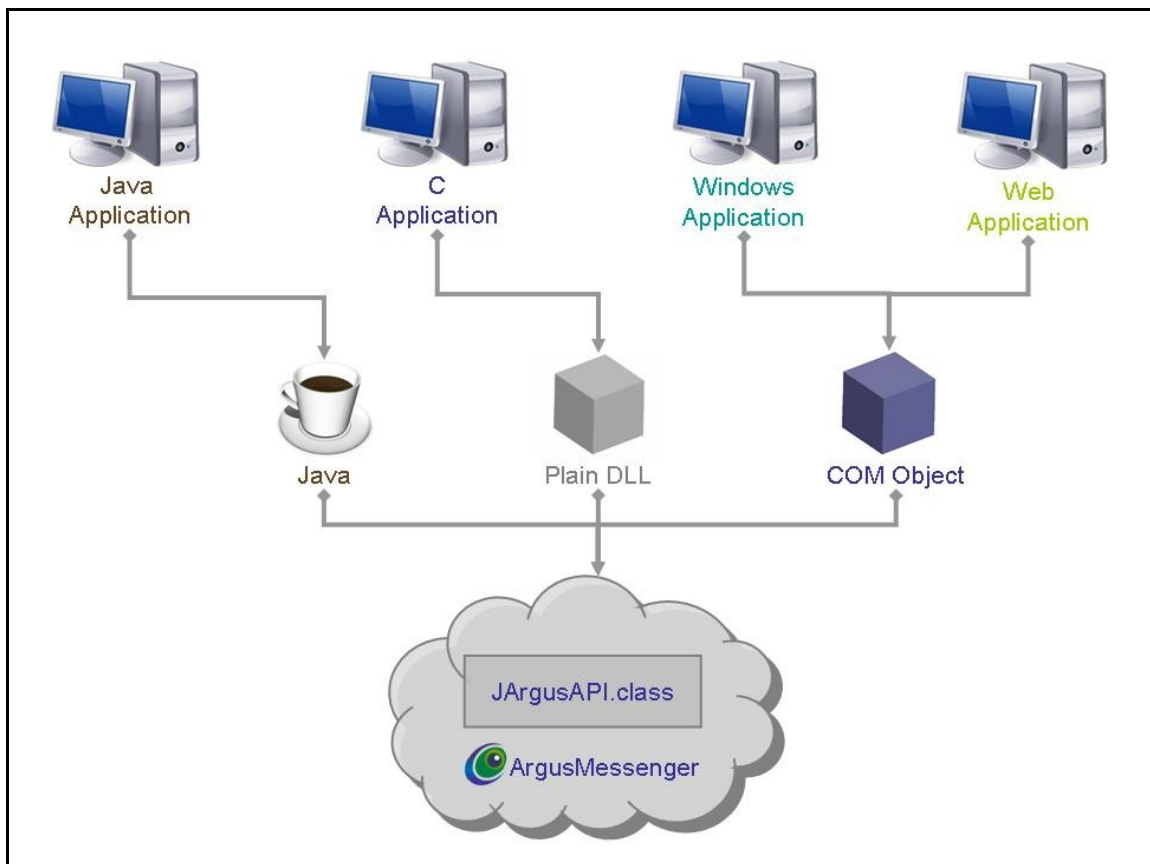
2.2.2 Argus API Versions

The current version of the API is called `ArgusAPI_V4_1.dll`. The COM objects which interface to the Argus API in `ArgusAPI.dll` and `ArgusAPI_V3.dll` are also registered to ensure backward compatibility with pre-4.1 versions of Argus.

Note: Medical Director calls the old Argus API ("ArgusAPI.dll") in order to submit requests for pathology tests, and via `\MDW2\Diabetes Transmission Utility.exe` for submitting diabetes reports to Cardiab sites. "ArgusAPI.dll" does NOT execute the functions `sendMail`, `synchronise` and `refreshAddressBook`. These functions simply return the string '<result>OK</result>' as if the function has run correctly, when in fact, it has not actually executed any Argus call.

2.3 How do I use the Argus API?

The Argus API can be called in a number of different ways – via COM, DLL, or direct Java call. At this time no .so library file has been compiled for use under Linux/Mac.



API Architecture

2.3.1 Calling the Argus API via COM

The Component Object Model (COM) is a component software architecture that allows applications to be built from components supplied by different software vendors. COM is the underlying architecture that forms the foundation for higher-level software services such as inter-application scripting and other software interactions.

The Argus API is best referenced under Windows via the COM object. Before calling the COM object which interfaces to the Argus API, the COM object must be registered on the machine on which it will be used. By default, the standard Argus install and the Argus Client Setup program will perform this automatically.

2.3.2 Calling the Argus API via DLL

A Dynamic Link Library (DLL) is an executable file that acts as a shared library of functions. Dynamic linking provides a way for an application to call a function that is not part of its executable code. The DLL contains one or more functions that are compiled, linked, and stored separately from the application that uses them.

Any DLL functions that will be called must be declared exactly as they are defined in the Argus API DLL as function headers. These function headers are provided in Appendix A.

2.3.3 Calling the Argus API via Java

The Argus API can also be accessed directly via Java. This method of calling the Argus API is limited to those who have a native Java application from which they wish to call the Argus API. All that is required is to import the `ArgusServer.jar` file as an external library into your Java application.

2.4 Argus API Functions

The Argus API contains a number of common functions which are most likely to be called by those interfacing with the API. These functions relate to the step-by-step process of sending a new message via Argus.

2.4.1 Initial / Required Functions

In order to call most of the functions in the API, you must first successfully connect to both the API and the Argus database, using the following 2 function calls.

TASK	API FUNCTION(S)
Connect to the API	connectToAPI : this function establishes a connection with the API. This function must be called before any other functions.
Optional : Enable debugging	activateTracing , sendTraceToFile : enables the Argus API to print detailed debugging information to the file defined by calling sendTraceToFile .
Connect to the Argus database	connectUserToDatabase : after establishing a connection to the API the next step is to log into the Argus database. None of the following functions will work as expected unless a connection has been made to the Argus database.

2.4.2 Frequently Used Functions

Along with the initial functions listed previously, the Argus API contains a number of common functions which are most likely to be called by those interfacing with the API. These functions relate to the step-by-step process of sending a new message via Argus.

TASK	API FUNCTION(S)
Retrieve error string from error code	getErrorString : most functions in the Argus API return an error code. To retrieve a human-readable string representation of the error code call this function, passing through the error code.
Retrieve Argus address book information	listAddressBookEntries , getAddressBookEntry : these functions may be used to retrieve address information from the Argus Address Book. This includes entries downloaded from the Argus Users Directory.
Retrieve Argus user information	listArgusUsers : this functions allows programmers to retrieve Argus users. When adding a message to Argus via the API, the email address of the sender must match that of an Argus user.

2.4.3 Adding a Message

The following functions relate to the actual 'sending' of a message via Argus. Please note that these methods do not actually send the message; it is simply created and stored in the Argus database. If the message was added to the 'Outbox' folder (whose folder key is -7), the message will be sent the next time Argus performs a mail synchronisation.

ATTACHMENT DATA FORMAT	API FUNCTION(S)
Plain message – no attachment, or non-HL7 attachment	addMessage : adds a plain (non-HL7) message to Argus. Messages added to Argus using any of the <code>addMessage</code> functions will not have guaranteed delivery, unless the attachment is marked as a HL7 message.
HL7 file	addHL7Message : creates a new message to be sent via Argus with the HL7 file as an attachment. Messages added using this function will have guaranteed delivery, as the attachment is a HL7 message.
HL7 string	addHL7MessageStream : creates a new message to be sent via Argus with the HL7 string as an attachment. Messages added using this function will have guaranteed delivery, as the attachment is a HL7 message.
Any other attachment data format (RTF, PIT, etc)	addWrappedDocuments : creates an HL7 message containing embedded blocks of formatted text, such as PIT or RTF or PDF (depending on files passed to function). Messages added using this function will have guaranteed delivery, as the attachment is a HL7 message.

2.4.4 Complete Function List

For Java/C developers, the complete list of all of the functions of the Argus API is documented in `JArgusAPI.html`, contained in the 'Documentation' directory of the Development Kit install. This document also contains detailed function/parameter descriptions.

For Windows developers intending to call the Argus API via COM or DLL, the function headers of all API functions can be found in Appendix A.

2.5 Code Examples

The Argus API and ArgusWord Development Kit contains example applications written in a number of different programming languages, which are designed to show how to call some of the most common API functions. Code snippets from one of these applications is also shown below.

2.5.1 Calling Common Functions

The following code demonstrates how the Argus API can be used to add an RTF document and a JPEG image to a message. The code also shows the `connectToAPI` and `connectUserToDatabase` functions that must be called before the `addWrappedDocuments` function. This function can take multiple files and create a HL7 message which is then sent by Argus. The full code which these snippets are based on can be found in the Argus install under `/Server/Argus API/Calling Argus via COM/`, as well as other examples using Delphi and Visual Basic.

```
foArgus: variant;

foArgus:= CreateOleObject('ArgusAPI_4_2.APIWrapper');

if VarIsEmpty( foArgus) then begin
    showMessage( 'Unable to initialise COM object' );
    exit;
end;

if foArgus.JVMRunning then begin
    showMessage( 'JVM already running' );
    exit;
end;

iErrorCode:= foArgus.connectToAPI;

case iErrorCode of
    0 : showMessage( 'API Connection established (Via COM)' );
    -2 : showMessage( 'API Error: Thread detached from the VM.' );
    -3 : showMessage( 'API Error: JNI version error.' );
    -4 : showMessage( 'API Error: Not enough memory.' );
    -5 : showMessage( 'API Error: VM already created.' );
    -6 : showMessage( 'API Error: Invalid arguments.' );
    -101 : showMessage( 'API Error: Local error for not finding the
DLL.' );
```

Initializing the COM object and connecting to the API

```
foArgus.activateTracing;
foArgus.sendTraceToFile('C:\ArgusTraceFile.txt');
```

Enabling debugging

```
if foArgus.isConnected then begin
    showMessage( 'Already connected to database' );
    exit;
end;

if foArgus.connectUserToDatabase('localhost',
                                'C:\ArgusInstall\Database\Argus.gdb',
                                'username',
                                'password') = 0
then
    showMessage( 'Connected to Database' );
else
    showMessage( 'Unable to connect to database' );
```

Connecting to the Argus database

```

if foArgus.addWrappedDocuments(
    'joebloggs@demoradiology.com.au',
    'Re: Kelly Slater - Report and scan',
    'Please find report and scan attached to this message',
    'bill.smith@democlinic.com.au,bob.black@democlinic.com.au','',',',
    'C:\My Reports\SLATER(18 June 2006).rtf*c:\SLATERscan01.jpg',
    'Letters and Reports,JPG Document',
    false, false,
    3, 86400, -7, true, '')= 0
then
    showMessage( 'Message added to Argus OK' );
else
    showMessage( 'Unable to add message to Argus' );

```

Adding a message to Argus

2.5.2 Other Sample code/applications

The following sample applications are included in the `\Server\Argus API\` directory of your Argus install.

Calling Argus via COM

This is a demonstration program written in Delphi that shows how to initialise and make a simple call to the API, via COM. It caters for functions in Version 3 - 4.0.x (`ArgusAPI_V3.dll`) and version 4.1 (`ArgusAPI_V4_1.dll`).

Calling Argus via DLL

The API has also been compiled into a standard (non-COM) DLL. This is a demonstration program written in Delphi that shows how to initialise and make a simple call to the API, either by calling a DLL, or by linking the code directly into the EXE (see compiler directive `DO_NOT_USE_DLL`). It caters for functions in version 4.1 (`ArgusAPI_V4_1.dll`) only.

2.6 Troubleshooting

The following table lists some of the most common problems encountered by developers when using the Argus API, and possible causes & solutions.

PROBLEM	SOLUTION(S)
I cannot call <code>connectToAPI</code> successfully	If using COM, make sure the COM object has been registered. (Check the Windows registry: <code>HKEY_CLASSES_ROOT</code> , key: <code>ArgusAPI_4_2.apiwrapper</code>).
I cannot call <code>connectToAPIWithJVM</code> successfully	Ensure the JVM uses Java 1.5 or greater (required by Argus version 4.2 or greater).
After calling <code>connectToAPI</code> successfully, I am unable to call the <code>connectUserToDatabase</code> function or any other 'connect X to database' function.	After ensuring that login details are correct (by attempting to login to ArgusMessenger for example), make sure that firewall software such as ZoneAlarm is not blocking the database connection.

--- fin ---

Andrew Shrosbree, Rachel Naus
August 2007

Appendix A: COM Object Function Calls

2.1 Introduction

This document lists all the functions that can be called in version 4.1 of the Argus API (ArgusAPI_V4_2.dll), when accessed through a COM interface. Since the code was written in Delphi, Pascal syntax is used.

2.2 Functions for connecting to Argus

function connectToAPI: integer;

This is the first function to call. It initializes the COM object and establishes a Java 'bridge' between the calling application and Argus.

function connectToAPIWithJVM(const psArgusJARs, psJVMLocation: string): integer;

Deprecated.

function JVMRunning: boolean;

Allows a program to determine whether connectToAPI has already been called, because the JVM may remain loaded when execution of a program that links to the API has previously terminated.

2.3 Functions for logging into Argus database

function connectToDatabase(const psHost, psDatabase, psPassword: string): integer;

Calls connectAdminToDatabase.

function connectAdminToDatabase(const psHost, psDatabase, psPassword: string): integer;

Logs into Argus as if running ArgusMessenger. i.e. it connects as SYSDBA, with role 'argusAdmin'.

function connectUserToDatabase(const psHost, psDatabase, psUser, psPassword: string): integer;

Logs into Argus as if running ArgusMail. i.e. it connects as psUser, with role 'argusUser'.

function isConnected: boolean;

Allows a program to determine whether connectAdminToDatabase or connectUserToDatabase have already been called, because the database connection may remain when execution of a program that links to the API has previously terminated.

2.4 Functions for producing debug information

procedure activateTracing;

Causes Argus to direct debug information to the console.

```
procedure sendTraceToFile(const psDirectory: WideString);
```

Causes debug information being directed to the console to be saved in the file 'trace.txt' in the folder *psDirectory*.

2.5 Functions for dropping messages into the ArgusMail outbox

```
function addMessage(const psSender, psSubject, psBody, psAttachments, psRecipients,
psRecipientsCC, psRecipientsBCC: string; pbEncrypt, pbSeekKeyLocally: boolean; piResends:
integer; pbConfirmReading, pbConfirmDelivery, pblsHL7Message: boolean; piResendHL7Every:
integer): string;
```

This is generic method for dropping a standard email message into the Argus outbox, emulating the ordinary MAPI behaviour of a non-HL7 enabled email program like MS Outlook. Never send clinical messages using this function. Use `addHL7Message` instead.

```
function addHL7Message(const psSender, psSubject, psBody, psHL7File: string; pbEncrypt: boolean;
piResends, piResendHL7Every: integer; const psProviderHQL: string): string;
```

Adds HL7 messages to the Argus 'Outbox' for sending later.

```
function addHL7MessageStream(const psSender, psSubject, psBody, psRecipients, psRecipientsCC,
psRecipientsBCC, psHL7Message: string; pbEncrypt: boolean; piResends,
piResendHL7Every: integer): string;
```

Adds a message to the Argus 'Outbox' for sending later. This method is the same as `addHL7Message`, but instead of receiving the name of a file containing HL7 data, it receives the HL7 as a stream of text.

```
function addMIMEMessage(const psSender, psMIMEText: string): string;
```

This drops a MIME-encoded message into the Argus outbox. It prevents the double encryption of messages that may have been encrypted by the application that is using Argus as a transport layer. This may have relevance when individual signing becomes more widespread.

```
function addWrappedLetterOrReport(const psSender, psSubject, psBody, sRecipients,
psRecipientsCC, psRecipientsBCC, psFileName, psPatientFirstName, psPatientMiddleName,
psPatientLastName, psPatientDOB: string; pbEncrypt, pbSign: boolean; piResends,
piResendHL7Every, piFolder: Integer; const psReserved: string): string;
```

This method constructs a HL7 message containing an embedded Letter or Report inside an OBX segment. Such embedded files can be exported from messages created with this rule by using the Letters and Reports export template in ArgusMessenger.

```
function addWrappedDocuments(const psSender, psSubject, psBody, psRecipients, psRecipientsCC,
psRecipientsBCC, psFileNames, psOBXSegmentLabels: string; pbEncrypt, pbSign: boolean;
piResends, piResendHL7Every, piFolder: Integer; const psReserved: string): string;
```

This method constructs a HL7 message containing embedded blocks of formatted text, such as PIT or RTF or PDF. The OBX Segment Labels are required to allow the receiving export rule to determine the correct embedded document to export.

```
function addArgusWordDocument(const psSenderName, psSenderEmail, psSubject, psBody,
psPatientTitle, psPatientFirstName, psPatientLastName, psPatientDOB, psReference,
psDocumentTitle, psReportDate, psRecipientEmails, psRecipientNames, psAttachments,
psSegmentLabels: string; pbEncrypt, pbSign: boolean; piResends, piResendHL7Every, piFolder:
Integer; const psReserved: string): string;
```

This method constructs a HL7 message containing files embedded inside OBX segments. It is used by the macros in ArgusWord to embed an RTF document, and any attachments the ArgusWord user adds to the message. Extra fields allow patient information to be added, making it easier to construct proper HL7 and PIT messages at the recipient end. The psSegmentLabels are required to allow the receiving export rule to determine the correct embedded document to export.

2.6 Functions for initiating sending and receiving of messages

```
function sendMail: string;
```

Initiates the sending of messages currently in the outbox.

```
function synchronise: string;
```

Initiates the sending of messages currently in the outbox and the downloading of messages from the mail server.

2.7 Functions for retrieving messages from the Argus database

```
function listMessages(const psOwnerEmail, psDirectory, psDownloadType: string; piFolderKey:
integer; const psDateFrom, psDateTo, psDateType, psFields: string): string;
```

Messages are stored locally by Argus in an Interbase database. This function returns a subset of these messages as XML, which can be loaded into a GUI window as the programmer pleases. This allows the creation of a message viewer as an alternative to ArgusMail. The function listMessages must be used when retrieving messages from a folder in a date range.

If user wishes to load either for a date range or for a folder, use listMessagesInFolder() or listMessagesForDate() instead. psDateFrom and psDateTo should be left blank to retrieve messages created but not sent.

```
function listMessagesInFolder(const psOwnerEmail, psDirectory, psDownloadType: string;
piFolderKey: integer; const psFields: string): string;
```

If Argus has, for example been configured to filter all discharge summaries into a particular Argus folder upon arrival, a calling program can retrieve only those (filtered) messages as XML.

```
function listMessagesForDate(const psOwnerEmail, psDirectory, psDownloadType, psDateFrom,
psDateTo, psDateType, psFields, psReserved: string): string;
```

Fetches a list of messages as XML in a date range.

```
function getMessage(piMessageKey: integer; const psDirectory, psFields: string): string;
```

Fetches a single message, placing attachments into a designated directory

```
function getHL7Attachment(piKey: integer; const psAttachment: string): string;
```

Fetch the HL7 content of the attachment to a single message.

```
function listFolders(const psOwnerEmail: string): string;
```

Retrieve a list of Argus folders. This helps the programmer to determine the primary key of the appropriate folder for use as a parameter to methods: listMessages and listMessagesInFolder.

2.8 Functions for address book handling

```
function refreshAddressBook: string;
```

Initiates the service that causes ArgusMessenger to refresh its local Address Book from centralized LDAP servers such as the Argus Users Directory.

```
function updateFromArgusUsersDirectory: string;
```

Initiates the service that causes ArgusMessenger to refresh its local Address Book from the Argus Users Directory.

```
function updateLocalAddressBook: string;
```

This function updates the local address book with entries from either/both LDAP Servers which have been configured to auto update.

```
function listAddresses(const psOwnerEmail, sReserved: string): string;
```

Retrieve a list of Addresses from the Address Book

```
function getAddressBookEntry(piAddressKey: Integer; const psFields: string; pbIncludeDTD: boolean; const psReserved: string): string;
```

Retrieves an Address from the Address Book according to the key of the address book entry passed through to the method.

```
function hasAddressBookEntry(const psEmailAddress, psProviderNumber, psReserved: string): string;
```

Determines whether a recipient is in the local Address Book.

```
function listAddressBookEntries(const psAddressType, psFields: string;
pbIncludeDTD: boolean; const psReserved: string): string;
```

Retrieve a list of Addresses from the Address Book using the fields passed through to the method.

2.9 Miscellaneous functions

```
function deleteMessage(piKey: integer): string;
```

Deletes a message from the Argus database. All add[] methods return an integer key, which can be used by deleteMessage to remove the message from the outbox. Obviously, this can only be done if ArgusServer has not already sent the message.

```
function messageCount(const psOwnerEmail, psRecipientEmail: string; piKey: integer; const
psMessageType: string): string;
```

Calculates how many messages exist. Parameters equal to "" or 0 are ignored. All fields are optional, if all are either "", null, or 0, entire message count is returned.

```
function moveMessageToFolder(piMessageKey, piFolderKey: integer): string;
```

Move a message to a specified folder. The email must exist relating to a person registered with Argus, the message key must be valid and the folder key must be valid and relate to the specified person.

```
function getErrorString(piError: integer): string;
```

Returns a descriptive String for the given error constant.

```
function listArgusUsers(const psReserved: string): string;
```

Retrieves a list of the accounts that have been set up in ArgusMessenger (one per email account).

```
function listFilterRules(const psUserEmail, psReserved: string): string;
```

Retrieves a list of the Argus rules that filter incoming messages into specific Argus folders. This helps the programmer to determine the primary key of the appropriate folder, for use as a parameter to methods: listMessages and listMessagesInFolder.

```
function getFilterRuleFolder(piKey: Integer; const psReserved: string): string;
```

Some rules may be created for filtering incoming messages into particular folders. This function retrieves the primary key for such a folder by using the primary key of the rule with which it is associated. Refer to listFilterRules.