Quest® Archive Manager 5.11

SDK Reference Guide

© 2026 Quest Software Inc.

ALL RIGHTS RESERVED.

This guide contains proprietary information protected by copyright. The software described in this guide is furnished under a software license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of the applicable agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Quest Software Inc.

The information in this document is provided in connection with Quest Software products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest Software products. EXCEPT AS SET FORTH IN THE TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST SOFTWARE ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST SOFTWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest Software makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest Software does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

Quest Software Inc.
Attn: LEGAL Dept.
20 Enterprise, Suite 100,
Aliso Viejo, CA 92656

Refer to our website (www.quest.com) for regional and international office information.

Patents

Quest Software is proud of our advanced technology. Patents and pending patents may apply to this product. For the most current information about applicable patents for this product, please visit our website at www.quest.com/legal.

Trademarks

Quest Software, Quest, and the Quest logo are trademarks and registered trademarks of Quest Software Inc. in the U.S.A. and other countries. For a complete list of Quest Software trademarks, please visit our website at www.quest.com/legal. All other trademarks, servicemarks, registered trademarks, and registered servicemarks are the property of their respective owners.

Legend

CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.

IMPORTANT, **NOTE**, **TIP**, **MOBILE**, or **VIDEO:** An information icon indicates supporting information.

Archive Manager SDK Reference Guide
Updated - February 2026
Software Version - 5.11

Contents

SDK reference **4**
Overview 4
Namespaces 4
SDK usage 4
SDK collections 6
Message creation 7
Attachment creation 7
Embedded messages 8
Extended properties 8

About us **9**

1

SDK reference

- Overview
- Namespaces
- SDK usage
- SDK collections
- Message creation
- Attachment creation
- Embedded messages
- Extended properties

Overview

The Archive Manager SDK allows 3rd party application developers to interface with the archive for creation and retrieval of content and information. The Archive Manager SDK provides a high-level .NET class library that communicates with the Archive Manager RESTful Web Service API. SDK code documentation can be found in the **Quest.AM.Public.Documentation.chm** file.

Namespaces

There are three general SDK namespaces: Interfaces, Models, and Objects. Models and Objects implement the interfaces within the Interfaces namespace. Each class in the Objects namespace derives from a class in the Models namespace. Models are the base data classes that contain the properties that define the data type you are accessing. Objects are active types that have navigation, collections, and special properties that further describe relationships of a given Object to other Objects. Think of this as a three layer system where Interfaces are the Model and Object definitions, Models are the data storage, and Objects are the Model access.

The types and interfaces included in **Quest.AM.Public** and **Quest.AM.Public.Interfaces** are shared between the internal system and the SDK. These types are used when creating new messages or accessing certain message properties.

SDK usage

The primary Object is a class implementation of the **ISDKInstance** interface called **ArchiveManager**. The ArchiveManager class describes the Archive Manager RESTful Web Service API endpoint to which you are talking. The constructor takes a base URL and application name.

```
ISDKInstance archiveManager = new ArchiveManager("http://archivemanager/",
     "Archive Manager Example Application");
ISDKLogin login = archiveManager.Login("Admin", "Password", "DEFAULT", false);
```

```
ISDKMailbox myMailbox = login.PrimaryMailbox;
ISDKMailboxCollection myMailboxes = login.Mailboxes;
```

Login must be called to authenticate to the endpoint. It will return the **ISDKLogin** Object for the authenticated user f successful. The **ISDKLogin** Object contains a property for the login's primary mailbox and a collection of mailboxes the login has delegated to it. The last argument changes the authentication mode to either Basic or Windows authentication. If the API endpoint is configured for Anonymous authentication, this argument should be set to **false**. If the API endpoint is configured for Windows authentication, this argument should be set to **true**. Passing a **null** username causes the application to authenticate as the user executing the application.

There is a collection Object for each of the primary types. In this example, **ISDKMailboxCollection** myMailboxes contains an enumerable property called "Items" that can be used to access each of the individual **ISDKMailbox** Objects within the collection.

```
foreach (ISDKMailbox mailbox in myMailboxes.Items)
{
      String mailboxName = mailbox.Name;
}
```

The SDK consists of many relationships branching from the **ArchiveManager** Object. All of the SDK Objects start with SDK followed by the Model for which they provide navigation. For example, **SDKMailbox** is the access Object for the **Mailbox** Model. The **Mailbox** Model contains the property data related to the mailbox, but cannot navigate to mailbox folders or mailbox messages. The **SDKMailbox** Object contains relationship access properties to allow navigation to an **ISDKFoldersCollection** Object or **ISDKMessagesCollection** Object.

Each **ISDKCollection** provides a **GetCount()** method that can be used to retrieve the count of the collection without a full enumeration. Using Linq extensions such as **Count()** or **Where()** on the **Items** enumerable can cause performance issues because they operate on the entire collection, causing the yielded self-filling enumerable to be fully retrieved and processed locally. Using the ISDKCollection.GetCount() method performs a SQL query to calculate the number of records, and is far less expensive than retrieving all records.

```
ISDKMailbox testMailbox =
   archiveManager.Mailboxes.Items.FirstOrDefault(mb => mb.Name == "TestMailbox");
ISDKFolderCollection folders = testMailbox.Folders;

foreach (ISDKFolder folder in folders.Items)
{
   ISDKMessageCollection messages = folder.Messages;
   foreach (ISDKMessageSummary messageSummary in messages.Items)
   {
      ISDKMessage fullMessage = messageSummary.Email;
      String subject = fullMessage.Subject;
      IList<IAttachment2> attachments = fullMessage.Attachments;
      IList<IEmailAddress2> to = fullMessage.To;
   }
}
```

Any time an Object property is used, the SDK reissues the request for the data. In most cases, you will want to store the data you wish to access in its own variable to reduce sub requests while using Linq. For example, the following code would cause a new lookup for **MyLogin** for each item in the Mailboxes collection and would cause extra bandwidth to be used.

```
}
ISDKMailbox testMailbox =
   archiveManager.Mailboxes.Items.FirstOrDefault(mb => mb.OwnerLoginId ==
archiveManager.MyLogin.Id);
```

Each time **archiveManager.MyLogin.Id** is evaluated, a request for **MyLogin** will be executed first. The more efficient way to perform this Linq lookup is to store **MyLogin** in an **ISDKLogin** Object. This way the evaluation of the **ISDKLogin** Id property does not cause a new request to lookup **MyLogin**.

```
ISDKLogin myLogin = archiveManager.MyLogin;
ISDKMailbox testMailbox =
   archiveManager.Mailboxes.Items.FirstOrDefault(mb => mb.OwnerLoginId == myLogin.Id);
```

The SDK Objects contain methods to **Refresh** or **Save**. This allows you to make changes to an object and save or revert changes by calling **Save** or **Refresh**. Certain properties cannot be change and calling **Save** may throw an exception.

```
//User1
String displayName = myLogin.DisplayName;
myLogin.DisplayName = "New Display Name";
myLogin.Refresh();
//User1
displayName = myLogin.DisplayName;
myLogin.DisplayName = "Another New Display Name";
myLogin.Save();
//Another New Display Name
displayName = myLogin.DisplayName;
```

SDK collections

SDKCollection Objects contain methods to **Add**, **AddByRef**, **RemoveByRef**, and **New**. The **Add** method allows you to create a new Object within the collection. In this case, the **Add** method only cares about the model data so it would require a model interface.

```
ISDKFolderCollection folders = testMailbox.Folders;
IFolder newFolder = new Folder();
newFolder.MailBoxId = testMailbox.Id;
newFolder.Name = "FolderName";
newFolder.Visible = true;
folders.Add(newFolder);
```

Using the **AddByRef** and **RemoveByRef** allows you to link and unlink by object ID. Each type will have an associated ID property that can be used to link into a given type collection. For example, you could add a folder to the Subfolders collection of another folder to build hierarchy.

```
ISDKFolderCollection folders = testMailbox.Folders;
ISDKFolder testFolder = folders.Items.FirstOrDefault(f => f.Name == "TestFolder");
ISDKFolder testFolder2 = folders.Items.FirstOrDefault(f => f.Name == "TestFolder2");
testFolder.Subfolders.AddRef(testFolder2.Id);
```

The **New** method returns an empty Model for convenience. Once you “**Add**” the Model to a collection, the **Add** method will return an SDK version ready for navigation. The ID field and other relevant links are populated.

```
ISDKFolderCollection folders = testMailbox.Folders;
IFolder newFolder = new Folder();
newFolder.MailBoxId = testMailbox.Id;
newFolder.Name = "FolderName";
newFolder.Visible = true;
ISDKFolder newSDKFolder = folders.Add(newFolder);


int newFolderId = newSDKFolder.Id;
//ownerMailboxId == testMailbox.Id
int ownerMailboxId = newSDKFolder.MailBoxId;
//0 for top level folder
int parentFolderId = newSDKFolder.ParentFolderId;
```

Message creation

Messages are constructed using the **Message** and **Attachment** Models. These Models implement interfaces found in the **Quest.AM.Public.Interfaces** and require some types from **Quest.AM.Public**. Messages and attachments have many properties. Most are metadata that is not required. The following is an example of populating the general **Message** fields.

```
IMessage message = new Message();
message.Attachments = new List<IAttachment2>();
message.From = new EmailAddress("DisplayName", "Email@Address.com");
message.To = new List<IEmailAddress2>();
message.To.Add(new EmailAddress("DisplayName", "Email@Address.com"));
message.CC = new List<IEmailAddress2>();
message.BCC = new List<IEmailAddress2>();
message.MessageClass = "IPM.Note";
message.BodyCodePage = 65001;
message.BodyFormat = BodyFormat.Text;
message.DateSent = DateTime.MinValue;
message.DateReceived = DateTime.MaxValue;
//Internet Message-ID as defined by RFC5322 section 2.2
message.Header = "2C257C1D6C0466419D1E46B7EB0EE3120FEB2702@ESCMail.esc.quest.local";
//Low = 0, Normal = 1, High = 2
message.Importance = 1;
//None = 0, Personal = 1, Private = 2, CompanyConfidential = 3
message.Sensitivity = 0;
//Message = 1, Calendar = 2, DeliveryStatusNotification = 3, Contact = 4
message.MessageTypeID = 1;
//Internet Message-ID as defined by RFC5322 section 3.6.4
message.OrigionalMessageID = "InternetMessageID";
//Important when embeding email as attachments, always set to Email;
message.PartType = EmailPartType.Email;
//Total number of bytes
message.Size = 500;
message.Subject = "Message Subject";
//ConversationIndex as defined by Microsoft E-Mail Object Protocol Specification
//[MS-OXOMSG] PidTagConversationIndex Base64 encoded value,
//otherwise RFC2822 section 3.6.4 describes In-Reply-To and References headers
message.ThreadIndex = "AQHNjQWAdEBzhLInT0yR5EYT63xljA==";
//ConversationTopic as defined by Microsoft E-Mail Object Protocol Specification
//[MS-OXOMSG] PidTagConversationTopic, typically the message original subject
message.ThreadTopic = "Message Subject";
```

Body elements are provided as a **Stream** type using the **StreamAsIDataStream2** wrapper class. The Constructor contains a few fields that are for internal use only at the moment. The important fields will be the stream with the content, the **DataStreamPartType**, and the stream length.

```
MemoryStream bodyStream = new MemoryStream();
message.Body = new StreamAsIDataStream2(DataStreamPartType.Body, null, Guid.Empty,
false, bodyStream, false, 0, bodyStream.Length);
```

Attachment creation

Attachments are added to the message's **Attachments** collection. An **Attachment** contains many meta-properties used to describe the attachment, but not all are required.

```
Attachment attachment = new Attachment();
attachment.Name = "Attachment";
```

```
attachment.FileName = "Attachment.docx";
attachment.ContentType = "application";
attachment.ContentSubType = "docx";
attachment.ContentDescription = "Attachment Description";
attachment.PartType = EmailPartType.Attachment;
```

The attachment stream is also uses the **StreamAsIDataStream2** wrapper class. The property **StreamType** needs to be set based on the type of attachment it is. In this case, **ByValue** for an actual attachment file.

```
//ByValue = 2, EmbeddedMsg = 6, OLE = 7
attachment.StreamType = 2;

MemoryStream attachmentData = new MemoryStream();
attachment.Size = attachmentData.Length;
attachment.Data = new StreamAsIDataStream2(DataStreamPartType.AttachData, null,
Guid.Empty, false, attachmentData, false, 0, attachmentData.Length);
```

Embedded messages

Attaching messages as embedded messages can be done by assigning the message object to the **Data** property of the attachment. Setting the **StreamType** to 6 for EmbeddedMsg will allow the parser to properly handle the embedded message.

```
IMessage message = new Message();
//ByValue = 2, EmbeddedMsg = 6, OLE = 7
attachment.StreamType = 6;
attachment.Size = message.Size;
attachment.Data = message;
```

Extended properties

Certain message types may contain extended properties and can be added to the message **SourceProperties** property. For example, we can add the property “Task.StartTime” to the message. These extended properties require special formatting and encoding. A method was provided at the instance level to attach the properties to a message.

```
IMessage message = new Message();
MessageSourceProperties messageProperties = new MessageSourceProperties();
messageProperties.Add(new MessageSourceProperty("Task.StartTime", DateTime.Now,
SourcePropDataType.DateTime));
archiveManager.EncodeSourceProperties(messageProperties, message);
```

About us

Quest provides software solutions for the rapidly-changing world of enterprise IT. We help simplify the challenges caused by data explosion, cloud expansion, hybrid datacenters, security threats, and regulatory requirements. We are a global provider to 130,000 companies across 100 countries, including 95% of the Fortune 500 and 90% of the Global 1000. Since 1987, we have built a portfolio of solutions that now includes database management, data protection, identity and access management, Microsoft platform management, and unified endpoint management. With Quest, organizations spend less time on IT administration and more time on business innovation. For more information, visit www.quest.com.

Technical support resources

Technical support is available to Quest customers with a valid maintenance contract and customers who have trial versions. You can access the Quest Support Portal at https://support.quest.com.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request.
- View Knowledge Base articles.
- Sign up for product notifications.
- Download software and technical documentation.
- View how-to-videos.
- Engage in community discussions.
- Chat with support engineers online.
- View services to assist you with your product.