

# RMFT Web Service Guide

Software Version 2.4.2

August 16, 2009

August 16, 2009

Copyright © 2000-2009 by RepliWeb, Inc.

The information in this manual has been compiled with care, but RepliWeb makes no warranties as to accurateness or completeness, as the software described herein may be changed or enhanced from time to time. This information does not constitute commitments or representations by RepliWeb, and is subject to change without notice. The software described in this document is furnished under license and may be used or copied only in accordance with the terms of this license.

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written consent of RepliWeb, Inc. Any trademarks, trade names, service marks, or service names owned or registered by any other company and used in this manual are proprietary to that company.

Please direct correspondence or inquiries to:

RepliWeb, Inc.  
6441 Lyons Road  
Coconut Creek  
FL 33073

Phone: (954) 946-2274

Fax: (954) 337-6424

E-Mail: [info@repliweb.com](mailto:info@repliweb.com)

Support: <http://support.repliweb.com/>

Web Site: <http://www.repliweb.com/>

# Table of Contents

<b>Overview .....</b>	<b>6</b>
Architecture .....	6
Web Services Access .....	7
Compatibility.....	8
Developing Client Applications .....	9
Visual Studio 2005 .....	9
Eclipse .....	9
<b>1. SessionManager Object.....</b>	<b>10</b>
Types .....	12
Credentials .....	12
Session.....	12
Methods .....	13
AdminLogin.....	13
UserLogin .....	15
KeepAlive .....	17
Logout.....	18
Login Samples .....	19
C# .....	19
Java .....	20
IIS Sample .....	22
<b>2. VersionManager Object .....</b>	<b>24</b>
Overview .....	24
Methods .....	24
GetServiceInfo.....	24
GetComponentInfo .....	25
<b>3. ConfigManager Object .....</b>	<b>26</b>
Overview .....	26
Class Hierarchy.....	26

Class-Object Relationships ..... 27

Common Types..... 27

Configuring Users ..... 29

    User Object..... 29

User-Related Types ..... 37

    Common Types ..... 37

    IncomingPackageOptions ..... 38

    SendRestrictions and ReceiveRestrictions ..... 38

    UserCapabilities ..... 39

    UserDetailLevel ..... 39

    UserFilter ..... 40

    UserEnrollmentState ..... 40

    UserPrivileges ..... 41

    UserReadOnlyAttributes..... 42

Methods ..... 43

    Create..... 43

    Delete ..... 46

    FindEntity..... 47

    Get..... 49

    Modify ..... 51

    List ..... 53

    StartList ..... 57

    GetNextRows ..... 60

    EndList..... 61

Containers..... 62

    Group Object..... 63

    Group Related Types ..... 66

    DistributionList Object ..... 66

ContainerEntity Methods..... 67

    AddMemberToContainer ..... 67

    RemoveMemberFromContainer..... 69

Inspection Policies ..... 70

    InspectionPolicy Object Overview ..... 70

    Object Definition ..... 70

    InspectionPolicy Related Types ..... 73

InspectionPolicy Methods ..... 74

    VerifyInspectionPolicy ..... 74

    VerifyInspectionPoliciesForEntity ..... 75

    ListEntitiesByInspectionPolicy..... 76

Sample (C#) .....	77
Sample (Java) .....	77
Notifications.....	79
Notification-Related Types .....	80
NotificationContact.....	81
EmailTemplate .....	81
EmailTemplate - Related Types .....	82
CreationTemplate.....	84
Sample (C#) .....	84
<b>4. PackageManager Object.....</b>	<b>86</b>
Overview .....	86
Types .....	87
Package Object .....	87
PackageOptions .....	87
PackageNotificationsOptions .....	88
Methods .....	89
Abort.....	89
Create.....	90
Submit .....	91
<b>5. AddressBookManager Object .....</b>	<b>92</b>
Overview .....	92
AddressBook Types.....	92
Methods .....	96
CheckRecipients .....	96
ListAddressBook .....	100
StartListAddressBook.....	103
Address Book Samples .....	105
GetNextRows .....	107
EndList.....	108
<b>6. TransferManager Object .....</b>	<b>109</b>
Overview .....	109
Methods .....	110
GetPackageFileHash .....	110
GetPackageTransferProgress.....	111

UploadPackageFile ..... 113

UploadPackageFileChunk..... 114

**A. Class Hierarchy.....118**

**B. Troubleshooting .....119**

Log Files..... 119

# Overview

RepliWeb Managed File Transfer (RMFT) is a turnkey solution for managed, enterprise-scale file exchange. RMFT greatly improves the manageability, reliability, efficiency and security of file workflows and file transfers. RMFT's user-friendly Web interface enables end-users to exchange files with ease, while fully automated server-to-server file transfer and automated file processing provide complete "hands-off" operation. By streamlining management and administration of file transfers, RMFT removes key concerns about day-to-day file transfer processes and file workflows, allowing organizations to focus valuable resources elsewhere.

RMFT Web Service enables organizations to incorporate RMFT functionality into in-house and third party products.

This document assumes that the reader is familiar with the basic Web service concepts, including SOAP, WSDL, XML, and so on.

Web Service/WSDL terms such as *request*, *response*, and others are used frequently throughout this document. Syntax and sample code are provided in both C# and Java. With basic Web Service knowledge, you can use the sample code as a foundation for other programming languages.

**Note** A good introduction to Web Service concepts can be found at:

[www.oreillynet.com/lpt/a/webservices/2002/02/12/webservicefaqs.html](http://www.oreillynet.com/lpt/a/webservices/2002/02/12/webservicefaqs.html)

## Architecture

RMFT Web Service consists of a single SOAP Web Service called *RmftService*.

The Web Service provides various 'data objects' such as User, Group and Package. They only contain the data, which is accessed as properties. These properties enable you to set and get the data as well as being useful for creating binding and data sets for UI.

To manage the data objects (create, list, delete, and so on) the Web Service provides manager objects that are implemented as WSDL PortTypes.

The manager objects are as follows:

<b>WSDL PortType</b>	<b>Purpose</b>
<code>SessionManager</code>	Logging in, authentication and logging out.
<code>ConfigManager</code>	Managing users, groups, distribution lists, hosts, notifications, and so on.
<code>PackageManager</code>	Submitting packages.
<code>AddressBookManager</code>	Listing address book items and verifying package recipients.
<code>TransferManager</code>	Uploading and package files during package submission.
<code>VersionManager</code>	Retrieving information about web service components and their version. No login is required.

All the types defined in the WSDL belong to the same namespace prefix *bh*; for example, `bh:User`, `bh:Package`, and so on. For simplicity's sake, this prefix is not shown for each type.

## Web Services Access

You can install RMFT Web Services and enable the client to access it using either of the following methods:

- 1. Direct connection (RMFT/GQS)** - Using this method, clients access the web services using predefined port 3102 on RMFT Server and RMFT runs the web services as a DLL. This is a fast connection; however it requires port 3102 to be opened in the firewall.

To use this connection in Secured mode, the RMFT administrator should configure the RMFT/GQS service (using the GQS Configuration utility) to support secured mode.

- 2. IIS** - Using this method, the Web Services DLL is installed under the IIS enabling clients to access it via the standard HTTP ports (e.g., 80 or 443). This is a slower connection, but may be more compatible with organizations' security policies. This method also enables connecting to the IIS via a proxy server.

The IIS can be accessed using a secure (SSL) or non-secure connection.

From a client perspective, the only difference between both access types is the URL the client needs to specify in order to access the server. See more details in [1. SessionManager Object](#) below.

## Compatibility

For compatibility reasons, the Web Service identifies the client version, that is, the WSDL version used to compile this client (the version is included in the WSDL file).

If the client version is higher than the Web Service version, the Web Service will deny access to the Web Service. This will prevent cases in which data sent by the client is not processed due to an older server version.

The opposite situation (new server, old client) should be handled properly.

In addition, all fields added to the new version (as well as optional fields) should be defined as optional.

There should be a clear distinction between changing a value or ignoring it. If a client does not want the server to modify a field, it should set the value of the field to NULL (or not set it at all). If, on the other hand, the client wants to empty a certain field, it should send an empty value indicator (empty string, empty array) and not NULL.

# Developing Client Applications

## Visual Studio 2005

### Prerequisites:

Install Web Services Enhancements 3.0 for Microsoft .NET (WSE).  
See <http://msdn2.microsoft.com/en-us/webservices/Aa740663.aspx>.

### To add the RMFT Web Service reference to your solution:

1. Open **Web References > Add Web reference**.
2. Enter the following URL:

<http://<server>:3102/RMFT?wsdl>

Where <server> is the RMFT Server host name. This host is only used to download the WSDL file. Once the client is built, it can access other RMFT Servers.

## Eclipse

### Prerequisites:

Your IDE should support web service (WSDL) consumption. If necessary, download the relevant plug-in. See: <http://www.eclipse.org/webtools/>

### To add the RMFT Web Service package to your project:

1. In the package explorer, right-click your project and select **New > Other > Web Services > Web Service Client**
2. Enter the following URL:

<http://<server>:3102/RMFT?wsdl>

Where <server> is the RMFT Server host name. This host is only used to download the WSDL file. Once the client is built, it can access other RMFT Servers.

# 1. SessionManager Object

The `SessionManager` object establishes connections between a client application and an RMFT Server using the given credentials.

A session is returned by the various Login methods and should be passed as a parameter to all other methods.

A client can connect to multiple RMFT Servers simultaneously and perform tasks on each of the Servers. The client should maintain several session objects (one for each RMFT Server) and use the session according to the server to which it is currently connected.

There are two session types:

1. **Admin Session** - Enables RMFT management (configuration and monitoring) tasks such as adding users and monitoring packages. This session is created by the `AdminLogin` method.
2. **User Session** - Enables users to send packages, view their inbox, and perform other user oriented tasks. This session is created by the `UserLogin` method.

Both login types can be performed using two different methods, specified by the enum type `LoginOptions`:

1. **Credentials** – The client should specify the user name, password and optional domain.
2. **NTLM** – The current user logged-in to Windows is used for the login, using NTLM protocol.

The client should set the following properties of the `SessionManager` object:

- Set the URL property to the RMFT server address, for example:

```
SessionMgr.Url = "http://rmfthost:3102/RMFT";
```

The URL defines how the Web Services will be accessed. For details about the available access types, see [Web Services Access](#) above.

The following table specifies which URL to use for each of the available connection types:

Connection Type	Secured	URL
Direct connection	No	http://<server>:3102/RMFT
Direct connection	Yes	https://<server>:3102/RMFT
IIS	No	http://<server>/repliweb-mft/rw_soapd.dll?RMFT
IIS	Yes	https://<server>/repliweb-mft/rw_soapd.dll?RMFT

- For NTLM login, set the Credentials property as follows:

```
SessionMgr.Credentials = CredentialCache.DefaultNetworkCredentials;
```

- For IIS access, if a proxy is used, set the Proxy property to a `WebProxy` object. For example:

```
SessionMgr.Proxy = new WebProxy("http://myproxy:8080");
```

For details on setting the proxy, please refer to the `WebProxy` class in .NET documentation.

**Note** The URL and Proxy (if used) properties should be set in every manager object that you create i.e. `SessionManager`, `ConfigManager`, `PackageManager`, and so on.

## Types

### Credentials

#### Properties

Property	Description
Server	The RMFT Server to which the client should connect.
Username	Windows user name or virtual RMFT user name.
Password	Windows password or virtual RMFT password. The password is transmitted to the Web Service in plain text. For encryption, the Web Service should be secured by SSL.
Domain	For Windows users only.

### Session

This type is created by `AdminLogin()` or `UserLogin()`. It represents an administrator or user session. The session's properties are read-only and should never be modified by clients.

#### Properties

Property	Description
Token	Identifies a Web Service session.
RmftUserName	The virtual RMFT user name that was assigned to this session. This property is provided by <code>UserLogin()</code> . When calling <code>UserLogin()</code> with NTLM, the Web Service transforms the user name and domain included in the NTLM data, into an RMFT user name which it then uses to log in to RMFT Server. The client can use this property to reveal the RMFT user name assigned to this session.

## Methods

### AdminLogin

Call the `AdminLogin` method to log in to a given RMFT Server as an RMFT administrator or a group administrator. This login type enables you to perform RMFT management tasks such as configuring RMFT Server and monitoring transfers and other processes.

### Request Parameters

Parameter	Type	Description
<code>loginOptions</code>	<code>LoginOptions</code>	Either <code>LoginOptions.Credentials</code> or <code>LoginOptions.NTLM</code> .
<code>credentials</code>	<code>Credentials</code>	For <code>LoginOptions.Credentials</code> , this parameter should contain the RMFT server address, Windows user name and password, and optional domain. This user should be defined as an RMFT or group administrator.  For <code>LoginOptions.NTLM</code> , this parameter is ignored and can be null.
<code>clientInfo</code>	<code>ClientInfo</code>	This parameter should contain the version of the WSDL file used to build the client application. The version is provided in the WSDL file, for example: <pre>&lt;xs:element name="Version" type="xs:string" fixed="2007/12/31"/&gt;</pre>

**Note** In C#, the `ClientInfo` parameter is filled automatically with the version defined in the `fixed` attribute, so you can simply pass `new ClientInfo()`. In Java, you should explicitly set the version property.

## Response

Parameter	Type	Description
session	Session	A session object that should be used later with the other administrative methods. The application should not directly get or set any of the session's properties, since they are for internal use only.

## Syntax (C#)

```
Session AdminLogin(LoginOptions option,  
                   Credentials cred,  
                   ClientInfo clientInfo);
```

## UserLogin

Call `UserLogin` to log in to a given RMFT Server as an RMFT user. This login type enables users to send packages, view their inbox, and perform other user oriented tasks (similar to the functionality offered by RMFT Web Client or RMFT CLI clients).

### Request Parameters

Parameter	Type	Description
<code>loginOptions</code>	<code>LoginOptions</code>	Either <code>LoginOptions.Credentials</code> or <code>LoginOptions.NTLM</code> .
<code>credentials</code>	<code>Credentials</code>	For <code>LoginOptions.Credentials</code> , the parameter should contain the RMFT Server address, Windows user name and password and optional domain. This user should be defined as an RMFT administrator or an RMFT group administrator.  For <code>LoginOptions.NTLM</code> , this parameter is ignored and can be null.
<code>clientInfo</code>	<code>ClientInfo</code>	This parameter should contain the version of the WSDL file used to build the client application. The version is provided in the WSDL file, for example:  <pre>&lt;xs:element name="Version" type="xs:string" fixed="2007/12/31"/&gt;</pre>

**Note** In C#, the `ClientInfo` parameter is filled automatically with the version defined in the `fixed` attribute, so you can simply pass `new ClientInfo()`. In Java you should explicitly set the version property.

### Response

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session object that should be used later with the other user methods. The application should not directly get or set any of the session's properties,

since they are for internal use only.

### **Syntax (C#)**

```
Session UserLogin(LoginOptions option,  
                  Credentials      cred,  
                  ClientInfo      clientInfo);
```

## KeepAlive

Call `KeepAlive` to preserve a session created by `AdminLogin` or `UserLogin`. The default session timeout is five minutes.

### Request Parameters

Parameter	Type	Description
<code>Session</code>	<code>Session</code>	A session object created by <code>AdminLogin</code> or <code>UserLogin</code> .

### Response

None

### Syntax (C#)

```
void KeepAlive(Session s);
```

## Logout

Call `Logout` to terminate a session created by `AdminLogin` or `UserLogin`, and free the resources held by the Web Service.

### Request Parameters

Parameter	Type	Description
<code>Session</code>	<code>Session</code>	A session object created by <code>AdminLogin</code> or <code>UserLogin</code> .

### Response

None

### Syntax (C#)

```
void Logout(Session s);
```

## Login Samples

The samples below demonstrate how to perform `AdminLogin` to an RMFT Server located on machine **mftcomp**. Both samples use the Direct Connection (RMFT/GQS) method using port 3102. A sample using IIS is provided later below.

The client application should assign the Web Service's URL to the required host. For example, if the Web Service is installed on machine 10.0.33.44, then the URL should be:

**http://10.0.33.44:3102/RMFT**

### C#

```
using RmftClient.RmftService;

using System.Net;    // required for NTLM only

RmftService.Session session = null;

try
{
    RmftService.SessionManager SessionMgr = new SessionManager();

    SessionMgr.Url = "http://mftcomp:3102/RMFT";

    // AdminLogin using credentials

    Credentials cred = new Credentials();

    cred.Server = "mftcomp";

    cred.Username = "administrator";

    cred.Password = "topsecret";

    cred.Domain = "acme.com";

    session = SessionMgr.AdminLogin(LoginOptions.Credentials,
                                    cred,
```

```
        new ClientInfo());

    Display("Admin Login done ");

    // UserLogin using NTLM
    SessionMgr.Credentials = CredentialCache.DefaultNetworkCredentials;
    session = SessionMgr.UserLogin(LoginOptions.NTLM,
        null,
        new ClientInfo());

    Display("User Login done; RMFT user is " + session.RmftUserName);

    // Logout
    SessionMgr.Logout(session);

    Display("Logout done");
}

catch (Exception exp)
{
    Display("Error: " + exp.Message);
}
```

## Java

```
SessionManagerProxy SessionMgr = new SessionManagerProxy();
SessionMgr.setEndpoint("http://mftcomp:3102/RMFT");
com.repliweb.rmft.Session session = null;

try
{
```

```
// AdminLogin using credential

Credentials cred = new Credentials();

cred.setServer("mftcomp");

cred.setUsername("administrator");

cred.setPassword("topsecret");

cred.setDomain("");

ClientInfo clientInfo = new ClientInfo();

clientInfo.setVersion("2007/12/04");

session = SessionMgr.adminLogin(LoginOptions.Credentials,
                                cred,
                                clientInfo);

// AdminLogin using NTLM

session = SessionMgr.AdminLogin( LoginOptions.NTLM,
                                null,
                                clientInfo);

:

// Logout

SessionMgr.logout(AdminSession);

System.out.println("Logout done");

}

catch (Exception exp)

{
```

```
:  
}
```

## IIS Sample

This sample shows how to connect to Web Services via IIS using a proxy and NTLM authentication.

```
using RmftClient.RmftService;  
  
using System.Net; // required for NTLM only  
  
string IIS_URL = @"http://mftcomp/repliweb-mft/rw_soapd.dll?RMFT";  
string PROXY_URL = @"http://myproxy:8080";  
  
RmftService.Session session = null;  
  
SessionManager SessionMgr = new SessionManager();  
SessionMgr.Url = IIS_URL;  
SessionMgr.Proxy = new WebProxy(PROXY_URL);  
  
// UserLogin using NTLM  
SessionMgr.Credentials = CredentialCache.DefaultNetworkCredentials;  
session = SessionMgr.UserLogin(LoginOptions.NTLM,  
                                null,  
                                new ClientInfo());  
  
// Remember to assign the same URL and proxy of every  
// Web Services 'Manager' object that you create, for example:
```

```
PackageManager PackageMgr = new PackageManager();  
PackageMgr.Url = IIS_URL;  
PackageMgr.Proxy = new WebProxy(PROXY_URL);
```

# 2. VersionManager Object

## Overview

Use this object to get information about web service components and their version. Using this object does not require login, and is usually done before setting an admin or user session.

## Methods

### GetServiceInfo

Call `GetServiceInfo` to retrieve information about the RMFT Web Service such as its version. Call this method to check whether the Web Service version is compatible with the client version. Note that you do not have to log in to call this method.

### Request Parameters

Parameter	Type	Description
<code>serviceName</code>	<code>string</code>	Optional Web Service name. Currently, it should be left null.

### Response

Type	Description
<code>ServiceInfo</code>	The returned information consists of the following: <ul style="list-style-type: none"><li>Version: Web Service version string, in format yyyy/mm/dd.</li></ul>

### Syntax (C#)

```
ServiceInfo GetServiceInfo(string serviceName);
```

### GetComponentInfo

This method is for internal use only.

# 3. ConfigManager Object

## Overview

The `ConfigManager` object enables the management of users, groups and distribution list objects. You can list, add, modify and delete these objects in addition to performing other management tasks.

The `ConfigManager` object requires you to be logged in with `AdminLogin`.

Configurable objects such as `User`, `Host`, `Group`, `DistributionList` and `InspectionPolicy`, are called 'entities'. An entity represents an object that can be created independently of other objects and has a unique ID. Every entity can be accessed via the `ConfigManager` methods `Create`, `Get`, `Modify`, `Delete` and `List`.

The entity classes are derived from the base class `RmftEntity`.

## Class Hierarchy

The following classes are used in the `ConfigManager` object:

Class	Description
<code>RmftObject</code>	Base class of all the Web Service objects
<code>Entity</code>	Derived from <code>RmftObject</code> . Implements objects that exist as independent entities and have unique ID and CRUD (Create/Read/Update/Delete) methods, for example, inspection policies.
<code>AddressableEntity</code>	Derived from <code>Entity</code> . Implements objects that can send or receive packages. Examples include Users and Hosts.
<code>ContainerEntity</code>	Derived from <code>AddressableEntity</code> . Implements containers of Addressable Entities. Examples include Groups and Distribution Lists.

## Class-Object Relationships

The table below lists the `ConfigManager` classes and the objects that are associated with each class.

Class	Objects
<code>RmftObject</code> (only exist in other objects)	<ul style="list-style-type: none"> <li>▪ Certificate</li> <li>▪ Notification</li> </ul>
<code>Entity</code>	<ul style="list-style-type: none"> <li>▪ InspectionPolicy</li> <li>▪ NotificationContact</li> <li>▪ EmailTemplate</li> </ul>
<code>AddressableEntity</code>	<ul style="list-style-type: none"> <li>▪ User</li> <li>▪ Host</li> </ul>
<code>ContainerEntity</code>	<ul style="list-style-type: none"> <li>▪ Group</li> <li>▪ DistributionList</li> </ul>

## Common Types

```
class RmftObject
```

```
{
    // Currently empty
}
```

```
class Entity : RmftObject
```

```
{
    string    EntityID;
    string    DisplayName;    // For example, group description
}
```

```
class AddressableEntity : Entity
{
}

class ContainerEntity : AddressableEntity
{
    AddressableEntity[] Members;
}

enum EntityType
{
    User,
    Group,
    DistributionList,
    Host,
    InspectionPolicy
    Contact,
    EmailTemplate,
    Unknown
}
```

## Configuring Users

### User Object

This class defines the `User` object. The `User` object can use types (enum, flags, and so on) that are defined in the section [User-Related Types](#) below.

All of the fields, with the exception of `EntityId`, are optional and may be null. For instance, when retrieving a user with Basic Detail Level, only the first five fields will contain data while the others will be null.

```
public class User : RmftEntity
```

### Basic Detail Level

See also [UserDetailLevel](#).

```
string          EntityId;
string          FirstName;
string          LastName;
string          Email;
UserReadOnlyAttributes  ReadOnlyAttributes ;
```

### General Detail Level

See also [UserDetailLevel](#).

```
bool           Disabled;           // User is disabled
string         Company;
string         Department;
string         JobTitle;
string         Phone;
string         Fax;
string         MobilePhone;
UserCapabilities  Capabilities;
UserPrivileges   Privileges;
UserEnrollmentState  EnrollmentState;
DMZAssociation   DMZAssociation;
string          DMZExposedName;
string          Domain;           // For Active Directory users
```

Some of the properties are self-explanatory (e.g. company, department, etc.). Those that are not are described in the table below.

Property	Description
Capabilities	<p>Whether the user can send/receive packages, receive packages only or upload/download files (shared folders).</p> <p>See also <a href="#">UserCapabilities</a>.</p>
Privileges	<ul style="list-style-type: none"> <li>▪ Whether the user is "Lite" or "Enhanced" (i.e. is able to download RMFT Active components).</li> <li>▪ Whether the user is permitted to create Ad-Hoc users</li> <li>▪ Whether the user's account expiration is extended automatically each time a package is received (only applicable to Ad Hoc users)</li> <li>▪ Whether the user's account will expire at some point in the future</li> <li>▪ Whether the user is permitted to send administrative packages</li> </ul> <p>See also <a href="#">UserPrivileges</a>.</p>
EnrollmentState	<p>The enrollment state of the Ad Hoc user.</p> <p>See also <a href="#">UserFilter</a>.</p>
DMZAssociation	<p>In a two-server setup (Internal-DMZ), a user can be associated with the internal RMFT Server or the DMZ Front End RMFT Server.</p> <p>See also <a href="#">Common Types</a>.</p>
DMZExposedName	<p>In a two-server setup (Internal-DMZ), internal users need to be exposed to the DMZ in order to send packages to external entities (users, hosts, etc.). The exposed name is an alias that will identify the user to external entities instead of the internal user's actual user name.</p> <p>See also <a href="#">Common Types</a>.</p>
Domain	<p>Applicable to users synchronized with Active Directory. Users imported from Active Directory are appended with the Active Directory domain name.</p>

## Full Detail Level (until end of User class)

See also [UserDetailLevel](#).

### AccountSettings

The `AccountSettings` subclass defines various settings of the user account. It includes settings applicable to users who have permission to create ad hoc users as well as settings applicable to ad hoc user accounts.

```
Class AccountSettings
{
    DateTime          AccountExpirationTime;
    int               AccountExpirationPeriod; // Template only
    string            Notes;
    Notification[]    Notifications;
    string            WebClientTemplate;      // Web client template name
    WebStartMode      WebClientStartMode;    // ActiveX, Java or Lite
    ushort            ExtendedExpirationDays; // For ad-hoc users
    // For ad-hoc user creator:
    string            AdHocUserTemplateId;    // Template name
}
```

Property	Description
<code>AccountExpirationTime</code>	Date and time that the user's account is set to expire.
<code>AccountExpirationPeriod</code>	This property is only valid when the User object is part of a Creation Template. It defines how many days will elapse from the creation of a user with this template, until it will expire. Note that a template cannot contain property <code>AccountExpirationTime</code> .
<code>Notes</code>	General notes about the user.
<code>Notifications</code>	Any notifications configured for the user. Examples include notifying the user when a package has been delivered to a host, notifying the user when a new package is available for download, and so on.
<code>WebClientTemplate</code>	The RMFT Web Client template assigned to the user.
<code>WebClientStartMode</code>	The RMFT Web Client default start mode: ActiveX, Java or Lite
<code>ExtendedExpirationDays</code>	Only applicable to ad hoc users. The number of days to extend the user's account whenever a package is sent to the user.
<code>AdHocUserTemplateId</code>	Only applicable to users permitted to create ad hoc users. The

	template that will be used to determine the ad hoc user's account settings.
--	---

## Membership

The `membership` subclass contains arrays of groups and distribution lists to which the current user belongs. Each item in the arrays contains a `Group` or `DistributionList` object in Basic Detail Level. These arrays are read-only, and cannot be used to set or change the user's membership. See [ContainerEntity Methods](#).

```
Class Membership
{
    Group[]          Groups;           // Read-only
    DistributionList[] DistributionLists; // Read-only
}
```

Property	Description
Groups	Groups to which the user belongs.
DistributionLists	Distribution lists to which the user belongs.

## Security

The `Security` subclass defines the user's security attributes. The `Password` attribute is the RMFT password specified in `UserLogin`. You can set it when calling the `Create` and `Modify` methods; however, the `Get` and `List` methods never return this value to the client.

```
class Security
{
    string          Password; // Write-only for set by client
    Certificate[]   Certificates;
    OptionActivationStatus EnforcePackageSigning;
    bool            AutoEncryptIncomingFiles;
    bool            AutoSignIncomingFiles;
    OptionActivationStatus PasswordExpiration;
    ushort          PasswordExpirationDays;
    bool            MustChangePasswordOnNextLogin;
    string          SecurityData;
}
```

Property	Description
Password	Set by the client. The user's RMFT password.
Certificates	Public key certificates belonging to the user.
EnforcePackageSigning	The user can be required to sign packages before uploading them to RMFT Server.
AutoEncryptIncomingFiles	Packages can be encrypted with the user's public key before the user downloads them.
AutoSignIncomingFiles	Packages can be signed with the RMFT private key before the user downloads them.
PasswordExpiration	Whether or not the user's password should expire at some point in the future.
PasswordExpirationDays	After how many days the user's password should expire.
MustChangePasswordOnNextLogin	Whether the user is required to change his/her password at the next login.
SecurityData	Any parameters required by the RMFT security hook.

## Policies

The `Policies` subclass specifies the inspection policies used by this user. Note that the array `InspectionPolicies` only contain references to enabled policies. In other words, each item contains an `InspectionPolicy` object in Basic Detail Level.

```
class Policies
{
    bool                IgnoreSystemInspectionPolicies;
    bool                IgnoreGroupInspectionPolicies;
    InspectionPolicy[]  InspectionPolicies;
}
```

Property	Description
IgnoreSystemInspectionPolicies	Whether to apply system inspections to packages uploaded by the user.
IgnoreGroupInspectionPolicies	Whether to apply group inspections to packages uploaded by the user (if the user also belongs to

	a group configured with inspection policies).
InspectionPolicies	Which inspection policies to apply to packages uploaded by the user.

### DistributionRestrictions

The `DistributionRestrictions` subclass defines the send/receive restrictions for the current user.

See also [SendRestrictions and ReceiveRestrictions](#).

```

Class DistributionRestrictions
{
    SendRestrictions          SendTo;
    ReceiveRestrictions       ReceiveFrom;
    AddressableEntity []      SendToList
    AddressableEntity []      ReceiveFromList;
    bool                      AddSendersToSendList;
}

```

Property	Description
SendTo	Whether the user is permitted to send packages and, if so, to a predefined list or to everyone.
ReceiveFrom	Whether the user is permitted to receive packages and, if so, from a predefined list of senders or from everyone.
SendToList	Who the user is permitted to send packages to.
ReceiveFromList	Who the user is permitted to receive packages from.
AddSendersToSendList	Whether to automatically add senders to the list of users to whom the user is permitted to send packages.

## IncomingPackageHandling

The `IncomingPackageHandling` subclass defines optional handling of incoming packages. In this RMFT version you can choose either to perform inbox processing or to send the incoming package to the user's e-mail address (but not both).

See also [IncomingPackageOptions](#).

```
class IncomingPackageHandling
{
    IncomingPackageOptions options; // Process, email or none

    Class InboxProcessing
    {
        bool            Disabled;
        string          Name;
        string          Description;
        string          ExecutablePath;
        string          ExecutableParameters;
        bool            SendMonitorMessages;
        string          OnSuccessMessage;
        string          OnErrorMessage;
        ushort          MaxRetries;
    }

    Class EmailIncomingPackage
    {
        bool            EncryptEmail;
        bool            SignEmail;
    }
}
```

Property	Description
<code>IncomingPackageOptions</code>	Packages can undergo custom processing while in the user's inbox or they can be e-mailed to the user (but not both).
<code>Disabled</code>	Whether to enable inbox processing.
<code>Name</code>	The inbox processing name.
<code>Description</code>	The inbox processing description.
<code>ExecutablePath</code>	The full path of the .exe or .bat file.
<code>SendMonitorMessages</code>	Whether to report the processing event to the package monitor.
<code>OnSuccessMessage</code>	The message to display if the processing is successful.

OnErrorMessage	The message to display if the processing fails.
----------------	---

## Notifications

The `Notifications` subclass defines notifications to be sent for different events. Each event (for example, `OnPackageArrival`) may refer to an array of notifications to be executed when this event occurs. In this version, the arrays may contain one notification of type `EmailNotification`. The notifications for Inbox Processing events may contain one `InboxProcessingEmailNotification` object.

```
class Notifications
{
    // Arrays of EmailNotification objects
    Notification[]    OnPackageArrival
    Notification[]    OnPackageSortingSuccess
    Notification[]    OnPackageSortingError
    Notification[]    OnPushToHostSuccess
    Notification[]    OnPushToHostError

    // Arrays of InboxProcessingEmailNotification objects
    Notification[]    OnInboxProcessingSuccess
    Notification[]    OnInboxProcessingError
}
```

Property	Description
<code>OnPackageArrival</code>	Notify the user when a new package is available for download.
<code>OnPackageSortingSuccess</code>	Notify the user when sorting of an uploaded package completes successfully.
<code>OnPackageSortingError</code>	Notify the user when sorting of an uploaded package fails.
<code>OnPushToHostSuccess</code>	Notify the user when transfer of an uploaded package to the target hosts completes successfully.
<code>OnPushToHostError</code>	Notify the user when transfer of an uploaded package to the target hosts fails.
<code>OnInboxProcessingSuccess</code>	Notify specified recipients (the sender, the user and/or other individuals) if inbox processing completes successfully.
<code>OnInboxProcessingError</code>	Notify specified recipients (the sender, the user and/or other individuals) if inbox processing fails.

## User-Related Types

### Common Types

Public enum OptionActivationStatus

```
{  
    Always,  
    Never,  
    Custom,  
    SystemDefault    // As defined in Site Preferences  
}
```

Public enum DMZAssociation

```
{  
    NotApplicable,    // In standalone setup  
    Internal ,        // Internal, not exposed  
    InternalExposed,  
    External  
}
```

## IncomingPackageOptions

This type defines whether RMFT Server should perform special processing on packages sent to the current user. Two mutually exclusive processing options are available: Running a custom process on the package files (i.e. inbox processing) or sending the package to the user's email address.

```
[Flags]
Public enum IncomingPackageOptions
{
    ProcessPackage,    // Inbox processing
    EmailPackage      // Deliver packages by email
}
```

## SendRestrictions and ReceiveRestrictions

These types define whether a user is limited in his ability to send or receive packages. A user can be prevented from sending packages to certain recipients (users or hosts) or from receiving packages from certain senders.

```
Public enum SendRestrictions
{
    SendToEveryone,
    SendToLimitedList,    // Use list in SendToList
    SendToNoOne
}
```

```
Public enum ReceiveRestrictions
{
    ReceiveFromEveryone,
    ReceiveFromLimitedList,    // Use list in ReceiveFromList
    ReceiveFromNoOne
}
```

## UserCapabilities

This type defines one or more capabilities of the current user.

[Flags]

```
public enum UserCapabilities
{
    SendReceivePackages,
    UploadDownloadFiles,
    ReceivePackages
}
```

## UserDetailLevel

This type is used in the method `Get()`. It defines the level of detail (in terms of properties) the client wants the retrieved user object to contain. Basic Level only contains the user ID, first and last name, and read-only attributes. When getting a user list, for example, in `ListUsers()` or in a group's member list, the user objects are always returned in Basic Level.

See the `User` class definition for contents of the various levels.

Use this option to optimize the queries by preventing transfer of unnecessary data between the client and server.

```
public enum UserDetailLevel
{
    Basic,          // basic information fields
    General,       // basic + general fields
    Full           // all fields
}
```

## UserFilter

This type is optionally used in the `List` method when listing entities of type `User`. It may specify one or more search criteria for the query. See a sample in method `List`.

```
public UserFilter
{
    UserCapabilities Capabilities;    // One or more capability flags
    bool ShowExpired;    // Show expired users (default: no)
}
```

## UserEnrollmentState

This type is relevant to ad-hoc users. It defines the status of their enrollment.

```
Public enum UserEnrollmentState
{
    UserCreated,
    VerificationSentSuccessfully,
    VerificationFailed,
    Verified
    InvitationSent
    InvitationFailed
    RegistrationCompleted
}
```

Property	Description
UserCreated	Ad Hoc user account is created but not yet activated.
VerificationSentSuccessfully	Mail sent to ad hoc user asking him to verify that he is the intended recipient.
VerificationFailed	Failed to send mail to ad hoc user asking him

	to verify that he is the intended recipient.
Verified	Ad hoc user has confirmed that he or she is the intended recipient.
InvitationSent	Mail sent to Ad Hoc user inviting him to log in and download the files.
InvitationFailed	Failed to send mail sent to Ad Hoc user inviting him to log in and download the files.
RegistrationCompleted	Ad Hoc user had verified that he is the intended recipient and provided additional information about himself (after logging in to download the files).

## UserPrivileges

This type defines one or more privileges of the current user.

[Flags]

```
Public enum UserPrivileges
```

```
{
    LiteWebUser,
    AdHocUsersCreator,
    AutoExpirationExtension,
    AccountNeverExpires,
    CanSendAdminPackages
}
```

Property	Description
LiteWebUser	Active RMFT components are not downloaded to the user's computer when the user logs in to RMFT Web Client. Active RMFT components enable users to access shared folders, recover transfers and encrypt/decrypt files.
AdHocUsersCreator	User is permitted to create Ad Hoc users.

AutoExpirationExtension	Applicable to Ad Hoc users only. User's account expiration date is extended automatically whenever the user receives a new package.
AccountNeverExpires	User's account never expires.
CanSendAdminPackages	User is able to send special administrative packages (using BRAIN).

## UserReadOnlyAttributes

This type defines one or more features of the current user. In contrast to most of the other fields, these flags are *read-only*. They are set by the server and cannot be set or modified by the client.

[Flags]

```
Public enum UserReadOnlyAttributes
```

```
{
    SystemObject,
    AdHoc,
    ActiveDirectoryUser,
    Expired,
}
```

Property	Description
SystemObject	A user account created by an internal system process. System accounts should not be modified.
AdHoc	An Ad Hoc user account (i.e. an account created by another user).
ActiveDirectoryUser	A user imported from and synchronized with Active Directory.
Expired	An expired user account.

## Methods

### Create

Call the `Create` method to create a new RMFT entity.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>entity</code>	<code>Entity</code>	An entity object, for example, <code>User</code> , <code>Group</code> or <code>Host</code> , containing the properties to be included in the new object. Property <code>EntityId</code> must contain an entity ID which does not already exist.

### Response

None

### Syntax (C#)

```
void ConfigMgr.Create(Session session, Entity entity);
```

### Sample (C#)

```
RmftService.User u = new RmftService.User();

u.EntityId = "cchaplin";

u.FirstName = "Charlie";

u.LastName = "Chaplin";

u.Company = "Silent Movies Inc.";

u.JobTitle = "Actor";

u.Capabilities = UserCapabilities.SendReceivePackages;

u.CapabilitiesSpecified = true;

u.Privileges = UserPrivileges.CanSendAdminPackages
    | UserPrivileges.AccountNeverExpires;

u.PrivilegesSpecified = true;

u.Security = new UserSecurity();

u.Security.EnforcePackageSigning = OptionActivationStatus.Never;

ConfigMgr.Create(session, u);
```

### Sample (Java)

```
User u = new User();

u.setEntityId("cchaplin");

u.setFirstName("Charlie");

u.setLastName("Chaplin");

u.setCompany("Silent Movies Inc.");

u.setJobTitle("Actor");

String[] capabilities={"SendReceivePackages","UploadDownloadFiles"};

u.setCapabilities(capabilities) ;
```

```
String[] privileges= {"CanSendAdminPackages", "AccountNeverExpires"};  
u.setPrivileges(privileges) ;
```

```
UserSecurity security = new UserSecurity();  
security.setEnforcePackageSigning(OptionActivationStatus.Never);  
security.setPassword("charlie1234");  
u.setSecurity(security);
```

```
configMgr.create(session, u);
```

## Delete

Call the `Delete` method to delete entities by their ID.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>type</code>	<code>EntityType</code>	The type of the entity to be deleted, for example, <code>EntityType.User</code> . The method will verify that the deleted entity corresponds to the specified type.
<code>entityId</code>	<code>string</code>	The ID of the entity to be deleted.

### Response

None

### Syntax (C#)

```
void ConfigManager.Delete(  
    Session session,  
    EntityType type,  
    string entityId);
```

### Sample (C#, Java)

```
ConfigMgr.Delete(session, EntityType.User, "cchaplin");
```

## FindEntity

Call the `FindEntity` method to search for an entity matching a given ID. If a matching entity is found, the method returns its type.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>entityId</code>	<code>string</code>	Entity ID to be checked.

### Response

Type	Description
<code>EntityType</code>	If an entity with ID <code>&lt;entityId&gt;</code> is found, the method returns its type, for example, <code>EntityType.User</code> . Otherwise, it returns <code>EntityType.Unknown</code> .

### Syntax (C#)

```
EntityType ConfigManager.FindEntity(  
    Session session,  
    string entityId)
```

### Sample (C#)

Check what type of entity "cchaplin" is:

```
EntityType EntiType = ConfigMgr.FindEntity(session, "cchaplin");  
if (EntiType == EntityType.Unknown)  
    // No such entity  
else if (EntiType == EntityType.User)  
    // "cchaplin" is a user  
else ...
```

### Sample (Java)

```
EntityType EntiType = configMgr.findEntity(session, "cchaplin");  
  
if (EntiType == EntityType.Unknown){  
    System.out.println("No such entity");  
}  
  
else if (EntiType == EntityType.User){  
    System.out.println("\"cchaplin\" is a user");  
} else {  
    System.out.println("\"cchaplin\" is not a user");  
}
```

## Get

Call the `Get` method to retrieve an entity by its ID. The returned object is built according to the `level` parameter: `Basic`, `General` Or `Full`.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>type</code>	<code>EntityType</code>	The type of the entity to be retrieved, for example, <code>EntityType.User</code> . The method will verify that the retrieved entity corresponds to the specified type.
<code>specs</code>	<code>EntitySpecs</code>	(Optional). This parameter contains the requested level of detail: <code>Basic</code> , <code>General</code> or <code>Full</code> . If the parameter is omitted (null), the default level - <code>Basic</code> - is assumed.  <code>EntitySpecs</code> is an abstract type. Use one of its derived types according to the required entity: For example, to get a user, use <code>UserSpecs</code> , to get a group, use <code>GroupSpecs</code> , and so on. If the relevant 'specs' type has no <code>DetailLevel</code> member, this entity has no detail levels and the method always returns <code>Full</code> level (e.g. <code>EmailTemplateSpecs</code> ).
<code>entityId</code>	<code>string</code>	The ID of the entity to be retrieved.

### Response

Type	Description
<code>GetEntityResponse</code>	A wrapper class that contains the requested entity. To get the returned entity, get property <code>entity</code> and cast it to the required entity type.  The wrapper is requested in order to make the method polymorphic and enable it to return any entity objects.

## Syntax (C#)

```
GetEntityResponse ConfigManager.Get(  
  
    Session      session,  
  
    EntityType   type,  
  
    [EntitySpecs specs],  
    string       entityId);
```

## Sample (C#)

Retrieve all the details of user with ID "cchaplin":

```
UserSpecs specs = new UserSpecs();  
specs.DetailLevel = DetailLevel.Full;  
GetEntityResponse resp =  
    ConfigMgr.Get(session, EntityType.User, specs, "cchaplin");  
if (resp.Entity is User)  
    User u = (User)resp.Entity;
```

## Sample (Java)

```
UserSpecs specs = new UserSpecs();  
specs.setDetailLevel(UserDetailLevel.Full);  
GetEntityResponse resp =  
    configMgr.get(session, EntityType.User, specs, "cchaplin");  
if (resp.getEntity() instanceof User)  
    User userFound = (User) resp.getEntity();
```

## Modify

Call the `Modify` method to modify an existing RMFT entity. The `entity` parameter of this method should contain the properties to be modified. Null properties will be ignored and not modified. For example, to leave the `Company` and `Notes` properties unchanged, either set them to null or leave them unset as follows:

```
myUser.Company = null; // Don't change
myUser.AccountSettings.Notes = null; // Don't change
```

To *change* properties, set new values. These values will override the existing properties. For example:

```
myUser.Company = "New Company Inc.";
myUser.AccountSettings.Notes = new string[1] {"VIP"};
```

To *delete* object properties (string, object, array, and so on), define them as **empty** objects (empty string, empty array, and so on). For example:

```
myUser.Company = ""; // Delete
myUser.AccountSettings.Notes = new string[0]; // Delete
```

## Request Parameters

Parameter	Type	Description
<code>session</code>	Session	A session created by <code>AdminLogin</code> .
<code>entity</code>	Entity	An entity object containing the properties to be modified. The entity to be modified is identified by property <code>entityId</code> .

## Response

None

## Syntax (C#)

```
void ConfigManager.Modify(  
    Session      session,  
    Entity       entity)
```

## Sample (C#)

The following sample code modifies the *Company* property of a user (leaving all other properties unchanged).

```
// Get user in General (default) level  
GetEntityResponse resp =  
    ConfigMgr.Get(session, EntityType.User, null, "cchaplin");  
User u = (User)resp.Entity;  
  
// Modify property  
u.Company = "United Artists";  
u.AccountSettings.Notes = new string[0];  
ConfigMgr.Modify(session, u);
```

## List

Call the `List` method to return a list of entities of a given type. The result is an array where each entity is represented by an object with the given detail level.

You can limit the list by a search filter. Search filters are given by classes derived from the abstract class `EntityFilter`. Filters are available for specific entity types (see below).

## Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>type</code>	<code>EntityType</code>	The type of the entities to be listed, for example, <code>EntityType.User</code> .
<code>specs</code>	<code>EntitySpecs</code>	<p>(Optional). This parameter contains the requested level of detail: <code>Basic</code>, <code>General</code> or <code>Full</code>. If the parameter is omitted (null), the default level - <code>Basic</code> - is assumed.</p> <p><code>EntitySpecs</code> is an abstract type. Use one of its derived types according to the required entity: For example, to get a user, use <code>UserSpecs</code>, to get a group, use <code>GroupSpecs</code>, and so on.</p> <p>If the relevant 'specs' type has no <code>DetailLevel</code> member, this entity has no detail levels and the method always returns <code>Full</code> level (e.g. <code>EmailTemplateSpecs</code>).</p>
<code>filter</code>	<code>EntityFilter</code>	<p>(Optional). A class derived from <code>EntityFilter</code> containing search criteria for the requested entity type. The following filter classes are currently available:</p> <ul style="list-style-type: none"> <li>▪ For <code>User</code> entities: <code>UserFilter</code></li> </ul> <p>For details, see <a href="#">User-Related Types</a>.</p> <ul style="list-style-type: none"> <li>▪ For <code>EmailTemplate</code> entities: <code>EmailTemplateFilter</code></li> </ul> <p>See also <code>EmailTemplateFilter</code> below.</p>

sort	EntitySort	This parameter is for future use and must be NULL.
------	------------	--

## Response

Type	Description
Entity[]	An array of entity objects built with the requested level of detail.

## Syntax (C#)

```
Entity[] = ConfigManager.List(
    Session           session,
    EntityType        type,
    [EntitySpecs     specs],
    [EntityFilter     filter])
```

## Sample 1

List of all the groups in Basic (default) level:

### C#

```
Entity[] arr = ConfigMgr.List(session, EntityType.Group, null, null);
for (int i = 0; i < arr.Length; i++)
{
    Group g = (Group)arr[i];
    :
}
```

## Java

```
Entity[] arr;
arr = configMgr.list(session, EntityType.Group, null, null);
for (int i = 0; i < arr.length; i++) {
    Group g = (Group) arr[i];
    System.out.println(g.getEntityId());
}
```

## Sample 2

Only list users that have the capability to receive packages or upload/download files. Display the results in General detail level:

## C#

```
UserSpecs specs = new UserSpecs();

specs.DetailLevel = DetailLevel.General;

specs.DetailLevelSpecified = true;

UserFilter filter = new UserFilter();

filter.Capabilities = UserCapabilities.ReceivePackages
                    | UserCapabilities.UploadDownloadFiles;

filter.CapabilitiesSpecified = true;

Entity[] arr = ConfigMgr.List(session, EntityType.User, specs, filter);

for (int i = 0; i < arr.Length; i++)
{
    User u = (User)arr[i];
    :
}
```

**Java**

```
UserSpecs specs = new UserSpecs();

specs.setDetailLevel(UserDetailLevel.General);

UserFilter filter = new UserFilter();

String[] capabilities =
    {"ReceivePackages", "UploadDownloadFiles"};

filter.setCapabilities(capabilities);

Entity[] arr = configMgr.list(session, EntityType.User, specs, filter);

for (int i = 0; i < arr.length; i++){
    User u = (User)arr[i];
    System.out.println(u.getEntityId());
}
```

## StartList

This method is similar to `List` method, but executes in Paged List mode. It does not return the entire list but rather stores it on the server so the client can read the list page by page, using method [GetNextRows](#).

Use a paged list if the expected list is long. In this version, you can only use it for listing *users*. Paged list provides faster response times to the client, because it does not have to wait until the entire list is prepared on the server. Paged list also allow you to download the list in the background.

Note that the list generated on the server is a 'snapshot' of the database at the current moment. Therefore, changes made to the database while the client is downloading the list using `GetNextRows` will be invisible to the client. The Client will detect these changes only after closing the enumerator with `EndList`, and calling the next `StartList`.

## Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>type</code>	<code>EntityType</code>	The type of the entities to be listed. In this version it must be <code>EntityType.User</code> .
<code>specs</code>	<code>EntitySpecs</code>	(Optional). This parameter contains the requested level of detail of the listed objects: Basic, General or Full. If the parameter is omitted (null), the default level Basic is assumed.  In this version this parameter must be either null or <code>UserSpecs</code> .
<code>filter</code>	<code>EntityFilter</code>	(Optional). Search criteria for the requested entities. In this version this parameter must be either null or <code>UserFilter</code> .
<code>sort</code>	<code>EntitySort</code>	This parameter is for future use and must be NULL.

## Response

Type	Description
Enumerator	An object that identifies a list prepared in the server. use this object for getting the entire list using <code>GetNextRows</code> .

## Syntax (C#)

```
Enumerator ConfigManager.StartList(
    Session           session,
    EntityType        type,
    [UserSpecs       specs],
    [UserFilter      filter])
```

## Samples

List all the users. Use the default detail level (Basic). Read the results in 'pages' of 50 users each time.

### Sample (C#)

```
Enumerator en =
    ConfigMgr.StartList(session, EntityType.User, null, null);

WriteLine("List includes " + en.ListSize + " users");

for (int nRows = 0; nRows < en.ListSize;)
{
    Entity[] ListPage = ConfigMgr.GetNextRows(session, en, 50);
    for (int i = 0; i < ListPage.Length; i++)
    {
        User u = (User)ListPage[i];
        WriteLine(u.FirstName + " " + u.LastName);
    }
    nRows += ListPage.Length;
}
```

```
}  
ConfigMgr.EndList(session, en);
```

### Sample (Java)

```
Enumerator en =  
    configMgr.startList(session, EntityType.User, null, null);  
System.out.println("List includes " + en.getListSize() + " users");  
for (int nRows = 0; nRows < en.getListSize(); ) {  
    Entity[] ListPage = configMgr.getNextRows(session, en, 50);  
    for (int i = 0; i < ListPage.length; i++) {  
        User u = (User) ListPage[i];  
        System.out.println(u.getFirstName() + " " + u.getLastName());  
    }  
    nRows += ListPage.length;  
}  
configMgr.endList(session, en);
```

## GetNextRows

Call the `GetNextRows` method to download the next rows of a list of entities that was generated on the server by method `StartList`. For more details and samples see [StartList](#).

After you have downloaded the entire list using `GetNextRows`, call `EndList` to close the enumerator and free the list on the server.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>enumerator</code>	<code>Enumerator</code>	An <code>Enumerator</code> object returned by <code>StartList</code> .
<code>numRows</code>	<code>int</code>	Number of rows to read from the list, starting from the last row downloaded the previous time. If the value is 0, the method returns the entire list.

### Response

Type	Description
<code>Entity[]</code>	An array on <code>&lt;numRows&gt;</code> objects, containing the next entities from the list generated by <code>StartList</code> .

## EndList

Call the `EndList` method to close the enumerator returned by `StartList` and clear the list generated on the server.

Call this method after you have downloaded the entire list using `GetNextRows`.

## Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>enumerator</code>	<code>Enumerator</code>	An enumerator object returned by <code>StartList</code> .

## Response

None

## Containers

A Container is a collection of Addressable Entities (i.e. Users or Hosts). A container is implemented in the entity class `ContainerEntity`.

There are two derived types of `ContainerEntity`:

- `Group`
- `DistributionList`

The `ContainerEntity` class is derived from `AddressableEntity` and includes an array of member objects (in Basic Detail Level). Currently container members can only be users and hosts. There are two ways to add members to a container:

1. Fill the `Members` array with the `AddressableEntity` objects that only contain the member ID's (`EntityId`). Then call methods `Create` or `Modify` to create or modify the container. (See a sample below). If you modify a container and the `Members` array is not null, the existing member list will be deleted and overridden by the new list. To remove all the members from a container, set `Members` to an empty array and call `Modify`.
2. Use methods `AddMemberToContainer` and `RemoveMemberFromContainer` (see below).

Note that the `Membership` subclass of the `User` object specifies the groups and distribution lists to which the user belongs; however, this subclass is read-only and cannot be used to set or change the user's membership.

The container ID (`EntityId`) is the group or distribution list name.

```
public class ContainerEntity : AddressableEntity {  
  
    public AddressableEntity[] Members;  
  
}
```

## Group Object

```
Public class Group : ContainerEntity
```

### Basic Detail Level

```
string EntityId;           // Group name
string DisplayName;       // Description
```

### General Detail Level

```
public AddressableEntity[] Members;
```

### Full Detail Level

This subclass defines additional details about the group and its members. The property `AdminSystemUsers` may contain the names of Windows system users who are administrators of this group. The property `AdminSystemGroups` may contain the names of Windows system groups whose members are administrators of this group.

```
GroupPrivileges           Privileges;
string                    WebClientTemplate;
string[]                  AdminSystemUsers;
string[]                  AdminSystemGroups;
string                    AdHocUserTemplateID;
bool                      IgnoreSystemInspectionPolicies;
InspectionPolicy[]        InspectionPolicies; // Group policies
OptionActivationStatus    EnforcePackageSigning; // Always/never/default
```

Property	Description
WebClientTemplate	The RMFT Web Client template assigned to the group. The template determines the layout and functionality of RMFT Web Client and can be

	customized differently for each group.
AdminSystemUsers	Window system users who are permitted to perform RMFT administrative tasks related to this group.
AdminSystemGroups	Window groups whose members are permitted to perform RMFT administrative tasks related to this group.
AdHocUserTemplateID	The name of the template that will be used to create Ad Hoc users. Only applicable if the group is permitted to create Ad Hoc users.
IgnoreSystemInspectionPolicies	Whether to apply system inspection policies to packages uploaded by group members.
InspectionPolicies	Which inspection policies to packages uploaded by group members.
EnforcePackageSigning	Whether to require group members to sign packages before uploading them to RMFT Server.

### Sample (C#)

Create a group with two members:

```
Group g = new RmftService.Group();
g.EntityId = "Comedians";
g.DisplayName = "Funny people";

g.Members = new AddressableEntity[2];
g.Members[0] = new User();
((User)g.Members[0]).EntityId = "Jerry";
g.Members[1] = new User();
((User)g.Members[1]).EntityId = "Frasier";
ConfigMgr.Create(session, g);
```

### Sample (Java)

```
Group g = new Group();  
g.setEntityId("Comedians");  
g.setDisplayName("Funny people");  
configMgr.create(session, g);  
  
configMgr.addMemberToContainer(session, EntityType.User,  
                                "Jerry", ContainerType.Group, ("Comedians"));  
  
configMgr.addMemberToContainer(session, EntityType.User,  
                                "Frasier", ContainerType.Group, ("Comedians"));
```

## Group Related Types

### GroupPrivileges

This type defines the privileges of group members.

```
enum GroupPrivileges
{
    AdHocUsersCreator
}
```

### DistributionList Object

```
Public class DistributionList : ContainerEntity
```

#### Basic Detail Level

```
string EntityId;           // Distribution List name
string DisplayName;       // Description
```

#### General Detail Level

```
public AddressableEntity[] Members;
```

## ContainerEntity Methods

You can manage containers with the regular entity methods: `Create`, `Modify`, `delete`, `get` and `List`. In addition, you can use the following methods to manage the container's members.

### AddMemberToContainer

Call the `AddMemberToContainer` method to add an addressable entity as a member to a container.

#### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>memberType</code>	<code>EntityType</code>	The type of the member to be added. <code>EntityType.User</code> . If given, the method will verify that the added member corresponds to the specified type.
<code>memberId</code>	<code>string</code>	The ID of an addressable entity to be added as a member to the container.
<code>containerType</code>	<code>EntityType</code>	The type of the container, for example, <code>EntityType.Group</code> . If given, the method will verify that the container corresponds to the specified type.
<code>containerId</code>	<code>string</code>	The ID of the container.

#### Response

None

**Syntax (C#)**

```
void ConfigMgr.AddMemberToContainer(Session s,  
    EntityType memberType,  
    string memberId,  
    EntityType containerType,  
    string containerId)
```

## RemoveMemberFromContainer

Call the `RemoveMemberFromContainer` method to remove a member from a container. The method does not delete or affect the member entity, but only removes the member from the given container.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>memberType</code>	<code>EntityType</code>	The type of the member to be removed. <code>EntityType.User</code> . If given, the method will verify that the removed member corresponds to the specified type.
<code>memberId</code>	<code>string</code>	The ID of an addressable entity which is a member in the given container.
<code>containerType</code>	<code>EntityType</code>	The type of the container, for example, <code>EntityType.Group</code> . If given, the method will verify that the container corresponds to the specified type.
<code>containerId</code>	<code>string</code>	The ID of the container.

### Response

None

### Syntax (C#)

```
void ConfigManager.RemoveMemberFromContainer(  
    Session session,  
    [EntityType memberType],  
    string memberId,  
    [EntityType containerType],  
    string containerId)
```

## Inspection Policies

The `InspectionPolicy` entity is used to protect RMFT Server against potential security threats such as viruses. Inspection policies can be applied to all incoming packages and/or to packages uploaded by specific users or to packages created after files are pulled from hosts. Inspection policies can also be applied to packages uploaded by members of a specific group. Any number of inspection policies can be applied, depending on your specific needs. For example, you could have one inspection policy that scans all packages uploaded to RMFT Server for viruses and another inspection policy that prevents certain file types from entering the system. You could also create one inspection policy for the internal RMFT Server and a different, more stringent policy for the DMZ Front End RMFT Server. The internal inspection policy will be applied to packages uploaded by internal users whereas the DMZ policy will be applied packages uploaded by external users.

### InspectionPolicy Object Overview

An inspection policy is identified by a unique policy name. You should first create an inspection policy with a unique policy name which will then serve as its ID (`EntityId` property). You can also associate inspection policies with different entities such as users or hosts or with Containers such as groups.

When creating an inspection policy entity, you can disable it using `DisabledForSystem`. This will prevent its execution by users and other entities. When associating an inspection policy with a user, for example, fill the user's `InspectionPolicies` array (see samples below). Each item in the array only specifies the policy ID (name). If you want to temporarily disable the execution of this policy for this user, set the `DisabledForEntity` property of this item to `false`. To permanently disable the policy for this user, remove the reference to this from the `InspectionPolicies` array.

### Object Definition

```
Public class InspectionPolicy : Entity
```

#### Basic Detail Level

```
string      EntityId;           // Policy name
string      DisplayName;        // Description
bool        DisabledForSystem;  // Disabled in system level
bool        DisabledForEntity;  // Disabled for this entity
PolicyAttributes Attributes;    // DMZ/Internal/both
```

Property	Description
EntityId	The inspection policy name.
DisplayName	A description of the inspection policy.
DisabledForSystem	Whether this policy entity is disabled. Disabled policies cannot be used by other entities.
DisabledForEntity	This property is used only for references to policies from other entities, for example, <code>User.Policies.InspectionPolicies</code> . Use it to temporarily disable the execution of this policy for packages uploaded by this entity.
Attributes	Whether to run the inspection policy on the internal RMFT Server, the DMZ RMFT Server, or both (only applicable if there is a DMZ Front End RMFT Server). See also <a href="#">PolicyAttributes</a> .

## Full Detail Level

### Execution

Subclass **Execution** defines what executable file should be run for this policy, the parameters and what action to take if an error occurs. This subclass is mandatory.

```
class Execution
{
    PolicyExecutionType ExecutionType;    // EXE, DLL, ...
    srting                Path;           // Executable file path
    srting                EntryPoint;     // For DLL
    string                Parameters;     // Command line parameters
    PolicyOnErrorAction OnError;          // Abort/Ignore, ...
    string                SuccessMessage;
    string                ErrorMessage;
```

```

    ushort                MaxRetries;
}

```

Property	Description
ExecutionType	The inspection policy file type (exe, DLL or VBScript). See also <a href="#">PolicyExecutionType</a> .
Path	The path of the inspection policy file.
EntryPoint	The entry point if the inspection policy is a DLL.
Parameters	Any Parameters required by the inspection policy.
OnError	What action to take if an error occurs. See also <a href="#">PolicyOnErrorAction</a> .
SuccessMessage	The message to display in the package audit trail if the package is approved by the inspection policy.
ErrorMessage	The message to display in the package audit trail if the package is rejected by the inspection policy.
MaxRetries	The number of times to retry the policy if an error occurs.

## Notifications

Subclass **Notifications** defines what email notification should be sent when various events occur, for example, if a package is rejected by the inspection policy. Each event may have an array of `Notification` objects; in this version, each array may only contain one email notification (`EmailNotification` object). You can also set an array of contacts that will receive the email notifications. This subclass is optional.

```

class Notifications
{
    Notification[]    OnPackageRejected;
    Notification[]    OnPackageApproved;
    Notification[]    OnPolicyFailed;
    NotificationContact[] Contacts;
}

```

## InspectionPolicy Related Types

### PolicyAttributes

```
[flags]  
enum PolicyAttributes  
{  
    RunOnInternal  
    RunOnDmz  
}
```

### PolicyExecutionType

```
enum PolicyExecutionType  
{  
    EXE  
    DLL  
    VBScript  
}
```

### PolicyOnErrorAction

```
enum PolicyOnErrorAction  
{  
    Abort,  
    Ignore  
}
```

## InspectionPolicy Methods

You can manage the `InspectionPolicy` entity using the common `ConfigManager` methods: `Create`, `Modify`, `Delete`, `Get` and `List`. The following methods are specific to inspection policies.

### VerifyInspectionPolicy

Call method `VerifyInspectionPolicy` to verify that an inspection policy with a given name exists and can be loaded and executed.

#### Request Parameters

Parameter	Type	Description
<code>session</code>	Session	A session created by <code>AdminLogin</code> .
<code>policyName</code>	string	The name of an existing inspection policy.

#### Response

Type	Description
<code>PolicyVerifyResult[]</code>	The response contains one or more entries for each policy used by the given entity. If the policy is invalid, the response will contain an entry for each error detected. If the policy is valid, the array will contain one entry where <code>ErrorCode</code> equals 0.

#### Syntax (C#)

```
PolicyVerifyResult[] ConfigManager.VerifyInspectionPolicy(
    Session session,
    string policyName)
```

## VerifyInspectionPoliciesForEntity

Call `VerifyInspectionPoliciesForEntity` to verify all of the inspection policies used by a particular entity, for example, user. The method also checks that the particular entity can run this policy. For example, an inspection policy marked as "Internal" cannot be applied to packages uploaded by an external user.

The method returns an array of one or more entries for each policy used by the given entity. Each entry contains the policy name and an error code, where error code 0 means that the policy is valid. Note that if several errors are detected for a policy, the array will contain an entry for each error. Each entry all will include the same policy name but will have a different error code.

### Request Parameters

Parameter	Type	Description
<code>session</code>	Session	A session created by AdminLogin.
<code>entityId</code>	string	The ID of an existing entity which is associated with the inspection policies.

### Response

Type	Description
<code>PolicyVerifyResult[]</code>	The response contains one or more entry for each policy used by the given entity. If the policy is invalid, the response will contain an entry for each error detected. If the policy is valid, the array will contain a single entry where <code>ErrorCode</code> equals 0.

### Syntax (C#)

```
PolicyVerifyResult[]
```

```
ConfigManager.VerifyInspectionPoliciesForEntity(
```

```
    Session    session,  
    string    entityId)
```

## ListEntitiesByInspectionPolicy

Call `ListEntitiesByInspectionPolicy` to list all the addressable entities (i.e. users or hosts) associated with a given inspection policy.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> .
<code>policyName</code>	<code>string</code>	The name of an existing inspection policy.

### Response

Type	Description
<code>AddressableEntity[]</code>	List of addressable entities that are configured to use this inspection policy.

### Syntax (C#)

```
AddressableEntity[] ConfigManager.ListEntitiesByInspectionPolicy(  
    Session s,  
    string policyName)
```

## Sample (C#)

In the following sample, an inspection policy called `VirusScan` has been created. The policy is applied to a group called "Employees".

```
// Create policy

InspectionPolicy pol = new InspectionPolicy();

pol.EntityId = "VirusScan";

pol.DisplayName = "Check viruses in sent packages";

pol.Execution = new InspectionPolicyExecution();

pol.Execution.ExecutionType = PolicyExecutionType.EXE;

pol.Execution.Path = @"C:\av.exe";

:

ConfigMgr.Create(session, pol);

// Associate policy with a new group

Group g = new Group();

:

g.InspectionPolicies = new InspectionPolicy[1];

g.InspectionPolicies[0] = new InspectionPolicy();

g.InspectionPolicies[0].EntityId = "VirusScan";

ConfigMgr.Modify(session, g);
```

## Sample (Java)

```
// Create a policy

InspectionPolicy pol = new InspectionPolicy();

pol.setEntityId( "VirusScan");

pol.setDisplayName("Check viruses in sent packages");
```

```
pol.setExecution( new inspectionPolicyExecution(
    PolicyExecutionType.EXE,
    "c:\\av.exe",
    null,
    null,
    PolicyOnErrorAction.Abort,
    "Virus scan completed successfully",
    "Virus scan completed with errors",
    new UnsignedShort(4)));

configMgr.create(session, pol);

//Associate policy with a new group      GetEntityResponse resp=
configMgr.get(session, EntityType.Group, null, "g");

Group g = (Group)resp.getEntity();

InspectionPolicy[] policies= new InspectionPolicy[1];
policies[0]= pol;

g.setInspectionPolicies(policies);

configMgr.modify(session, g);
```

## Notifications

Use the `Notification` object to define events that RMFT Server should report. Notifications are derived from the `RmftObject` class and are not entities, that is to say, they have no ID and cannot be created as standalone objects. Rather, they are embedded in other entities such as `User` (see `User.Notifications`) or `Inspection Policy` (see `InspectionPolicy.Notifications`) to report about events that occur in the context of these entities, for example, a new package that arrives in the user's inbox.

`Notification` is an abstract class. To use an actual notification, instantiate one of the following derived classes: `EmailNotification` or `InboxProcessingEmailNotification`.

### Notification Object

```
public abstract class Notification : RmftObject
{
    bool    Disabled; // Disable notification for embedding entity
}
```

### EmailNotification Object

This class defines an email notification sent by RMFT Server. It can either use a default or a user-defined (i.e. custom) email template.

```
public class EmailNotification : Notification
{
    NotificationActionOptions    ActionOptions;
    bool                          UseDefaultTemplate;
    EmailTemplate                 CustomEmailTemplate;
}
```

## InboxProcessingEmailNotification Object

This class defines an email notification that is sent due to Inbox processing events. You can specify a list of email recipients.

```
Public class InboxProcessingEmailNotification : EmailNotification
{
    bool SendToPacakgeSender;

    bool SendToThisRecipient;

    string    SendToOthers;    // One or more email addresses
}
```

## Notification-Related Types

### NotificationActionOptions

This class defines additional notification options.

```
public class enum NotificationActionOptions
{
    Attach_Logs;    // Attach log files to email notification
}
```

### PackageNotificationsOptions

The PackageNotificationsOptions type defines one or more special notification options that can be set for the current package.

```
[Flags]
public enum PackageNotificationsOptions
{
    ActivityNotification = 8
}
```

## NotificationContact

Use the `NotificationContact` entity to define recipients of `InspectionPolicy` notifications as well all host-related notifications (e.g. whether files were successfully pulled from the source host and/or pushed to the target host).

`NotificationContact` is an entity whose unique key (`EntityId`) is the contact's email address. It has no detail levels, so its Basic level is the same as Full level.

### NotificationContact Object

```
public class NotificationContact : Entity
{
    string FirstName;
    string LastName;
    string Company;
}
```

## EmailTemplate

This entity is used by notifications. It defines a template for sending email notifications for a specific event. An event is a combination of event type and its status, for example, `SortingUserPackage Success` or `SortingUserPackage Error`.

Its unique key (`Entity.EntityId`) is the template name. Property `Entity.DisplayName` contains the template's description.

```
class EmailTemplate : Entity
{
    EventType                EventType;
    EventStatus              EventStatus;
    EmailTemplateOptions     Options;
    string                   Sender;
    string                   Subject;
```

```
string          Body;
}
```

## EmailTemplate - Related Types

### EmailTemplateFilter

This type can be used in method `List` to specify search criteria. You can list the template belonging to a specific event, statuses and/or options.

```
public class EmailTemplateFilter
{
    EventType          EventType;

    EventStatus       EventStatuses;    // One or more statuses

    EmailTemplateOptions  Options;      // One or more options
}
```

### EventType

```
public enum EventType
{
    NewPackageInInbox,
    InboxProcessing,
    SortingUserPackage,
    SortingHostPackage,
    PushUserPackageToHostTarget,
    PushHostPackageToHostTarget,
    PushPackageToHostTarget,
    PullFromHostSource,
    InspectionPolicy
}
```

## EventStatus

```
[flags]
public enum EventStatus
{
    Success,
    Error,
    Warning,
    Abort
}
```

## EmailTemplateOptions

This type defines special email template attributes.

```
[flags]
public enum EmailTemplateOptions
{
    SystemTemplate,
    DefaultTemplate // Default template for a given event
}
```

## CreationTemplate

This entity is used as a template to create objects. Currently it can be used to create ADSync and Ad-Hoc user objects. In the future, it may support the creation of additional entities.

Its unique key (`Entity.EntityId`) is the template name. The property `Entity.DisplayName` is the template description.

In addition to above, this entity includes a single member, `TemplateObject`. Currently this can only be an object of type `User`. This object contains properties that will be inserted in new `user` objects that are created using this template.

For a list of properties that can be set in a template, please refer to the section *User Templates* in the chapter *Adding and Configuring Users* in the *RMFT Administrator's Guide*.

If a new user belongs to one or more groups or distribution lists, you can fill the arrays `Membership.Groups` and `Membership.DistributionLists` of the user template (see sample below).

```
class CreationTemplate : Entity
{
    Entity    TemplateObject;
}
```

### Sample (C#)

Create a user-creation template. Users created with this template will be members of group `Evaluation` and will be only able to send packages to users in this group. New users created using this template will expire after 7 days.

```
RmftService.User u = new RmftService.User();

Group g = new Group();

g.EntityId = "Evaluation";

// Send-To Restriction

u.DistributionRestrictions = new UserDistributionRestrictions();
```

```
u.DistributionRestrictions.SendTo = SendRestrictions.SendToLimitedList;
u.DistributionRestrictions.SendToList = new AddressableEntity[1];
u.DistributionRestrictions.SendToList[0] = g;

// Membership
u.Membership = new UserMembership();
u.Membership.Groups = new Group[1];
u.Membership.Groups[0] = g;

// New users will expire 7 days after creation
u.AccountSettings = new UserAccountSettings();
u.AccountSettings.AccountExpirationPeriod = 7;
u.AccountSettings.AccountExpirationPeriodSpecified = true;

// Create the template
CreationTemplate ct = new CreationTemplate();
ct.EntityId = "Evaluation_Users";
ct.DisplayName = "Template for creating evaluation users";
ct.TemplateObject = u;

ConfigMgr.Create(session, ct);
```

# 4. PackageManager Object

## Overview

Use the `PackageManager` object to submit packages and to view packages in your inbox and outbox.

### To submit a package:

1. Call `SessionManager.UserLogin` to log in. The logged in user name will be used to indicate the package sender.
2. Instantiate a new `PackageManager.Package` object.
3. Fill the new object with the package properties, such as subject and recipients.
4. Fill the `PackageFiles` array with the names and sizes of the package files.
5. Call `PackageManager.Create` to register the package on the server and return a package ID for the new package.
6. Call methods `TransferManager.UploadPackageFile` or `TransferManager.UploadPackageFileChunk` to upload the contents of the package files specified in step 4 above.
7. Call method `PackageManager.Submit` to send the package to its recipients. The package sender is assumed to be the currently logged in user.

## Types

### Package Object

```
public class Package : RmftObject

    string                PackageId;

    string                Sender;

    string                Subject;

    string                Message;

    string[]              RecipientNames;    // recipient ID's

    AddressableEntity[]  Recipients;        // recipient objects

    DateTime              ArrivalTime;

    DateTime              LastModifiedTime;

    PackageEvent          LastEvent;

    FileInfo[]            PackageFiles;

    long                  PackageSize;      // Total size in bytes

    PackageOptions         Options;
```

### PackageOptions

The `PackageOptions` type defines one or more special options of the current package. The `HideRecipients` flag prevents recipients from seeing who else the packages was sent to.

```
[Flags]

public enum PackageOptions

{

    HideRecipients    // This is a 'private package'

}
```

## PackageNotificationsOptions

The `PackageNotificationsOptions` type defines one or more special notification options of the current package. The `ActivityNotification` flag notifies the sender (by email) each time the package recipients open the package or download any of its files. When the package expires, the sender will also receive an email listing which recipients opened/downloaded the package and which files each recipient downloaded.

Note that for notifications to be sent, both the user's email address and your organization's SMTP server need to be defined on RMFT Server.

```
[Flags]
public enum PackageNotificationsOptions
{
    ActivityNotification = 8
}
```

## Methods

### Abort

Call the `Abort` method to cancel the process of submitting and uploading a package. You can abort a package only after creating it using `PackageManager.Create`, and before submitting it using `Submit`. If the client program is uploading the package files it should stop the upload before calling `Abort`.

This method will cause the Web Service to delete the package files that have already been uploaded.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>UserLogin</code> .
<code>packageId</code>	<code>string</code>	The ID of this package, returned by <code>PackageManager.Create</code> .
<code>withRecovery</code>	<code>bool</code>	Currently not implemented.

### Response

None

### Syntax (C#)

```
void PackageManager.Abort(  
    Session session,  
    String packageId,  
    bool withRecovery)
```

## Create

Call the `Create` method to register a new package on RMFT Server, and get a unique ID for identifying the package.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>UserLogin</code> .
<code>packageInfo</code>	<code>Package</code>	An object that contains the package properties: subject, recipient IDs, file names, file sizes, and so on.

### Response

Type	Description
<code>string</code>	The method returns a unique ID assigned to the new package. Use this ID to refer to this package in the subsequent method calls.

### Syntax (C#)

```
string PackageManager.Create(
    Session session,
    Package packageInfo)
```

### Sample (C#)

Create a package with a single file (`c:\data\daily.xls`), addressed to user `cchaplin`.

```
Package p = new Package();
p.Subject = "Daily package";
p.RecipientNames = new string[1];
p.RecipientNames[0] = "cchaplin";
```

```
RmftService.FileInfo[] files = new RmftService.FileInfo[1];  
  
files[0] = new RmftService.FileInfo();  
  
files[0].RelativePath = "daily.xls"; // package file name  
  
files[0].Size = new System.IO.FileInfo(@"c:\data\daily.xls").Length;  
  
p.PackageFiles = files;  
  
  
string PkgId = PackageMgr.Create(session, p);
```

## Submit

Call the `Submit` method to send a package which is already stored on RMFT Server. Call this method after you have registered the package using `PackageManager.Create` and uploaded all of its files using the `TransferManager` upload methods.

## Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>UserLogin</code> .
<code>packageId</code>	<code>string</code>	The package ID, returned by <code>PackageManager.Create</code> .

## Response

None

## Syntax (C#)

```
void PackageManager.Submit(  
    Session session,  
    string packageId)
```

# 5. AddressBookManager Object

## Overview

Use the `AddressBookManager` object to show the package recipients allowed for a given sender. You can also use it to verify a recipient list and to resolve partial or ambiguous names.

## AddressBook Types

The following types are used by the address book methods `ListAddressBook` and `CheckRecipients`. For samples, see [Address Book Samples](#).

### AddressBookFilter

This type defines search criteria for the address book. Member `ShowTypes` is a `flags` property that can limit the address book to specific recipient types. If this member is null, the address book will include all the recipient types that are valid for the given sender.

Subclasses `UserFilter`, `DistributionListFilter` and `HostFilter` may be used as search criteria for users, distribution lists and host recipients respectively. Each member of these subclasses can contain a name (case insensitive) or its prefix. For example, to find recipients whose company name is "RepliWeb", enter `Company "RepliWeb"`, "Repli" or "repli".

All the subclasses and members of this class are optional.

```
public class AddressBookFilter
{
    AddressableEntityTypes ShowTypes; // user, host and or dist-list

    class UserFilter // Used for user recipients
    {
```

```
    string    EntityId;

    string    FirstName;

    string    LastName;

    string    Company;

    string    Email;
}

class DistributionListFilter {

    string    EntityId;

    string    DisplayName;
}

class HostFilter {

    string    EntityId;
}
}
```

## AddressBookSpecs

Use this class to specify the detail level of the entities returned in the address book. The default level is `Basic`.

```
public class AddressBookSpecs : EntitySpecs

{

    DetailLevel    Level;           // Basic/General/Full
}
}
```

## CheckRecipientResult

Method `CheckRecipients` returns an array of this type. Each array entry corresponds to one of the names passed to this method and contains the following:

- `OriginalName`: The original name (full or partial) passed to the method.
- `MatchingEntity`: If the input name uniquely matches a recipient, this member will contain its entity object (Basic detail level).
- `Suggestions`: If a partial input name was ambiguous, this member will contain an array of suggested entities (Basic level), and `MatchingEntity` will be null. Note that this array may contain entities of different types.
- If no match is found, both of the above members will be null.

For a sample, see [CheckRecipients](#).

```
public class CheckRecipientResult
{
    string          OriginalName;    // Original name to be checked
    AddressableEntity MatchingEntity; // Matching entity (Basic level)
    AddressableEntity[] Suggestions; // Suggestions for ambiguous names
}
```

## EntitySort

Use this class to specify whether the returned address book should be sorted in a specific order. You can specify a sort property as well as whether the list should be displayed in ascending or descending order. By default, the address book is sorted by entity IDs in ascending order.

```
public class EntitySort {  
    EntitySortFields      SortField; // See below.  
    bool                  AscendingOrder;  
}
```

```
public class EntitySortFields {  
    EntityId,  
    DisplayName,  
    First Name,  
    LastName,  
    Company,  
    Email  
}
```

## Methods

### CheckRecipients

Call this method to verify the recipient list given for a specific sender, namely, a user or host entity. You can specify either the full recipient names (case insensitive) or partial recipient names, for example, their prefix, and instruct the method to complete the names. If a prefix matches several recipients, the method can return a list of the names that match the given prefix.

This method can be called in the context of either an admin session or a user session. In a user session, the address book is built from the currently logged-in user whereas in an admin session, you need to explicitly set the sender ID.

If you call `CheckRecipients` with an array of N names, it will return an array of type `CheckRecipientResult[]` with N entries, in the order specified in the input array. Each array entry will contain the following:

- `OriginalName`: The original name (full or partial) passed to the method.
- If the input name uniquely matches a recipient, member `MatchingEntity` will contain its entity object (Basic detail level). If the input name is a partial name, the method will insert the matching entity only if parameter `autoComplete` is `true`.
- If the input name is ambiguous and parameter `showSuggestions` is `true`, member `Suggestions` will contain an array of suggested entities (Basic level). In this case, `MatchingEntity` will be null.
- If no match is found, or if the input name is a partial name and both `autoComplete` and `showSuggestions` are `false`, the above members will be null.

### Request Parameters

Parameter	Type	Description
<code>Session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> or <code>UserLogin</code> .
<code>senderId</code>	<code>string</code>	(Optional). In a user session, enter <code>null</code> since the address book is built for the logged-in user. In an admin session, enter the ID of an entity for which the address book should be built.
<code>recipientNames</code>	<code>string[]</code>	A list of recipient names. You can specify full names or partial (prefix) names (case-insensitive). If you specify partial names, set parameter <code>autoComplete</code> to <code>true</code> .

autoComplete	bool	If true, the specified names may contain partial names. The method will automatically complete partial names that match a single recipient name. If the partial name matches several recipients, the result depends upon the parameter <code>showSuggestions</code> .
showSuggestions	bool	This parameter is required when a partial name matches several recipients: <code>true</code> : Find all the matching names and build in array <code>Suggestions</code> . <code>false</code> : Do not offer suggestions; treat the input name as unknown.

## Response

Type	Description
CheckRecipientResult[]	An array where each entry represents the results of the corresponding input name. See also <code>CheckRecipientResult</code> .

## Syntax (C#)

```
CheckRecipientResult[] PackageManager.CheckRecipients (
    Session session,
    string senderId,
    string[] recipientNames,
    bool autoComplete,
    bool showSuggestions)
```

## Sample (C#)

Check a recipient list for the currently logged-in user (in a `UserLogin` session) and show the results.

```
string[] RecipNames;          // List of recipient names (full or partial)

CheckRecipientResult[] results =

    AddressBookMgr.CheckRecipients(session, null, RecipNames, true, true);

for (int i=0; i<results.Length; i++)
{
    WriteLine("Input name: " + results[i].OriginalName);

    if (results[i].MatchingEntity != null)
        // We have a unique match. Print its type and name
        {
            if (results[i].MatchingEntity is User)
                WriteLine("Exact match: User " +
                    results[i].MatchingEntity.EntityId);
            else if (results[i].MatchingEntity is DistributionList)
                WriteLine("Exact match: DistList " +
                    results[i].MatchingEntity.EntityId);
        }

    else if (results[i].Suggestions != null)
        // The input name is ambiguous. Print the suggested names.
        {
            for (int j = 0; j < results[i].Suggestions.Length; j++)
```

```
        WriteLine("Suggestion: " + results[i].Suggestions[j].EntityId);
    }
    else // No match found for the input name
        WriteLine("Unknown");
}
```

## ListAddressBook

Call this method to return the address book of a given sender, namely, a user or host entity. The address book includes the recipients to which this entity is permitted to send packages. The address book entries can be refined using a filter.

This method can be called in the context of either an admin session or a user session. In a user session, the address book is built from the currently logged-in user whereas in an admin session, you need to explicitly set the sender ID.

The returned address book is limited to 500 recipient names. If there are over 500 recipients, the method will raise an exception "Address book is too long". In this case, either use the filter parameter to limit the address book size, or call method `StartListAddressBook` to get the list in paged mode.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> or <code>UserLogin</code> .
<code>senderId</code>	<code>string</code>	(Optional). In a user session, enter <code>null</code> since the address book is built for the logged-in user. In an admin session enter the ID of an entity for which the address book should be built.
<code>specs</code>	<code>AddressBookSpecs</code>	(Optional). Contains the requested level of detail: Basic, General or Full. If the parameter is omitted (null), the default level Basic is assumed. (See <a href="#">AddressBookSpecs</a> ).
<code>filter</code>	<code>AddressBookFilter</code>	(Optional). Refines the results by limiting the search to specific entity types and search criteria. (See <a href="#">AddressBookFilter</a> ). If this parameter is null, the method will return all the valid recipients.
<code>sort</code>	<code>EntitySort</code>	(Optional). Sorts the results by a specific property in a given order (See <a href="#">EntitySort</a> ). If this parameter is null, the results will be sorted by entity ID in ascending order.

## Response

Type	Description
AddressableEntity[]	An array of addressable entities containing the permitted recipients of the given sender.

## Syntax (C#)

```
AddressableEntity[] PackageManager.ListAddressBook(
    Session session,
    [string senderId],
    [AddressBookSpecs specs],
    [AddressBookFilter filter],
    [EntitySort sort])
```

## Sample (C#)

Get the address book of the currently logged-in user (in a `UserLogin` session). Show only users who belong to company `RepliWeb` and sort by the last name.

```
EntitySort sort = new EntitySort();
sort.SortField = EntitySortFields.LastName;
sort.SortFieldSpecified = true;

AddressBookFilter filter = new AddressBookFilter();
filter.ShowTypes = AddressableEntityTypes.User;
filter.UserFilter = new AddressBookFilterUserFilter();
filter.UserFilter.Company = "RepliWeb";

AddressableEntity[] book =
    PackageMgr.ListAddressBook(session, null, null, filter, sort);
```

```
for (int i = 0; i < book.Length; i++)  
{  
    User u = (User)book[i];  
    Display(u.FirstName + " " + u.LastName);  
}
```

## StartListAddressBook

Call this method to generate the address book of a given sender, namely, a user or host entity. It does not return the address book but rather stores it on the server so that the client can read it page by page, using method `GetNextRows`.

Use a paged list to provide a faster response times to the client if the expected address book list is long. Paged list also allow you to download the list in background.

The address book includes the recipients to which this entity is permitted to send packages. The address book entries can be refined using a filter.

Note that the list generated on the server is a 'snapshot' of the database at the moment method `StartListAddressBook` is called. Therefore, the client will not be aware of any changes made to the database while it is downloading the list using `GetNextRows`. The client will detect these changes only after closing the enumerator with `EndList`, and calling the next `StartListAddressBook`.

This method can be called in the context of either an admin session or a user session. In a user session, the address book is built from the currently logged-in user whereas in an admin session, you need to explicitly set the sender ID.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> or <code>UserLogin</code> .
<code>senderId</code>	<code>string</code>	(Optional). In a user session, enter <code>null</code> since the address book is built for the logged-in user. In an admin session enter the ID of an entity for which the address book should be built.
<code>specs</code>	<code>AddressBookSpecs</code>	(Optional). Specifies the requested level of detail: Basic, General or Full. If the parameter is omitted ( <code>null</code> ), the default level Basic is assumed. (See <a href="#">AddressBookSpecs</a> ).
<code>filter</code>	<code>AddressBookFilter</code>	(Optional). Refines the results by limiting the search to specific entity types and search criteria. (See <a href="#">AddressBookFilter</a> ). If this parameter is <code>null</code> , the method will return all the

		valid recipients.
sort	EntitySort	(Optional). Sorts the results by a specific property in a given order (See <a href="#">EntitySort</a> ). If this parameter is null, the results will be sorted by entity ID in ascending order.

## Response

Type	Description
Enumerator	An object that identifies a list prepared on the server. Use this object for getting the entire list using <code>GetNextRows</code> .

## Syntax (C#)

```

Enumerator PackageManager.ListAddressBook (
    Session session,
    [string senderId],
    [AddressBookSpecs specs],
    [AddressBookFilter filter],
    [EntitySort sort])

```

## Address Book Samples

### Sample (C#)

Get the address book of the currently logged-in user (in a `UserLogin` session). Use the default detail level (Basic) and sort. View the results in 'pages' of 50 recipients at a time.

```
Enumerator en =  
    PackageMgr.StartListAddressBook(session, null, null, null, null);  
  
WriteLine("Address book includes " + en.ListSize + " names");  
  
for (int iPage = 1, nRows = 0; nRows < en.ListSize; iPage++)  
{  
    WriteLine("=== Page " + iPage + "===");  
    BookPage = PackageMgr.GetNextRows(session, en, 50);  
    PrintBook(BookPage);  
    nRows += BookPage.Length;  
}  
PackageMgr.EndList(session, en);
```

```
private void PrintPage(Entity[] bookPage)  
{  
    for (int i = 0; i < bookPage.Length; i++)  
    {  
        if (bookPage[i] is User)  
        {
```

```
        User u = (User)bookPage[i];  
        WriteLine(u.FirstName + " " + u.LastName);  
    }  
    else if (bookPage[i] is DistributionList)  
    :  
}
```

## GetNextRows

Call method `GetNextRows` to download the next rows of a list that was generated on the server by method `StartListAddressBook`. You can get the results in parts ('pages') consisting of a specific number of rows each time.

For more details and a sample see [StartListAddressBook](#).

After you have downloaded the entire list using `GetNextRows`, call `EndList` to close the enumerator and free the list on the server.

## Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> or <code>UserLogin</code> .
<code>enumerator</code>	<code>Enumerator</code>	An <code>Enumerator</code> object returned by <code>StartListAddressBook</code> .
<code>numRows</code>	<code>int</code>	Number of rows to read from the list, starting from the last row downloaded last time. If the value is 0, the method returns the entire list.

## Response

Type	Description
<code>Entity[]</code>	An array of <code>&lt;numRows&gt;</code> objects, containing the next entities from the list generated by <code>StartListAddressBook</code> .

## EndList

Call this method to close the enumerator returned by `StartListAddressBook` and clear the list generated on the server.

Call this method after you have downloaded the entire list using `GetNextRows`.

## Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>AdminLogin</code> or <code>UserLogin</code> .
<code>enumerator</code>	<code>Enumerator</code>	An <code>Enumerator</code> object returned by <code>StartListAddressBook</code> .

## Response

None

# 6. TransferManager Object

## Overview

Use the `TransferManager` object to upload package files to and from RMFT Server (using the client). You can also use MD5 hash to verify that the data received by the client is intact.

Files can only be uploaded as part of a package and only after the package has been created using `PackageManager.Create`.

You can upload a file using either of the following modes: file mode or chunk mode.

- **File Mode** - In File Mode, the client uploads an entire file using the method `UploadPackageFile`. Although these methods support files of any size, this mode is recommended for files no larger than 1MB.
- **Chunk Mode** - In Chunk Mode, the client uploads the file in parts, called chunks, using the method `UploadPackageFileChunk`. You can set the chunk size to any size, but 1MB is the recommended size. Smaller chunks may enable better granularity when monitoring the transfer progress but they might also adversely affect the transfer speed.

## Methods

### GetPackageFileHash

Call this method to verify that a package file has been written correctly on RMFT Server. It returns the MD5 hash and size of the server file, which the client can then compare to the hash and size of the local file.

Note that the other `TransferManager` methods allow you to send or receive the MD5 hash during upload. However, these methods calculate the hash of each chunk separately. Method `GetPackageFileHash`, on the other hand, calculates the hash and size of the entire file while it is stored on the server disk.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>UserLogin</code> or <code>AdminLogin</code> .
<code>packageId</code>	<code>string</code>	Package ID.
<code>remotePath</code>	<code>string</code>	File name of the package file, or relative path below the package files root.
<code>size</code>	<code>out long</code>	Size (in bytes) of the package file on the server.

### Response

Type	Description
<code>string</code>	MD5 hash of the requested server file, after being Base64 encoded into a string.

### Syntax (C#)

```
string TransferManager.GetPackageFileHash(
    Session session,
    string packageId,
    string remotePath,
```

```
out long size)
```

## GetPackageTransferProgress

Call this method to view a report of the transfer progress when a package is uploaded from a client to RMFT Server.

Note that the progress report is not necessarily the exact number of bytes transferred at the time the method is called. For optimization reasons, RMFT Server updates the transfer progress in the RMFT database in intervals as follows:

- In non-chunked upload (method `UploadPackageFile`), the progress is updated once for each file, when the entire file has been uploaded.
- In chunked upload (`UploadPackageFileChunk`), the progress is updated every five seconds. You can change this default by setting the required interval (in seconds) in the following registry key:

```
SoftLink\WebServices\ProgressReportIntervalSec
```

## Request Parameters

Parameter	Type	Description
<code>session</code>	Session	A session created by <code>UserLogin</code> or <code>AdminLogin</code> .
<code>packageId</code>	string	Package ID.

## Response

Type	Description
<code>PackageTransferProgress</code>	<p>The response object provides the following information:</p> <ul style="list-style-type: none"> <li>▪ Package transfer state.</li> <li>▪ Name of file currently being transferred. If the method is called at the precise moment that one transfer ends and before the next transfer begins, this field may be empty.</li> <li>▪ Number of bytes transferred for this file</li> <li>▪ Percentage of file transferred</li> <li>▪ Number of bytes transferred for the entire package</li> <li>▪ Percentage of package transferred</li> </ul>

## Syntax (C#)

```
PackageTransferProgress  
    TransferManager.GetPackageTransferProgress(  
        Session session,  
        string packageId);
```

## Sample (C#)

```
PackageTransferProgress progress =  
    TransferMgr.GetPackageTransferProgress(session, PackageId);  
  
Display(string.Format("{1}% of package uploaded",  
    progress.PackageProgressPercent));
```

## UploadPackageFile

Call this method to upload the contents of an entire package file to the server. You can verify that the sent data was received intact by the server using the parameter `md5Hash`.

You can only upload files that are part of the currently uploaded package, and are specified in the package object. This means that when calling `Create` for this package, the files to be uploaded should be specified in property `Package.PackageFiles`.

### Request Parameters

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>UserLogin</code> .
<code>packageId</code>	<code>string</code>	Package ID created by the method <code>PackageManager.Create</code> . This identifies the uploaded file as part of the specified package.
<code>remotePath</code>	<code>string</code>	Target file name of the uploaded file on the server. This parameter may include a relative path. In this case, the file will be written with this path under the package's "Files" folder on the server.
<code>data</code>	<code>byte[]</code>	Byte array containing the entire contents of the uploaded file.
<code>md5Hash</code>	<code>string</code>	(Optional) String containing the MD5 hash of the uploaded file after being Base64 encoded into a string. If this string is given, the server calculates an MD5 hash of the received file and verifies that both hashes are equal.  If this parameter is null, verification is not performed.

### Response

None

**Syntax (C#)**

```
void TransferManager.UploadPackageFile(
    Session session,
    string packageId,
    string remotePath,
    byte[] data,
    [string md5Hash])
```

**UploadPackageFileChunk**

Call this method to upload the next chunk of a package file to the server. The chunk will be appended to the file being uploaded. You can set the chunk to any size.

You can verify that the uploaded data was received intact by the server using the parameter `md5Hash`.

You can only upload files that are part of the currently uploaded package and that are specified in the package object. This means that when calling `Create` for this package, the files to be uploaded should be specified in property `Package.PackageFiles`

**Request Parameters**

Parameter	Type	Description
<code>session</code>	<code>Session</code>	A session created by <code>UserLogin</code> .
<code>packageId</code>	<code>string</code>	Package ID created by method <code>PackageManager.Create</code> . This identifies the uploaded file as part of the specified package.
<code>remotePath</code>	<code>string</code>	File name of the uploaded file. This is the file's target name on the server.  This parameter may include a relative path. In this case, the file will be written with this path under the package's "Files" folder on the server.
<code>data</code>	<code>byte[]</code>	Byte array containing the current chunk of the uploaded file.
<code>md5Hash</code>	<code>string</code>	(Optional) String containing the MD5 hash of the current chunk after being Base64 encoded into a string. If this string is given, the server calculates

		<p>an MD5 hash of the received chunk and verifies that both hashes are equal.</p> <p>If this parameter is null, verification is not performed.</p>
--	--	--

## Response

None

## Syntax (C#)

```
void TransferManager.UploadPackageFileChunk(
    Session      session,
    string        packageId,
    string        remotePath,
    byte[]        data,
    [string      md5Hash])
```

## Sample (C#)

A package containing a single file is uploaded and submitted in chunk mode. The chunk size is 1MB.

```
const string CLIENT_PATH = @"c:\data\daily.xls";
const string SERVER_PATH = @"daily.xls";
const int CHUNK_SIZE = 1024 * 1024; //1MB

Package p = new Package();

// Set package sender, recipients, subject, and so on

// Set the package's data file in the Package object
p.PackageFiles = new RmftService.FileInfo[1];
p.PackageFiles[0] = new RmftService.FileInfo();
```

```
p.PackageFiles[0].RelativePath = SERVER_PATH;

p.PackageFiles[0].Size = new System.IO.FileInfo(CLIENT_PATH).Length;

// Create package in server, and start uploading the file
string PkgId = m_PackageMgr.Create(m_session, p);
UploadChunked(PkgId, CLIENT_PATH, SERVER_PATH);

// Submit package after files have been uploaded
m_PackageMgr.Submit(m_session, PkgId);

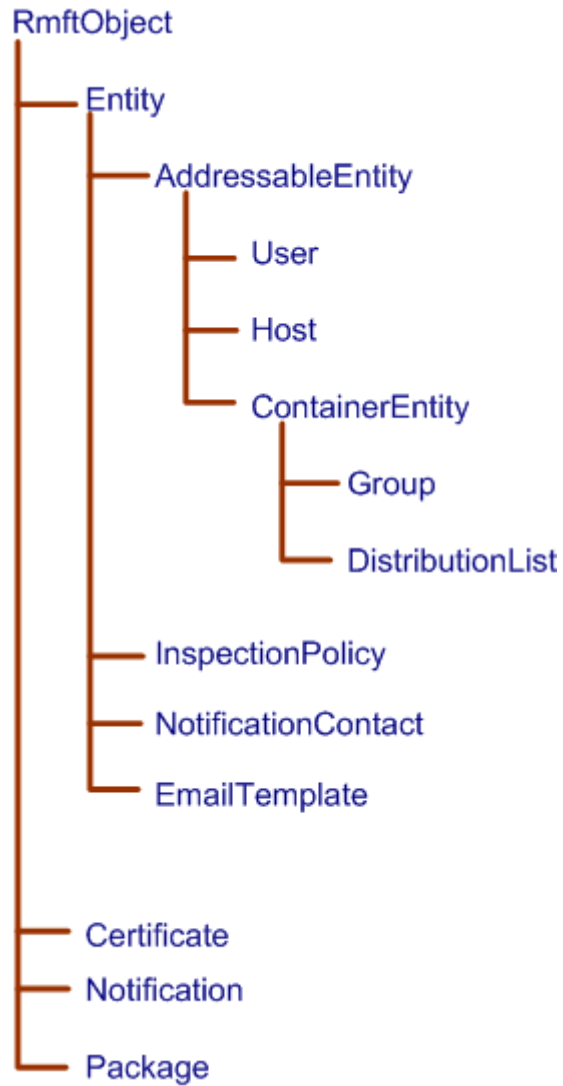
/*
 * Upload file in chunks
 */

void UploadChunked(string packageId, string localPath, string remotePath)
{
    byte[] data = null;
    long offset = 0;
    int nextChunkSize = 0;
    Stream s = File.OpenRead(localPath);

    if (s.Length == 0) // special case: file is empty
    {
        m_TransferMgr.UploadPackageFileChunk(
            m_session, packageId, remotePath, new byte[] {}, null);
        s.Close();
        return;
    }
}
```

```
// Read the next chunk from file and upload it (no verify)
while (s.Position < s.Length)
{
    nextChunkSize = Math.Min(CHUNK_SIZE, (int)(s.Length - offset));
    data = new byte[nextChunkSize];
    s.Read(data, 0, nextChunkSize);
    m_TransferMgr.UploadPackageFileChunk(
        m_session, packageId, remotePath, data, null);
    offset += nextChunkSize;
}
s.Close();
}
```

# A. Class Hierarchy



# B. Troubleshooting

## Log Files

If necessary for support and debug, you can configure RMFT Web Service to generate log files, which will be generated in the folder C:\RMFT\logs.

### To configure RMFT Web Service to generate log files:

1. Open the registry key:  
HKEY\_LOCAL\_MACHINE\SOFTWARE\SoftLink\WebServices
2. Add a string value **TraceMask** if not already present.
3. Edit the **TraceMask** value and enter one of these keywords:

<i>Keyword</i>	<i>Description</i>
<i>bh_flow</i>	Log the Web Service flow to file <b>soap_ws.log</b>
<i>bh_flow,gsoap</i>	Log Web Service flow as well as SOAP messages to these files: <ul style="list-style-type: none"><li>▪ <b>soap_recv.log</b>: SOAP requests sent by the client to Web Service</li><li>▪ <b>soap_sent.log</b>: SOAP responses sent by Web Service to the client</li></ul>
<i>(empty value)</i>	No logging.

4. Restart the RMFT GQS service.