Add-On Products
RealTime Service

API for emails and appointment handling in Exchange

Version: 1.9

Add-On Products
Roms Hule 8 – 7100 Vejle – Denmark
Phone: +45 7944 7000 Fax: +45 7944 7001

Mail: info@add-on.com
Internet: www.add-on.com

Table of contents

CHAPTER 1.

Introduction

Overview

Real Time Service ® (RTS) can be considered as a tool which collects data from Exchange Server and stores it in specified destinations, tracks changes in distribution lists (groups) and user changes in these groups from Active Directory. It also provides interface for developing client applications that use RTS to communicate with Exchange, client applications can use the APIs to send email messages, work with calendar appointment and contact information. The APIs are addressed in the document.

The tool and API is © Add-On Products.

Disclaimer

The content in this document is subject to change without notice.

CHAPTER 2.

RealTime Calendar Service Reference

Applies to: Exchange Online | Office 365 | Exchange Server 2013 | Exchange Server 2016 | Exchange Server 2019

RealTime calendar service API includes two APIs:

- RealTimeCalendarClient
- RealTimeCalendarObjects

Realtime calendar service namespaces

No	Namespaces	Description
1	RealTimeCalendarClient	This is a proxy used to communicate with RTS service.
2	RealTimeCalendarObjects	Contains data and service contracts that are used to communicate with an RTS service. This namespace provides the core RTS APIs functionality.

RealTimeCalendarClient

Classes

No	Name	Description
1	CalendarChannel	Represents a channel to RTS service
2	CalendarConnectionParams	Contain parameters used to setup a connection to RTS service.
3	CalendarClientConnection	Creates a channel to RTS service with the parameters specified by an instance of CalendarConnectionParams
4	CalendarClientConnectionBase<TService>	The abstract class represents a strong typed of RTS service. It contains functions to initiate a new channel to RTS service

RealTimeCalendarObjects

Classes

No	Name	Description
1	CalendarFolder	Represents a folder that contains appointments
2	AppointmentBody	Represents body of appointment
3	Appointment	Contains the properties and methods used to define an appointment or a meeting
4	AppointmentAttachment	Represents an attachment to an Appointment.
5	AppointmentAttachmentData	Represents data of an attachment to an Appointment.
6	AppointmentRecurrencePattern	Specifies recurrence pattern of appointment

No	Name	Description
7	Attendee	Represents an Attendee to an Appointment
8	AvailabilityUser	The availability of an individual user
9	CalendarConnectionConfigurator	The class defines the function to help client connect to Calendar Service
10	CalendarUser	Represents a domain user
11	RTSCalendarServiceFault	Represents the error information returned by RTS service method
12	EmailAddress	Represents an email address of domain user
13	Mail	Represents an email
14	UserAvailabilityResponse	Represents the response to a service operate that get Free/busy status of users in the exchange
15	UserCalendarEvent	Represents a calendar event of users in the exchange
16	UserInfo	Represents information of users in the Active Directory
17	Resource	Represents a domain resource
18	MailboxCalendarSettings	Represents a mailbox calendar setting of user/resource

Interfaces

No	Name	Description
1	ICalendarService	Specifies the APIs that calendar service support client work with service.

Enumerations

No	Name	Description
1	AppointmentMeetingStatus	Specifies the meeting status of appointment
2	AppointmentPriority	Specifies appointment priority
3	AppointmentBodyType	Defines the type of body of an Appointment.
4	AppointmentItemType	Defines the type of appointment items, this specified appointment is either a single, occurrence, exception, or recurring master
5	AppointmentBusyStatus	Specifies constants that define the legacy free/busy status that is associated with an appointment
6	AppointmentRecurrenceType	Represents a recurrence pattern, as used by Appointment item
7	AppointmentRecurrenceEndType	Specifies the end type of appointment
8	AppointmentDayOfWeek	Specifies the day of the week.
9	AppointmentMonthName	Specifies the name of the month.
10	AppointmentDayOfWeekIndex	Defines the index of a week day in a month.

No	Name	Description
11	RecurringDay	Specifies the recurring weekday
12	ResponseType	Specifies response type of the attendee with meeting request
13	AvailabilityUserType	Specifies the type of user availability
14	MailboxType	Specifies the type of Mailbox in the exchange server
15	CalendarSecureMode	Specifies the type of securities mode(binding type) of calendar service
16	CalendarAuthorizationMode	Specifies the type of authorization of calendar service
17	AppointmentUpdateOperation	Specifies the type of update operation of appointment
18	ResourceType	Specifies the type of resource, which can be a user, room or either of them.
19	CalendarSearchObjectType	Specifies the type of object to search in the Active Directory, the available types are User, Group or Any
20	MailPriority	Specifies the priority of the email that is sent via Calendar service
21	MailBodyType	Specifies the type of body of email
22	CalendarErrorType	Specifies the type of error when access calendar
23	CalendarOptions	Specifies the type of body of calendar
24	SearchMask	Specifies the condition type to search for use

CHAPTER 3.
RealTime Calendar Service API

Defines the interface for service client to work with exchange server via Calendar Service

Namespace: RealTimeCalendarObjects

Syntax: `public interface ICalendarService`

Members

Methods

No	Method	Description
1	`string GetCalendarServiceVersion()`	**Summary**: Get Calendar Service Version **Returns**: Version of Real time calendar service
2	`string GetUserAvailability(AvailabilityUser user, DateTime from, DateTime to, int interval, string timeZoneKeyName);`	**Summary**: Get mask of time intervals when user will be available **Parameters**: • user: User, required • from: Date from, in UTC • to: Date to, in UTC • interval: Interval of time window • timeZoneKeyName: Standard name of user's time zone **Returns**: Mask like this "001002400" For example, a request for free/busy data includes four hours(to-from) and an interval of 60 minutes. If the requested user's calendar is OOF for the first 60 minutes, busy for the next 90 minutes, and unscheduled for the final 90 minutes in the time window, the MergedFreeBusy stream will be 3220. If an interval contains more than one availability classification, the highest number is used to classify that interval. Meanings: 0 - Free, 1 - Tentative, 2 - Busy, 3 - Out of Office (OOF), 4 - No data
3	`UserAvailabilityResponse GetUserAvailabilityEvents(AvailabilityUser user, DateTime from, DateTime to, int interval, string timeZoneKeyName)`	**Summary**: Get mask of time intervals when user will be available **Parameters**: • user: User, required

No	Method	Description
		• from: Date from, in UTC • to: Date to, in UTC • interval: Interval of time window • timeZoneKeyName: Standard name of user's time zone **Returns**: Structure contains availability mask like "001002400" and Calendar Events Array For example, a request for free/busy data includes four hours(to-from) and an interval in 60 minutes If the requested user's calendar is OOF for the first 60 minutes, busy for the following 90 minutes, and unscheduled for the final 90 minutes in the time window, the MergedFreeBusy stream will be 3220. If an interval contains more than one availability classification, the highest number is used to classify that interval. Meanings: 0 - Free, 1 - Tentative, 2 - Busy, 3 - Out of Office (OOF), 4 - No data Each Calendar Event contains StartTime, EndTime and AvailibilityStatus
4	`CalendarFolder[] GetUserCalendars(CalendarUser user)`	**Summary:** Get all user's calendar folders **Parameters**: • user: User, required **Returns**: List of calendar folders
5	`Appointment[] GetCalendarAppointmentsPreview(CalendarUser user, CalendarFolder calendar, DateTime from, DateTime to);`	**Summary**: Get appointments at specified interval Interval should not be more than 2 years **Parameters**: • user: User, required • calendar: User's calendar folder, if null then default calendar is used • from: Date from • to: Date to **Returns**:

No	Method	Description
		List of appointments. Only start/end date and subject properties of appointments will be set
6	`Appointment GetCalendarAppointment(CalendarUser user, string appointmentId);`	**Summary:** Loads appointment from exchange **Parameters**: • user: User, required • appointmentId: ItemId of appointment **Returns**: Appointment with all properties been loaded
7	`Appointment GetRecurringMasterAppointment(CalendarUser user, string appointmentId)`	**Summary:** Loads master recurrence Should be called only if Appointment property "AppointmentItemType" equals to Occurrence or Exception **Parameters**: • user: User, required • appointmentId: ItemId of appointment **Returns**: Appointment with all properties and recurring pattern been loaded
8	`Appointment CreateCalendarAppointment(CalendarUser user, CalendarFolder calendar, Appointment, bool sendInvitations);`	**Summary:** Creates new meeting or appointment in exchange **Parameters**: • user: User, required • calendar: User's calendar folder, if null then default calendar is used • appointment: appointment • sendInvitations: Will have no influence if Appointment.MeetingStatus equals to AppointmentMeetingStatus.None or AppointmentMeetingStatus.NonMeeting **Returns**: Appointment is created
9	`Appointment UpdateCalendarAppointment(CalendarUser user, Appointment, AppointmentUpdateOperation updateOperation);`	**Summary:** Modifies already existing meeting or appointment **Parameters**: • user: User, required • appointment: appointment • updateOperation: will have no influence if Appointment.MeetingStatus

No	Method	Description
		equals to AppointmentMeetingStatus.None or AppointmentMeetingStatus.NonMeeting **Returns**: Modified appointment with new ChangeKey
10	`void DeleteCalendarAppointment(CalendarUser user, string appointmentId, bool notifyCancelation);`	**Summary:** Deletes meeting or appointment **Parameters**: • user: User, required • appointmentId: ItemId of appointment to delete • notifyCancelation: If appointment is meeting and has attendees - specify this parameter, otherwise it is not used **Returns**: None
11	`EmailAddress[] GetGlobalAddressList(CalendarUser user, string mask, ResourceType resourceType)`	**Summary**: Get GlobalAddressList(GAL) from user's domain **Parameters**: • user: User, required • mark: Mask to search for user properties, can be null or empty • resourceType: Type or resource - room, user or any **Returns**: List of Email addresses
12	`EmailAddress[] GetGlobalAddressListSpecifiedType(CalendarUser user, string mask, ResourceType, CalendarSearchObjectType objectType)`	**Summary**: Get GlobalAddressList(GAL) from user's domain **Parameters**: • user: User, required • mark: Mask to search for user properties, can be null or empty • resourceType: Type or resource - room, user or any • objectType: Type or calendar object - group, user or any **Returns**: List of Email addresses
13	`AppointmentAttachment CreateAttachment(CalendarUser user, Appointment, AppointmentAttachmentData attachment)`	**Summary:** Saves attachments in appointment **Parameters:** • user: User, required • appointment: appointment

No	Method	Description
		• attachments: Attachment data **Returns:** Appointment attachment
14	`void DeleteAttachment(CalendarUser user, string attachmentId)`	**Summary:** Deletes attachment in exchange **Parameters** • user: User, required • attachmentId: Id of attachment **Returns:** None
15	`byte[] GetAttachmentBody(CalendarUser user, AppointmentAttachment attachment)`	**Summary: Loads attachment body** **Parameters** • user: User, required • attachment: Attachment to be loaded **Returns:** body of attachment in binary
16	`EmailAddress GetMailAddress(CalendarUser user)`	**Summary:** Get email address of user **Parameters:** • user: User, required. Name should be in form of domain\username **Returns:** Email address of domain user
17	`string GetExchangeVersion(CalendarUser user)`	**Summary:** Returns version of exchange for specified user **Parameters**: • user: User, required. Name should be in form of domain\username **Returns:** Exchange2013, Exchange2016, Exchange2019
18	`void SendMail(CalendarUser user, Mail)`	**Summary:** Sends mail **Parameters:** • mail: mail to send • user: User, required. Name should be in the form of domain\username **Returns:** None
19	`UserInfo GetUserInformation(CalendarUser user)`	**Summary:** Gets information about user such as account, first name, last name, display name, address, phone numbers, etc.

No	Method	Description
		Parameters: • user: User, required. Name should be in the form of domain\username **Returns:** User information in Active Directory
20	`Resource[] GetSelectedResources();`	**Summary:** Retrieve a complete list of all selected resources in the RTS. **Parameters:** None **Returns:** A list with SMTP and DisplayName for all resources selected in the RTS

21	`Appointment`
	`CreateCalendarAppointmentWithLogin(CalendarFolder calendar, Appointment appointment, bool sendInvitations, string login, string password);`
		Summary: Creates new meeting or appointment in Exchange with credentials
		Parameters:
		• calendar: User's calendar
		folder, if null then default calendar is used
		• appointment: appointment
		• sendInvitations: Will have no impact if Appointment.MeetingStatus equals AppointmentMeetingStatus.None or AppointmentMeetingStatus.NonMeeting
		• login: email of a user who can make change in the another user's calendar
		• password: password of user who can make a change in another user's calendar
		Returns: Appointment is created

22	`void DeclineCalendarAppointment(CalendarUser user, string appointmentId, string message)`	**Summary:** Declines meeting or appointment **Parameters:** • user: User, required • appointmentId: ItemId of appointment to delete • • message: Content of declined response **Returns**: None

23	`Void`
	`DeleteCalendarAppointmentWithLogin(string`
		Summary: Deletes meeting or appointment with credentials
		Parameters:

No	Method	Description
	`appointmentId, bool notifyCancelation, string login, string password)`	• appointmentId: ItemId of appointment to delete • notifyCancelation: If appointment is meeting and has attendees - specify this parameter, otherwise it is not used • login: email of a user who can make change in the another user's calendar • password: password of user who can make a change in another user's calendar **Returns:** None
24	`Dictionary<string, string> GetADUserProperties(string user, string[] properties)`	**Summary:** Retrieve information about user in the Active Directory **Parameters:** • user: user names in the form of domain/name • properties: list of property names whose values we need **Returns:** A list of pair value, consisting of property and relevant value
25	`Appointment UpdateCalendarAppointmentWithLogin(Appointment appointment, AppointmentUpdateOperation updateOperation, string login, string password);`	**Summary:** Modifies existing meeting or appointment with credentials **Parameters:** • appointment: appointment • updateOperation: will have no impact if Appointment.MeetingStatus equals to AppointmentMeetingStatus.None or AppointmentMeetingStatus.NonMeeting • login: email of a user who can make change in the another user's calendar • password: password of user who can make a change in another user's calendar **Returns**: Modified appointment with new ChangeKey. Appointment's properties RequiredAttendees, OptionalAttendees, Resources, RecurrencePattern will have no impact if Appointment .UpdateProhibitedIsSkipped=true.

No	Method	Description
26	`MailboxCalendarSettings GetMailboxCalendarSettings(CalendarUser user);`	**Summary:** Retrieve a specific mailbox **Parameters:** • user: User, required. Name should be in the form of domain\username **Returns:** return settings of mailbox
27	`string[] SearchAppointments(CalendarUser user, string searchFilterSerialized, string[] propertyDefinitionsSerialized, bool onlyAppointment, bool onlyCalendarFolder)`	**Summary:** Search appointments in Calendar folder/Calendar Logging/Deleted Items folders with help of EwsSearchFilterHelper class for serialize/deserialize. **Parameters:** • user: User, required. Name should be in the form of domain\username • searchFilterSerialized: serialized EWS Managed SearchFilter (use SearchFiler.Serialize extension method) • propertyDefinitionsSerialized: array of serialized props to be retrivieiwed (use SerializePropertyDefinitionBase extension method) • onlyAppointments: return only items of EWS Managed type “Appontment” • OnlyCalendarFolder: return only items for Calendar folder. **Returns**: return sppointments, use DeserializeServiceObject extension method to deserialize
28	`UserInfo[] SearchUsers(SearchUserOptions searchUserOptions)`	**Summary:** Search user by email/AD ID (Guid)/X500Id **Parameters:** • searchUserOptions: search options, fill SomeIds with list of IDs to search **Returns:** Found users.
29	`void FullSyncCalendar(List<CalendarUser> calendars)`	**Summary:** Initiate full synch of users **Parameters:** • calendars: list of users to be full-synched.

No	Method	Description
30	`void RespondCalendarAppointment(CalendarUser user, ResponseType responseType, string appointmentId, string message);`	**Summary:** Change ResponseType of appointment **Parameters:** • user: User, required. Name should be in the form of domain\username • ResponseType: new ResponseType • AppointmentId: Id of appointment • message: response message/reason

Access RTS Public API

To Access RTS Service API, client application must follow these requirements:

1. Use two libraries provided by RTS application
   - *RealTimeCalendarClient.dll*
   - *RealTimeCalendarObjects.dll*
2. Communication port must be opened in client and RTS server machines
   By default:
   - Port 5001 is opened for in-domain access.
   - In case of cross-domain access, port 5003 must be opened.

   NOTE: The ports could be changed via RTS Manager
3. Address of RTS server: IP address or name of RTS server

Client application and RTS server in the same domain

Client application is trusted with RTS service when it runs in the same machine/domain with RTS Server. This is sample code to create a client connection to RTS service in case of in-domain access:

```
private const string Host = "192.168.1.148";
private const int Port = 5001;
private List<CalendarClientConnection> CreateClients()
        {
            List<CalendarClientConnection> list = new List<CalendarClientConnection>();
             CalendarConnectionParams =
CalendarClientConnectionBase<ICalendarService>.CreateDemoParams();
                calendarConnectionParams.Host = Host;
                calendarConnectionParams.Port = Port;
                calendarConnectionParams.CalendarAuthorizationMode =
CalendarAuthorizationMode.Windows;
                calendarConnectionParams.CalendarSecureMode = CalendarSecureMode.NetTcp;
                CalendarClientConnection =
CalendarClientConnection.DemoConnect(calendarConnectionParams);
                calendarClientConnection.Init();
                list.Add(calendarClientConnection);

            return list;
        }
```

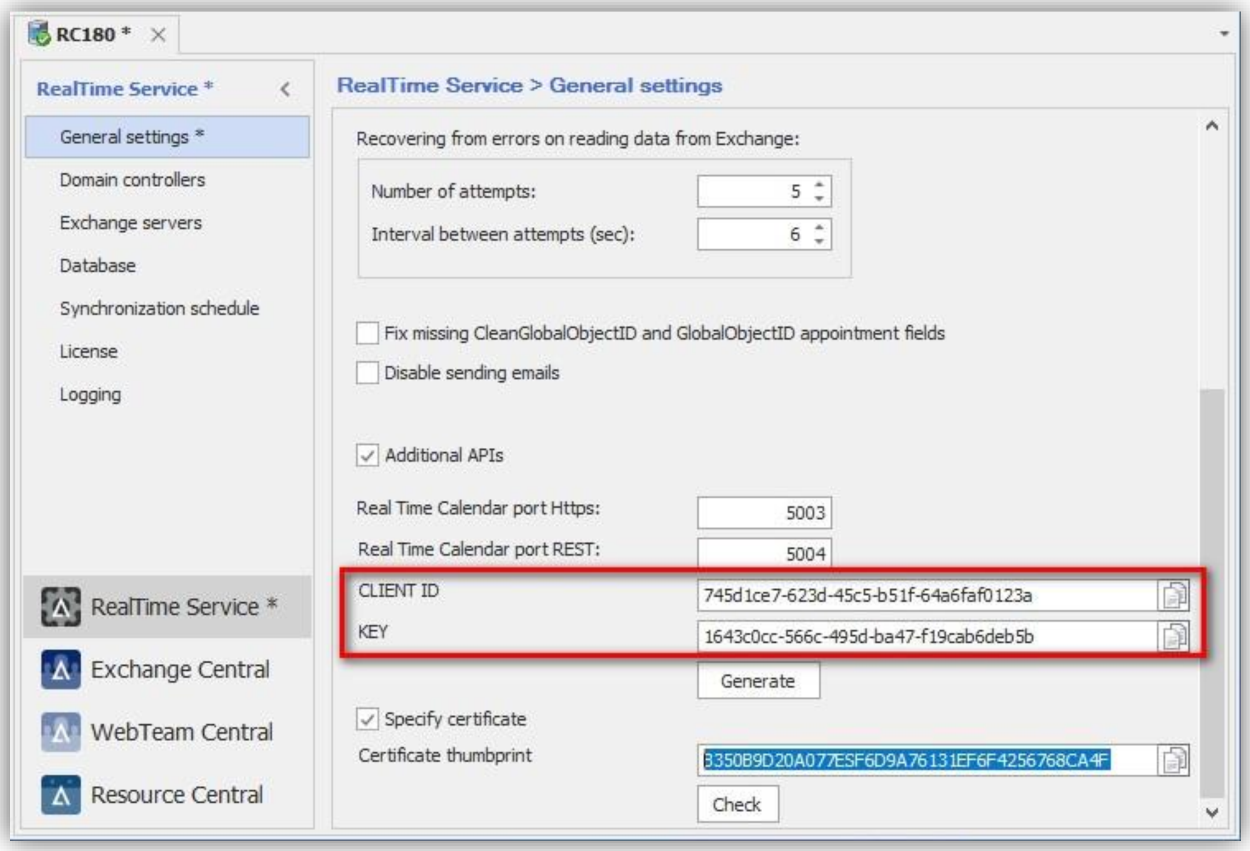
Client application and RTS server in difference domains

In cross-domain connection,

- Client application must use CLIENT ID and KEY generated in RTS service to access API
- A Certificate with private key is required on client and RTS server

1. CLIENTID and KEY can be retrieved from RTS Manager with these steps:

Open RTS Manager, go to **General settings** panel:



Recovering from errors on reading data from Exchange:
Number of attempts: 5
Interval between attempts (sec): 6
☐ Fix missing CleanGlobalObjectID and GlobalObjectID appointment fields
☐ Disable sending emails
☑ Additional APIs
Real Time Calendar port Https: 5003
Real Time Calendar port REST: 5004
CLIENT ID 745d1ce7-623d-45c5-b51f-64a6faf0123a
KEY 1643c0cc-566c-495d-ba47-f19cab6deb5b
☑ Specify certificate
Certificate thumbprint B350B9D20A077E5F6D9A76131EF6F4256768CA4F

RTS Manager – General settings

Check on '*Additional APIs*', then click [**Generate**]. After that, you can have the ID and the key as highlighted in the above figure.

1. Retrieve certificates to make client trusted by RTS server

   Refer to **Appendix A** for more details.

   After retrieving server certificate, we can use RTS function to check with Certificate thumbprint to make sure certificate correctly.

   The thumbprint will be used in the client application to secure connection with RTS.

2. Example of how to create connection in case of multi-domain connection

```
private const string Host = "192.168.1.148";
private const int Port = 5001;

private const string CLIENTID = " cb17feb2-012d-4dd2-90f1-b3c5cc91d6bc";
private const string KEY = " 3624c7e9-5205-4e2c-a1e1-f5ddf204dec2";
private const string THUMBPRINT  = "B2956AF93CA7A76E840BFC5EC1714BD88D60066";
```

```
        private List<CalendarClientConnection> CreateClients()
                {
                    List<CalendarClientConnection> list = new
        List<CalendarClientConnection>();

                        CalendarConnectionParams =
        CalendarClientConnectionBase<ICalendarService>.CreateDemoParams();
                        calendarConnectionParams.Host = Host;
                        calendarConnectionParams.Port = Port;
                       calendarConnectionParams.CalendarAuthorizationMode =
        CalendarAuthorizationMode.UserAuth;
                        calendarConnectionParams.CalendarSecureMode =
        CalendarSecureMode.Https;
                        calendarConnectionParams.UserName = CLIENTID;
                        calendarConnectionParams.Password = KEY;
                       calendarConnectionParams.CertificateThumbprint = THUMBPRINT;
                        CalendarClientConnection =
        CalendarClientConnection.DemoConnect(calendarConnectionParams);
                        calendarClientConnection.Init();
                        list.Add(calendarClientConnection);


                    return list;
                }
```

Examples

Create appointment to Exchange

```
private const string Host = "192.168.1.148";
        private const int Port = 5001;
        private const string Organizer = "administrator@lab4rc.com";

        static void Main(string[] args)
        {
            var appointment = BuildAppoitment();
            try
            {
                var client = CreateClient();
                Console.WriteLine("Connection to Calendar Service successfully with version
{0}", client.Channel.GetExchangeVersion(new CalendarUser { Name = Organizer }));
                var app = client.Channel.CreateCalendarAppointment(new CalendarUser {Name =
Organizer}, new CalendarFolder(),appointment, true);
                Console.WriteLine("Create appointment successfully with id {0}", app.UID);
            }
            catch (Exception exception)
            {
                Console.WriteLine(exception.Message);
            }
            Console.ReadLine();
        }

        private static Appointment BuildAppoitment()
        {
            var app = new Appointment
            {
                Subject = "Test appointment",
                Organizer = new Attendee{EmailAddress = new EmailAddress{Email =
Organizer}},
                StartTime = DateTime.UtcNow,
                EndTime = DateTime.UtcNow.AddHours(1),
                Body = new AppointmentBody { BodyType = AppointmentBodyType.Text,Value =
"Testing appoitment"},
                RequiredAttendees = new List<Attendee> { new Attendee{EmailAddress = new
EmailAddress{Email = "thanh@lab4rc.com"}}},
                Resources = new List<Attendee> { new Attendee { EmailAddress = new
EmailAddress { Email = "res1@lab4rc.com"} } }
            };
            return app;
        }

        private static CalendarClientConnection CreateClient()
        {
            var p = CalendarClientConnection.CreateDemoParams();
            p.Host = Host;
            p.Port = Port;
            var client = CalendarClientConnection.DemoConnect(p);
            client.Init();
            return client;
        }
```

Read appointment from Exchange

```
class Program
    {
        private const string Host = "192.168.1.148";
        private const int Port = 5001;
        private const string Organizer = "administrator@lab4rc.com";

        private const string AppointmentId =
"AAMkADUwNGU5YmFhLTU5MTMtNGY3ZC1hMjgzLTUyOWQ3YjdlNzQ4MABGAAAAAADGhDylQ879SqMl8GTF2xA3BwCCB
OjUCiAPTqolQnMVpPnVAAAAVxByAACCBOjUCiAPTqolQnMVpPnVAAAAWA5QAAA=";

        static void Main(string[] args)
        {
            try
            {
                var client = CreateClient();
                var calendarUser = new CalendarUser
                {
                    Name = Organizer
                };
                ReadAppointmentById(client,calendarUser);

                DateTime fromTime = DateTime.UtcNow;
                DateTime toTime = DateTime.UtcNow.AddDays(1);
                ReadAppointmentInUserCalendarByTime(client, calendarUser, fromTime,
toTime);

                Console.ReadLine();
            }
            catch (Exception exception)
            {
                Console.WriteLine(exception.Message);
            }
        }

        private static void ReadAppointmentInUserCalendarByTime(CalendarClientConnection
client, CalendarUser, DateTime fromTime, DateTime toTime)
        {
            Console.WriteLine("Get appointments from {0} to {1} in user calendar
{2}",fromTime,toTime,Organizer);
            var calendarFolder =
client.Channel.GetUserCalendars(calendarUser).LastOrDefault();
            var appointments = client.Channel.GetCalendarAppointmentsPreview(calendarUser,
calendarFolder, fromTime, toTime);
            foreach (var appointment in appointments)
            {
                PrintAppointment(appointment);
            }
        }

        private static void ReadAppointmentById(CalendarClientConnection
client,CalendarUser calendarUser)
        {
            Console.WriteLine("Read appointment by id {0}", AppointmentId);
            var appointment = client.Channel.GetCalendarAppointment(calendarUser,
AppointmentId);
            if (appointment != null)
            {
                PrintAppointment(appointment);
            }
            else
            {
                Console.WriteLine("Appointment with Id {0} does not exist in user
calendar", AppointmentId);
            }
        }

        private static void PrintAppointment(Appointment appointment)
        {
            Console.WriteLine("Apointment id {0}", appointment.ItemId);
            Console.WriteLine("Subject {0}", appointment.Subject);
            Console.WriteLine("Start: {0}", appointment.StartTime);
            Console.WriteLine("End: {0}", appointment.EndTime);
            Console.WriteLine("Status: {0}", appointment.MeetingStatus);
        }

        private static CalendarClientConnection CreateClient()
        {
            var p = CalendarClientConnection.CreateDemoParams();
            p.Host = Host;
            p.Port = Port;
            var client = CalendarClientConnection.DemoConnect(p);
            client.Init();
            return client;
        }
    }
```

Get global address list

```
class Program
    {
        private const string Host = "192.168.1.148";
        private const int Port = 5001;
        private const string Organizer = "administrator@lab4rc.com";

        private const string Mark = "admin";

        static void Main(string[] args)
        {
            try
            {
                var client = CreateClient();
                var calendarUser = new CalendarUser {Name = Organizer};
                var results = client.Channel.GetGlobalAddressList(calendarUser,
Mark, ResourceType.Any);
                foreach (var emailAddress in results)
                {
                    Console.WriteLine("Mail: {0}",emailAddress.Email);
                    Console.WriteLine("Name: {0}",emailAddress.Name);
                    Console.WriteLine("DisplayName: {0}", emailAddress.DisplayName);
                    Console.WriteLine("MailboxType: {0}", emailAddress.MailboxType);
                }
            }
            catch (Exception exception)
            {
                Console.WriteLine(exception.Message);
            }
            Console.ReadLine();
        }
        private static CalendarClientConnection CreateClient()
        {
            var p = CalendarClientConnection.CreateDemoParams();
            p.Host = Host;
            p.Port = Port;
            var client = CalendarClientConnection.DemoConnect(p);
            client.Init();
            return client;
        }
    }
```

Send email via RTS

```
class Program
    {
        //RTS calendar service server address (sever name or IP address)
        private const string Host = "192.168.1.148";//rc148.rbt-nbg.de
        //RTS calendar service port
        private const int Port = 5001;
        //The account with "SendAs" permision (email address or domain\username)
        private const string SendAsUser = "administrator@rbt-nbg.de";//rbt-
nbg.de\administrator

        static void Main(string[] args)
        {
            try
            {
                //Initial calendar service
                var client = CreateClient();
                //Create a new email
                var mail = GetEmail();
                //Send email via RTS calendar service
                client.Channel.SendMail(new CalendarUser { Name = SendAsUser },
mail);

                Console.WriteLine("Email send from {0}, to {1}
successfully",mail.From, mail.To.FirstOrDefault());
                Console.ReadLine();
            }
            catch (Exception exception)
            {
                Console.WriteLine(exception.Message);
            }
        }

        private static Mail GetEmail()
        {
            //Create a new email
            var mail = Mail.CreateMail();
            Console.Write("Enter mail from: ");
            string mailFrom = Console.ReadLine();
            if (string.IsNullOrEmpty(mailFrom))
            {
                mailFrom = SendAsUser;
            }
            mail.From = mailFrom;
            Console.Write("Enter mail to: ");
            string mailto =  Console.ReadLine();
            if (string.IsNullOrEmpty(mailto))
            {
                mailto = SendAsUser;
            }
            mail.To.Add(mailto);
            Console.Write("Enter subject: ");
            string subject = Console.ReadLine();
            if (string.IsNullOrEmpty(subject))
            {
                subject = "Test mail subject";
            }
            mail.Subject = subject;
            mail.Body = "Test mail body";
            mail.BodyType=MailBodyType.Text;
            return mail;
        }

        private static CalendarClientConnection CreateClient()
        {
            var p = CalendarClientConnection.CreateDemoParams();
            p.Host = Host;
            p.Port = Port;
            var client = CalendarClientConnection.DemoConnect(p);
            client.Init();
            return client;
        }
    }
```

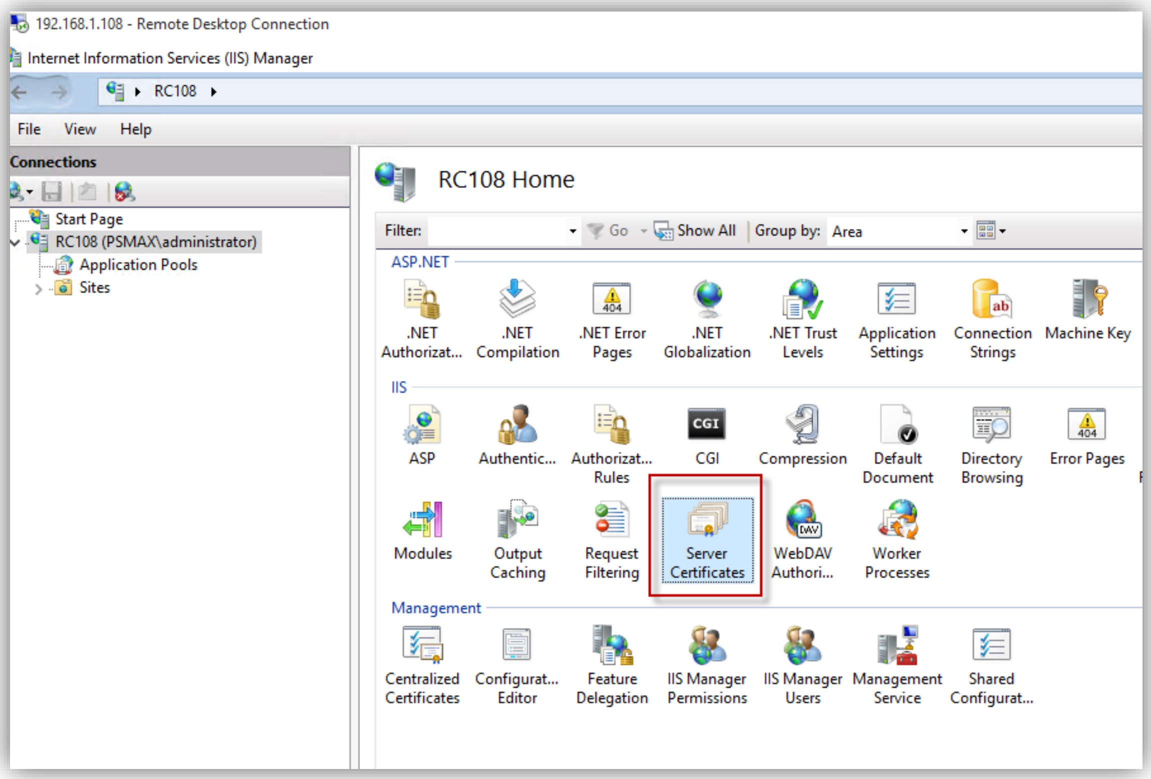
CHAPTER 4.

Appendix

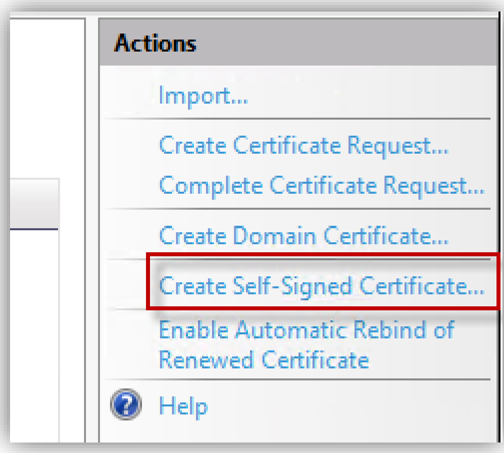
Appendix A – How to create a certificate and import it to client

a. Create certificate in RTS server

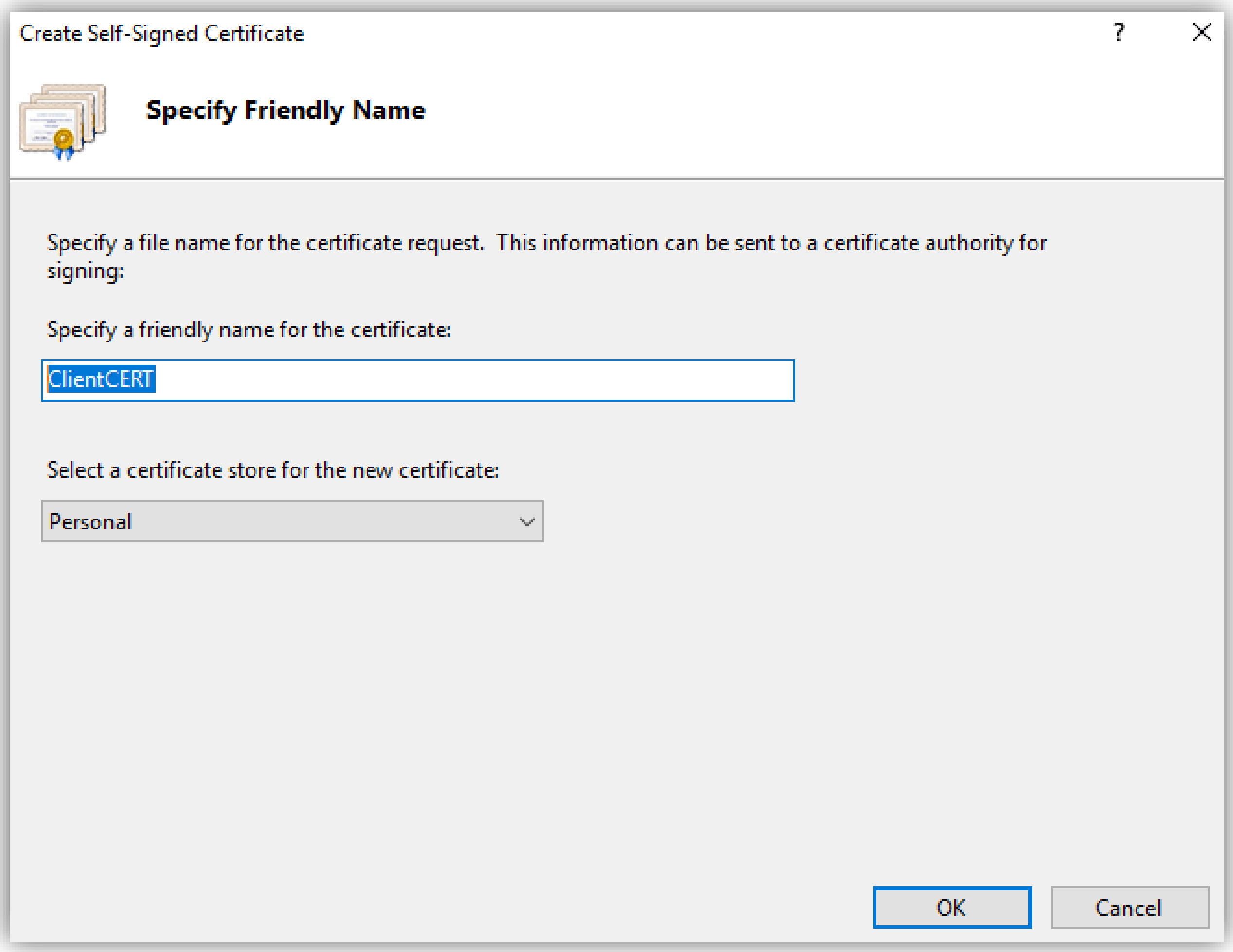
1. In RST server, open IIS manager\Root location\Server Certificates



2. On the right menu, click on Create Self-Signed Certificate


Actions
Import...
Create Certificate Request...
Complete Certificate Request...
Create Domain Certificate...
Create Self-Signed Certificate...
Enable Automatic Rebind of Renewed Certificate
Help

3. Name that Certificate, then click on OK


Create Self-Signed Certificate

Specify Friendly Name

Specify a file name for the certificate request. This information can be sent to a certificate authority for signing:

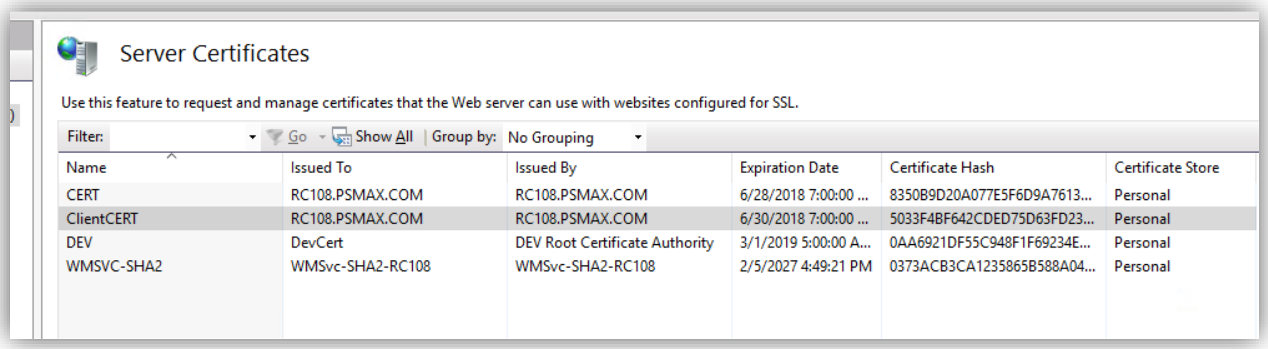
Specify a friendly name for the certificate:

ClientCERT

Select a certificate store for the new certificate:

Personal

Proceed to the next step:

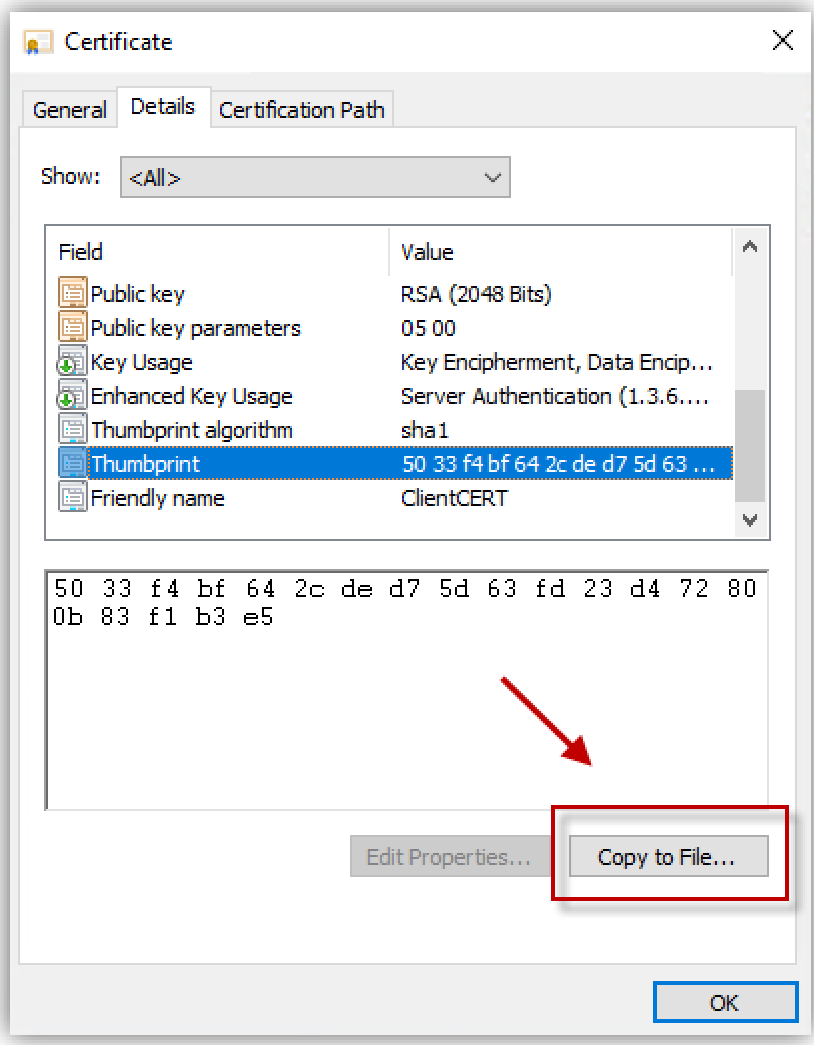

Server Certificates

Use this feature to request and manage certificates that the Web server can use with websites configured for SSL.

Filter: Go Show All Group by: No Grouping

Name	Issued To	Issued By	Expiration Date	Certificate Hash	Certificate Store
CERT	RC108.PSMAX.COM	RC108.PSMAX.COM	6/28/2018 7:00:00 ...	8350B9D20A077E5F6D9A7613...	Personal
ClientCERT	RC108.PSMAX.COM	RC108.PSMAX.COM	6/30/2018 7:00:00 ...	5033F4BF642CDED75D63FD23...	Personal
DEV	DevCert	DEV Root Certificate Authority	3/1/2019 5:00:00 A...	0AA6921DF55C948F1F69234E...	Personal
WMSVC-SHA2	WMSvc-SHA2-RC108	WMSvc-SHA2-RC108	2/5/2027 4:49:21 PM	0373ACB3CA1235865B588A04...	Personal

4. Double click on that certificate, open **Details** tab, then click on **Copy to File**

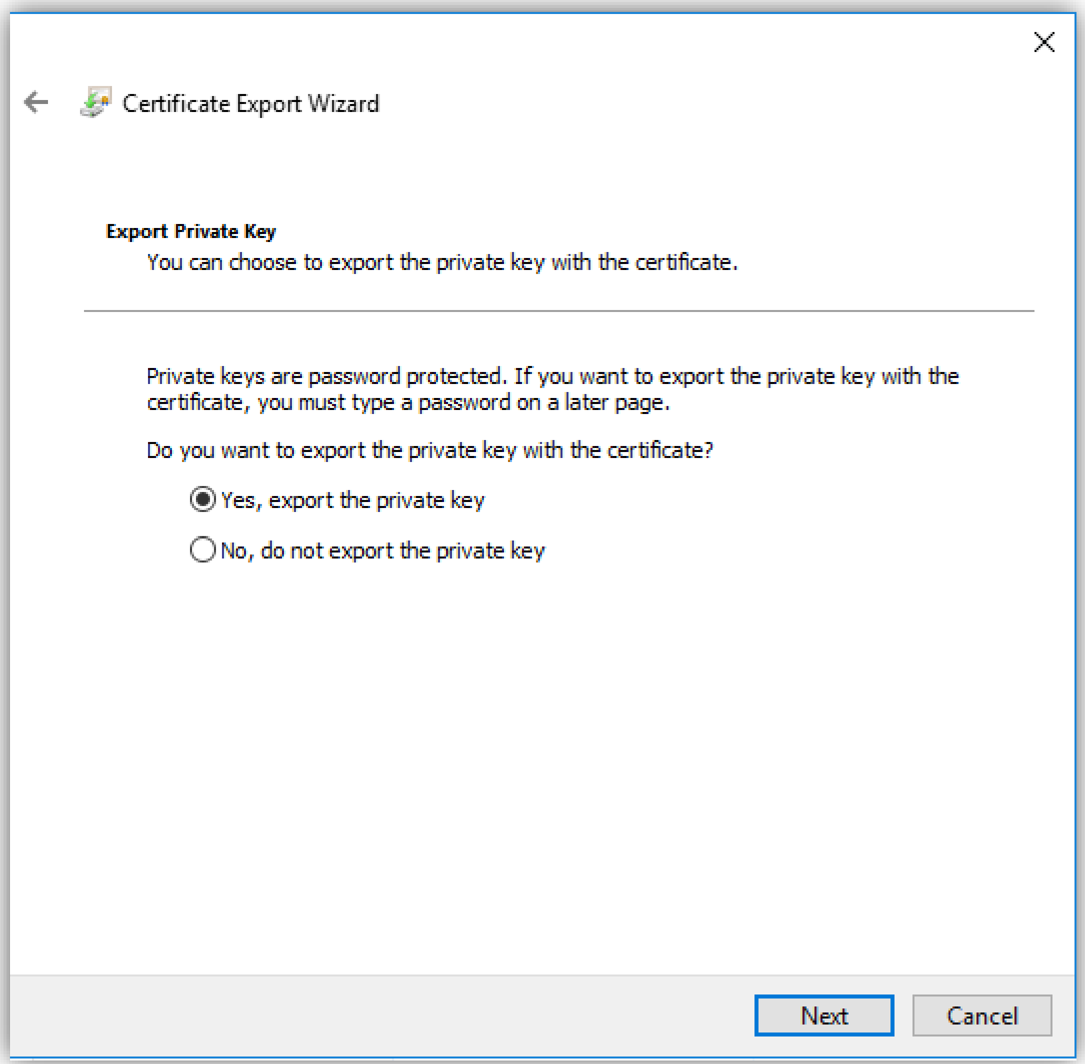


Show: <All>

Field	Value
Public key	RSA (2048 Bits)
Public key parameters	05 00
Key Usage	Key Encipherment, Data Encip...
Enhanced Key Usage	Server Authentication (1.3.6....
Thumbprint algorithm	sha1
Thumbprint	50 33 f4 bf 64 2c de d7 5d 63 ...
Friendly name	ClientCERT

```
50 33 f4 bf 64 2c de d7 5d 63 fd 23 d4 72 80
0b 83 f1 b3 e5
```

5. Follow the instruction flow to finish creating the certificate:



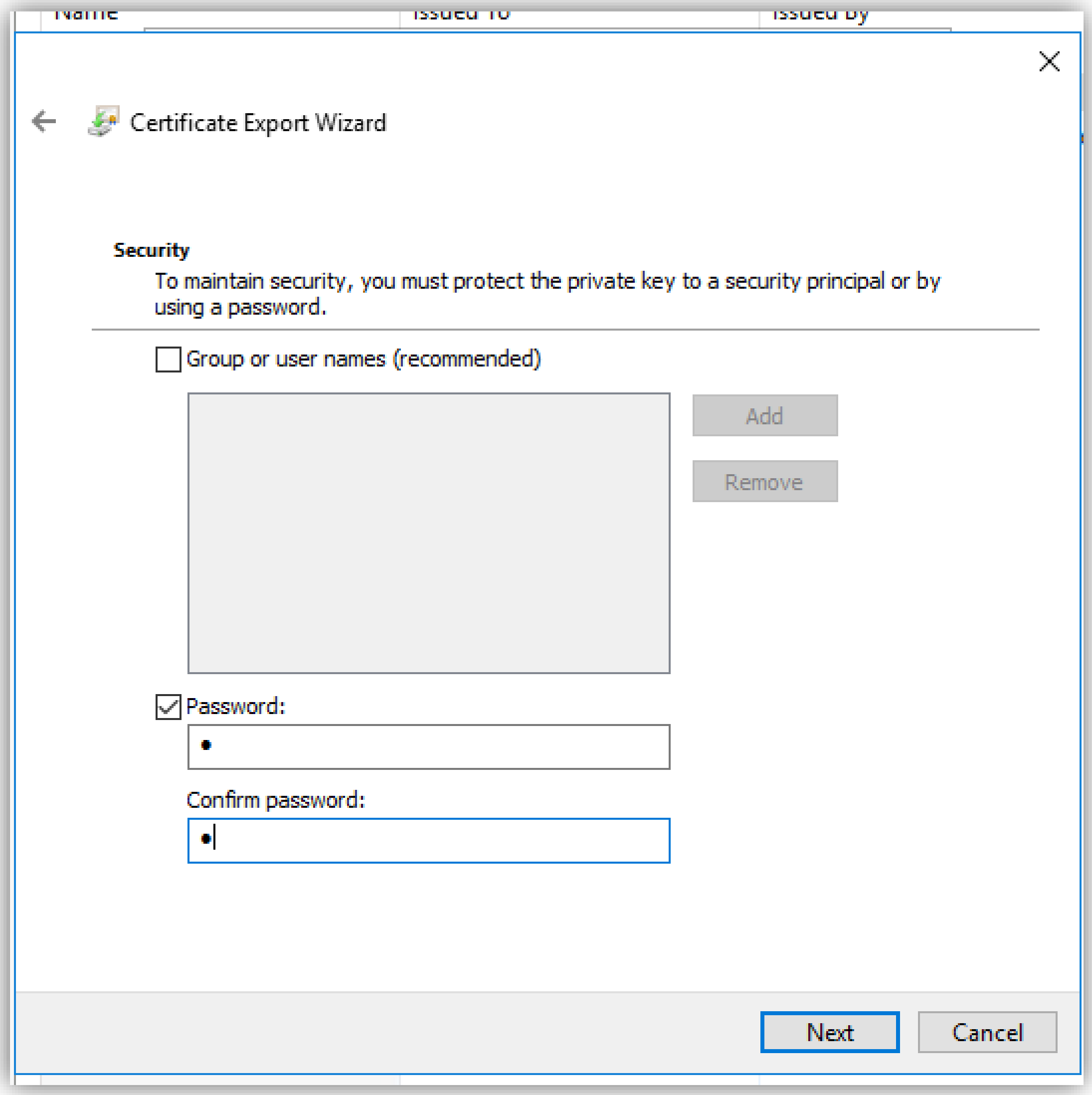
Certificate Export Wizard

Export Private Key
You can choose to export the private key with the certificate.

Private keys are password protected. If you want to export the private key with the certificate, you must type a password on a later page.

Do you want to export the private key with the certificate?

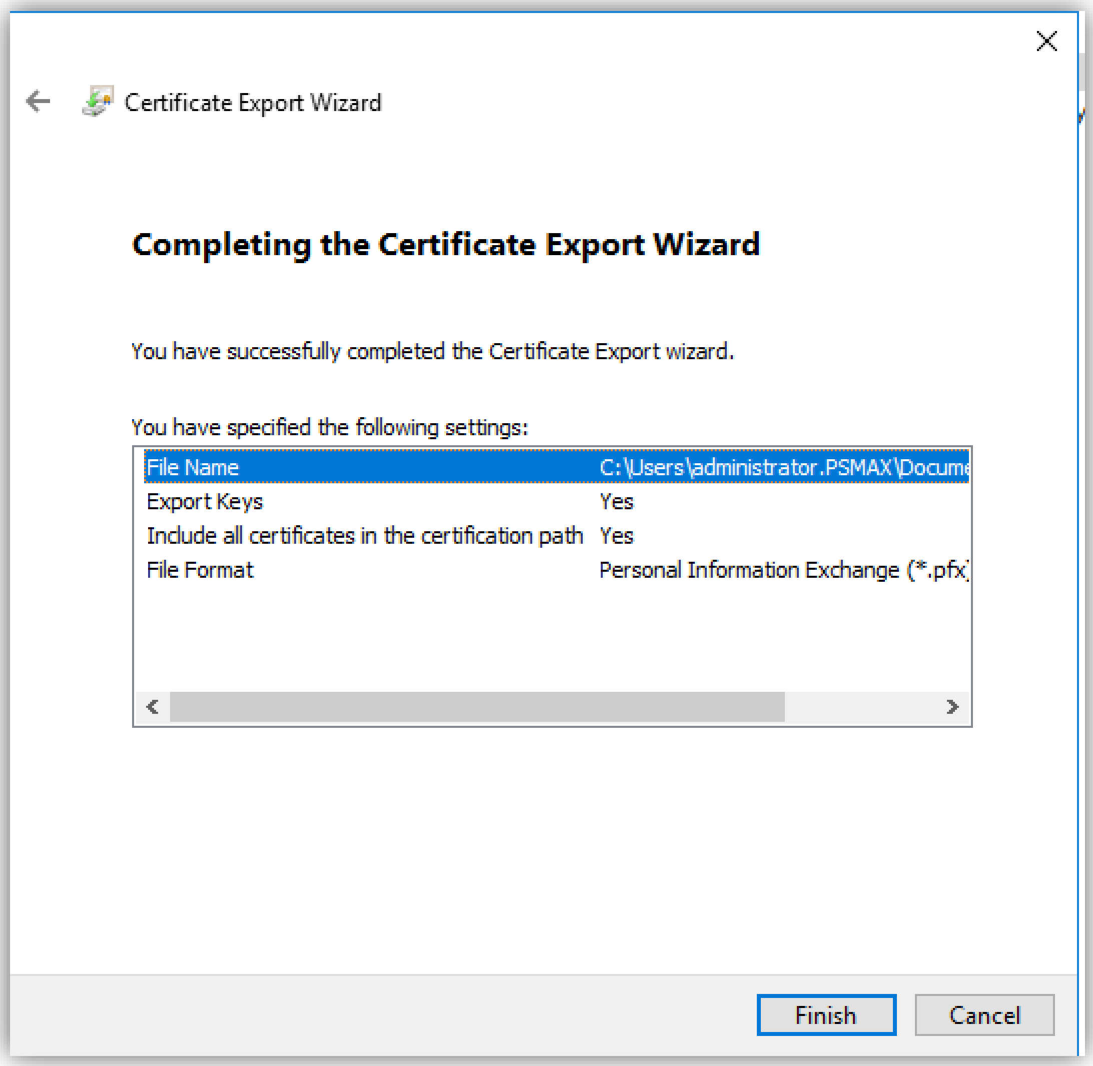
[✓ Yes, export the private key]
No, do not export the private key



Certificate Export Wizard

Security
To maintain security, you must protect the private key to a security principal or by using a password.

- [ ] Group or user names (recommended)
- [x] Password: •
- Confirm password: •



Completing the Certificate Export Wizard

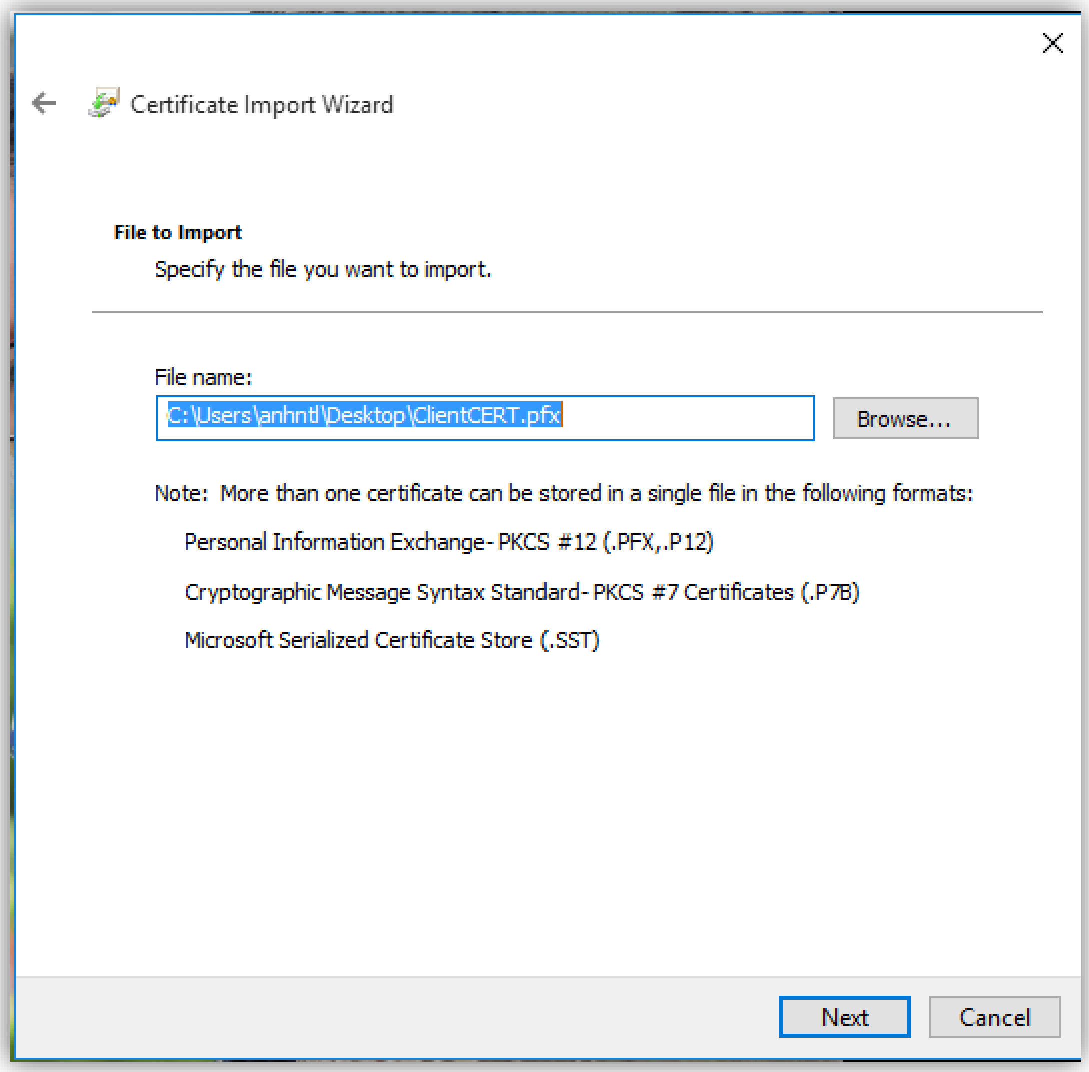
You have successfully completed the Certificate Export wizard.

You have specified the following settings:

File Name	C:\Users\administrator.PSMAX\Docume
Export Keys	Yes
Include all certificates in the certification path	Yes
File Format	Personal Information Exchange (*.pfx)

b. Import the certificate to Client

Copy the certificate file (.pfx) to client machine. Double click that file:



File to Import

Specify the file you want to import.

File name:

C:\Users\anhntl\Desktop\ClientCERT.pfx

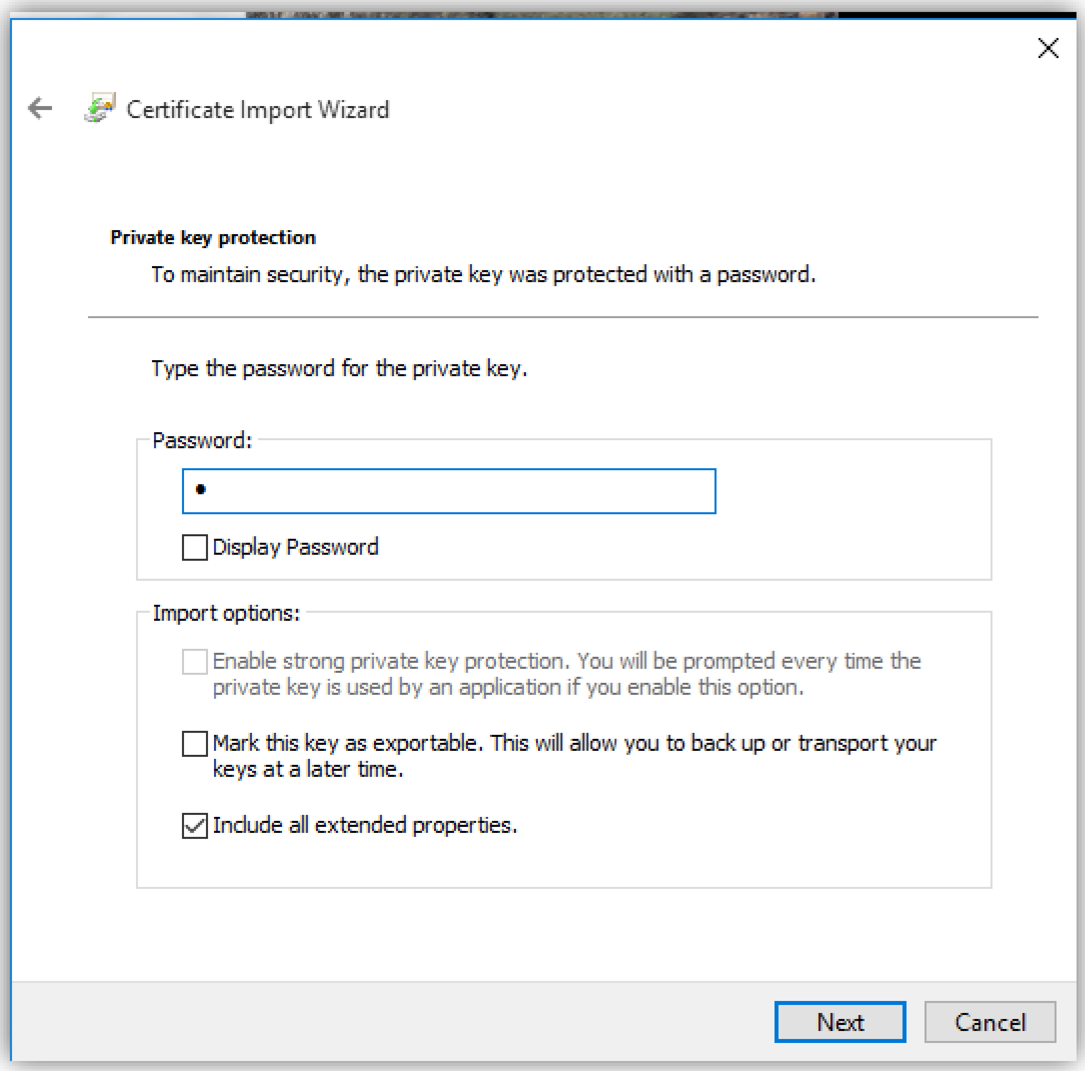
Note: More than one certificate can be stored in a single file in the following formats:

Personal Information Exchange- PKCS #12 (.PFX,.P12)

Cryptographic Message Syntax Standard- PKCS #7 Certificates (.P7B)

Microsoft Serialized Certificate Store (.SST)

Follow the instruction flow to finish importing the certificate:



Certificate Import Wizard

Private key protection

To maintain security, the private key was protected with a password.

Type the password for the private key.

Password:

•

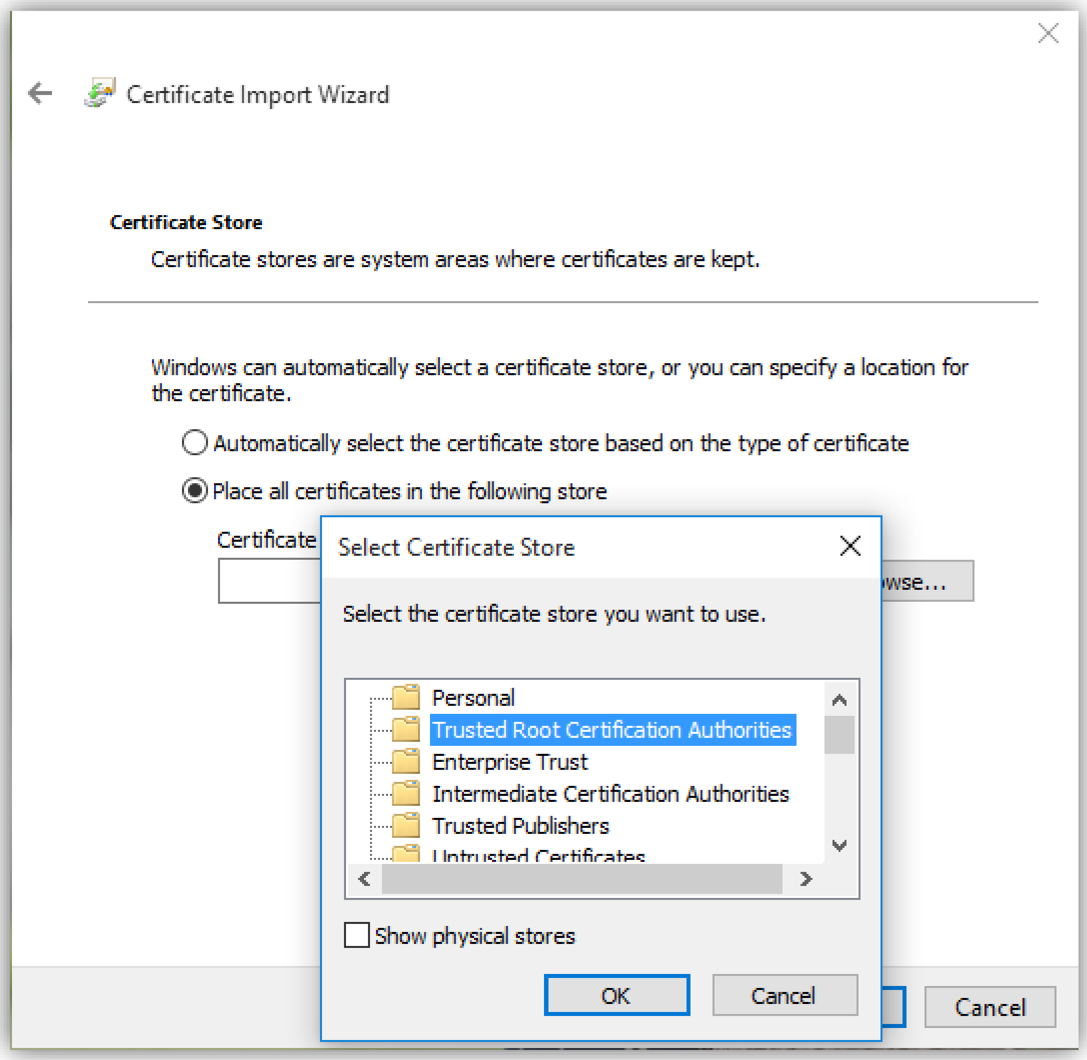
[ ] Display Password

Import options:

[ ] Enable strong private key protection. You will be prompted every time the private key is used by an application if you enable this option.

[ ] Mark this key as exportable. This will allow you to back up or transport your keys at a later time.

[✓] Include all extended properties.



Certificate Import Wizard

Certificate Store

Certificate stores are system areas where certificates are kept.

Windows can automatically select a certificate store, or you can specify a location for the certificate.

○ Automatically select the certificate store based on the type of certificate

● Place all certificates in the following store

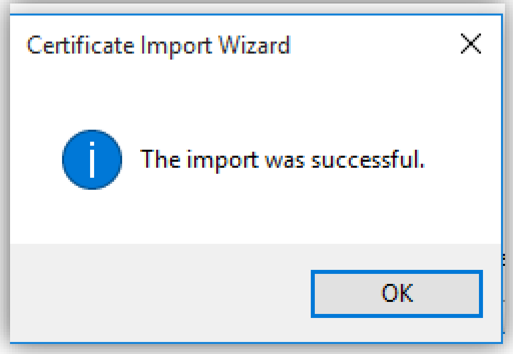
Certificate

Select Certificate Store

Select the certificate store you want to use.

- Personal
- Trusted Root Certification Authorities
- Enterprise Trust
- Intermediate Certification Authorities
- Trusted Publishers
- Untrusted Certificates

☐ Show physical stores



Certificate Import Wizard

The import was successful.