



Customer Relationship
Management

CRM WebServices Administrator Guide

Version 14.1

Notices

Copyright © 2004–2022. Aurea Software, Inc. (“Aurea”). All Rights Reserved. These materials and all Aurea products are copyrighted and all rights are reserved by Aurea.

This document is proprietary and confidential to Aurea and is available only under a valid non-disclosure agreement. No part of this document may be disclosed in any manner to a third party without the prior written consent of Aurea. The information in these materials is for informational purposes only and Aurea assumes no responsibility for any errors that may appear therein. Aurea reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of Aurea to notify any person of such revisions or changes.

You are hereby placed on notice that the software, its related technology and services may be covered by one or more United States (“US”) and non-US patents. A listing that associates patented and patent-pending products included in the software, software updates, their related technology and services with one or more patent numbers is available for you and the general public’s access at <https://markings.ip-dynamics.ai/esw/> (the “Patent Notice”) without charge. The association of products-to-patent numbers at the Patent Notice may not be an exclusive listing of associations, and other unlisted patents or pending patents may also be associated with the products. Likewise, the patents or pending patents may also be associated with unlisted products. You agree to regularly review the products-to-patent number(s) association at the Patent Notice to check for updates.

Aurea and Aurea Software are registered trademarks of Aurea Software, Inc. in the United States and/or other countries. Additional Aurea trademarks, including registered trademarks, are available at: <https://www.aurea.com/legal/trademarks/>. Jive is a registered trademark of Jive Software, Inc. in the United States and/or other countries. Additional Jive trademarks, including registered trademarks, are available at: <https://www.jivesoftware.com/legal/>.

Table of Contents

Preface.....	4
About this documentation.....	4
Notation conventions.....	4
Aurea global support.....	5
 Chapter 1: Introduction.....	 6
 Chapter 2: Product Overview.....	 7
Product artifacts.....	7
Architecture Overview.....	8
Major changes with CRM.webservices 8.0.....	8
New Features in Service Pack 2.....	9
 Chapter 3: Installation and configuration.....	 11
 Chapter 4: Services.....	 12
Common Services.....	12
XML Service.....	14
Custom Services (CRUD Services).....	15
Webservices configuration to activate HTTPS.....	15
 Chapter 5: Tools.....	 17
Service Generator.....	17
Test Client.....	24
Further Tools for testing and troubleshooting.....	26
 Chapter 6: Examples.....	 27

Preface

For details, see the following topics:

- [About this documentation](#)
- [Notation conventions](#)
- [Aurea global support](#)

About this documentation

This guide is part of the documentation set for Aurea CRM.

Notation conventions

This document uses the following notation conventions:

Convention	Meaning
Fixed-width	<code>Fixed-width</code> font indicates code, path names, file names, environment variable names, parameter names, command names, machine names, URLs.
Bold Fixed-width	Bold Fixed-width font is used to indicate user input or to emphasize certain lines of code.
<i>Italic Fixed-width</i>	<i>Italic Fixed-width</i> font indicates a placeholder for which you must supply a value.
Bold Sans serif	Bold sans serif typeface indicates the names of graphic user interface elements such as dialog boxes, buttons, and fields.
<i>Italic serif</i>	In text, <i>italic serif</i> typeface indicates the first use of an important term. The term is defined in the glossary.
Underlined	Underlined text in command lines and parameter descriptions indicate that you only have to enter the underlined part of the command or parameter name. For example, if you use the <u>-LOGFILE</u> parameter in a command, you only need to enter -LOGF.
[]	Brackets enclose optional arguments.
{ a b c }	Braces enclose two or more items. You can specify only one of the enclosed items. Vertical bars represent OR separators. For example, you can specify a or b or c.

Convention	Meaning
...	Three consecutive periods indicate that you can repeat the immediately previous item. In code examples, they can be horizontal or vertical to indicate omissions.
Menu > Choice	An angle bracket between two menu items indicates that you should choose an item from a menu. For example, the notation File > > Exit means: "Open the File menu and choose Exit ."
>>	Links to related information in other chapters or documents are indicated using the >> symbol.

Aurea global support

If you encounter a problem while using an Aurea product or require assistance with downloading the software or upgrading a product release, please open a ticket on [Aurea Support Central](#). Preferably, search the articles on the [Aurea Knowledge Base](#) for solutions to your issues before opening a ticket.

Information about the support organization is available on Support Central. The product documentation is available at <https://help.aurea.com/crm/#>.

For information about purchasing an upgrade or professional services, contact your account executive. If you do not know who your account executive is, or for other queries, [contact us](#) through our [website](#).

1

Introduction

CRM.webservices is a library of SOAP and RESTful web services allowing access to update.CRM business logic and data.

The web services can be used to integrate any SOAP-capable or REST-capable 3rd-party application with ease.

From a technical point of view CRM.webservices are based on the **Microsoft Windows Communication Foundation (WCF)** and on the Microsoft .NET Framework 4.5.

CRM.webservices are aimed at application-to-application (for example, displaying CRM data on your corporate extranet portal) as well as application-to-platform integration projects, such as SOA initiatives.

It is perfectly suited to connecting your CRM system to the rest of the enterprise using available ESB or other Middle-ware solutions.

It is important to note that the web services are only intended to provide services. If you want to consume 3rd-party services "from within" the update.CRM application, webservices is not (necessarily) a requirement.

This document gives an overview of the architecture of CRM.webservices and contains tips and tricks on how to use CRM.webservices.

This document does not cover the description of technologies such as SOAP, REST, WCF, etc. For further details refer to the documentation of these technologies.

Please contact the update support center for feedback on this document. This document is also available in the download section of the support area of <https://support.aurea.com>.

2

Product Overview

General Information and Product Overview

Product artifacts

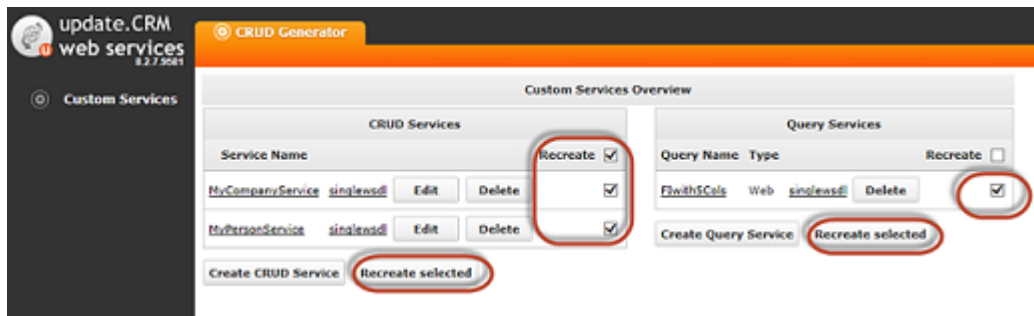
Learn about the product artifacts.

Basically the product consists of the following web services libraries:

1. Common Services
2. XML Service
3. Custom Services

Additionally the following tools are shipped with the product

1. Examples
2. Under some circumstances it is unavoidable that CRM.webservices contain breaking changes and therefore it might be necessary to recreate the services. To make the process easier, it is possible to select multiple services and recreate them all at once.

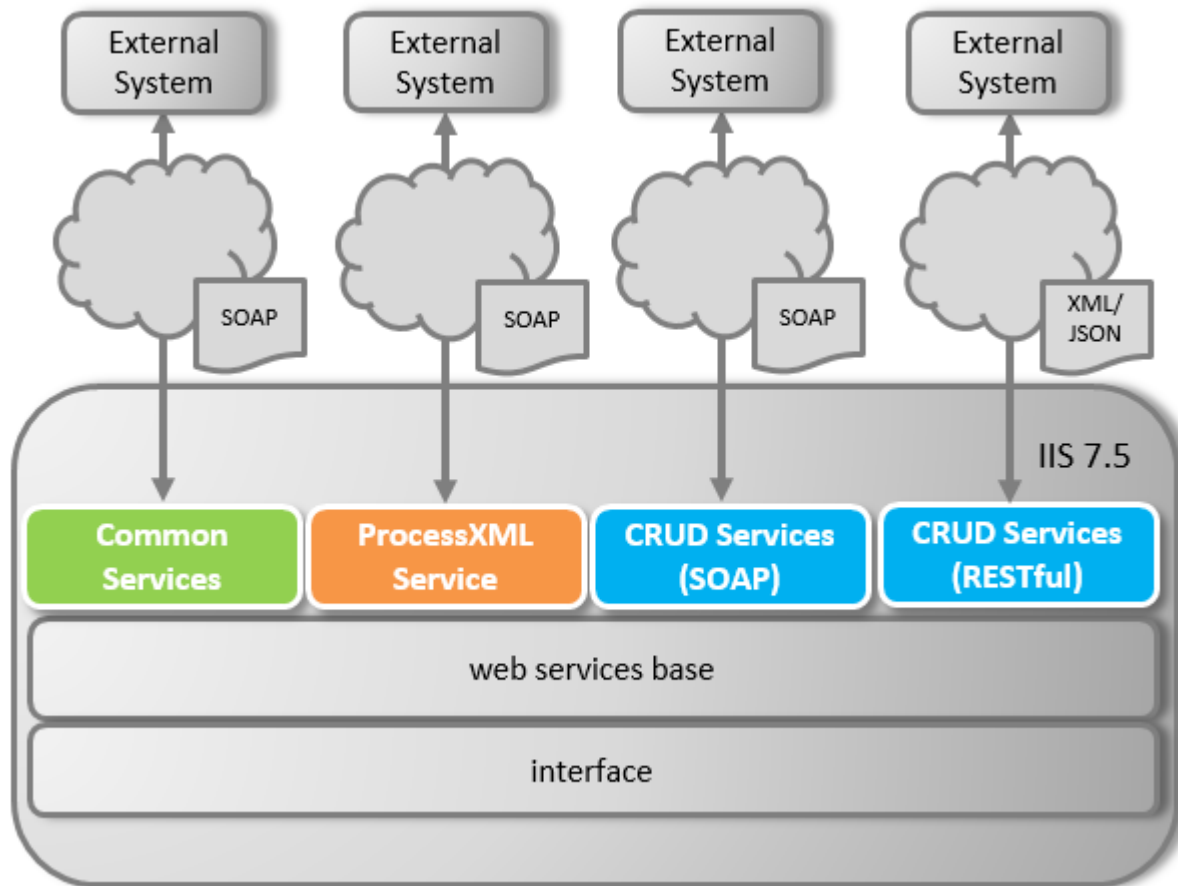


Recreating services:

- Test Client
- Service Generator

Architecture Overview

CRM.webservices accesses AureaCRM business logic and data via CRM.interface.



All services are integrated in the rights and roles framework of the CRM system. Thus, any access limitations you may have defined in your system are automatically and transparently respected by the webservices.

Major changes with CRM.webservices 8.0

This topic informs about the major changes in CRM.webservices 8.0.

Note: If you are not familiar with the predecessor of CRM.webservices you may confidently skip this chapter.

From a technical point of view the main difference is that the CRM.webservices are no longer ASP.NET web services but based on WCF. The change of the technology stack causes some differences in creating client application for CRM.webservices – these differences are outlined in chapter [Examples](#) below.

Beyond that we introduced a much easier and sophisticated way of handling logins, see [Login and session cookie management](#) for further information.

In addition to SOAP CRUD endpoints it's now also possible to create RESTful CRUD services.

Additionally you might have noticed that the "Toolbar" Service is no longer part of the web service library – the reason for that is, that the functionality of the is now integrated into update.CRM web.

Migrating webservices from update.seven to Aurea.CRM

Services created with update.seven webservices can be recreated to be compatible with update.CRM by copying the `ConfigFile.xml` from version 7.0 to the `\web` directory of CRM.webservices and executing the `recreate` function in the service builder.

Note: It is possible that the recreation fails, if your update.seven service references fields which are no longer present in update.CRM or due to other data model changes.

New Features in Service Pack 2

Learn about the new features in SP2.

Note: These features are available in the latest hotfix of CRM.webservices for Service Pack 1 (version 8.1.7.6463) as well.

Restful CRUD services

It is now possible to not only to create SOAP endpoints, but also RESTful CRUD services. See [REST Support and REST Message format](#) for further information.

New CRUD services options

With this version we introduced a couple of new options for generating CRUD services. These options are intended to support client systems, which are not capable to handle complex data types or session cookies.

Generate methods with credentials

Some client applications are not able to handle session cookies – with this option it is possible to create CRUD methods, which implement the CRM-Login (user, password and language id) as additional parameter.

Generate simple conditions

Some client applications are not able to handle recursions in WSDL – with this option it is possible to create simple condition objects without recursions.

Generate service namespaces

You can now define if the service contract should be created with namespaces when a webservice is created.

REST Support and REST Message format

This option allows for additionally creating a RESTful service endpoint. RESTful services expose resources via URIs, and clients access these resources using the four HTTP verbs.

Note: For CRM.webservices only GET and POST are used.

For example - using GET for QueryCompany would require mapping objects describing search conditions or sort criteria to parts of query string because GET doesn't accept complex parameters. And e.g. conditions can be quite complex the generation and maintenance of such an URL can be a nightmare. An alternative would be encoding those objects into string for example and send this as query parameter. But that would require client applications to call additional methods before sending their request to the webservice endpoint and require an additional decoding step at the service endpoint.

CRUD Services for ES, R2 and R3

CRM.webservices can now also be created for the Central login configuration (ES), Login role (R2) and Rep role (R3) info areas.

Note: The following tables are still treated as internal and therefore cannot be exposed as CRUD services:

- T0 (Stationconfiguration), T1 (Usertracking), T2 (Sessiontracking), T3 (Processtracking), T5 (ProtocolRead), T6 (ProtocolWrite)
 - US (LoginFile)
 - FT (Format)
-

Recreation of Services

Under some circumstances it is unavoidable that CRM.webservices contain breaking changes and therefore it might be necessary to recreate the services. To make the process easier, it is possible to select multiple services and recreate them all at once.

Note: We explicitly mention it in the Bug Fix list, if the installation of a version requires the recreation of CRUD services.

3

Installation and configuration

Learn to install and configure CRM.webservices.

Below are the steps for installation and configuration:

Installation

Aurea software AG provides an installation wizard that guides you through the installation of CRM.webservices.

Basically the setup does not require any manual input.

Note: If you are installing CRM.webservices on a machine with under Windows 7 64Bit and IIS 7.0 a warning message is displayed and you have to ensure that you enable anonymous access in IIS.

– in course of the setup procedure an application pool (`CRMwebservices`) and a virtual directory in IIS is created.

If a matching update.CRM win installation is found, the data base connection settings are copied from this installation to the CRM.interface installation. If such an installation is not found you have to manually define the connection string to your update.CRM data base in the `mmdb.ini`

Note: By default the `mmdb.ini` is located in `C:\Program Files\update.CRM\CRM.webservices\web\system\sys`

Configuration and Logging

Basically no manual configuration is required after running the installation wizard. CRM.webservices lives in the update.CRM server framework as a plug-in and is based on CRM.interface. Therefore it's configured via:

- the `settings.xml` of the framework (`\web`) and
- additionally via the `settings.xml` of the framework (`\web\system`)

CRM.webservices generates two log files:

1. `webservices.log`
2. `u8_interface.log`

4

Services

Learn about the services of CRM.webservices.

Below topics describe the services:

Common Services

These services perform generic tasks such as authentication management, session management, user messaging and metadata extraction / manipulation.

Many of the more complex services rely on these, and they are intended be re-used when developing or integrating 3rd party applications.

One of the services provided as part of the Common Services is known as Query2Service. It can be called from within the administration interface and allows the generation and deployment of **new, custom services** on the basis of a query definition, in effect **making any CRM query available via its own, specific service** (e.g. a "Top 10 Opportunities" service). This is particularly useful in portal integration projects.

AddService

```
public string[] AddService(string service)
```

`AddService` creates a common CRUD service for a business object. The generated service can then be used to access data of this business object and all of its related business objects.

Note: Please note that services created directly using this method is not displayed in the Custom CRUD interface.

AddWebQueryAsService

```
public string[] AddWebQueryAsService(string name)
```

`AddWebQueryAsService` allows for executing a web query.

AddWinQueryAsService

```
public string[] AddWinQueryAsService(string name)
```

`AddWinQueryAsService` allows for executing a win query.

CatalogValues

```
public string[] CatalogValues(string tableName, string fieldName)
```

`CatalogValues` returns the catalog values for field of an business object (table).

CatalogValuesById

```
public string[] CatalogValuesById(long catalogId)
```

`CatalogValuesById` returns the available catalog values of a certain catalog with a given ID.

CatalogValuesDetailed

```
public object[] CatalogValuesDetailed(string tableName, string fieldName, string includeLocked)
```

`CatalogValuesDetailed` delivers the catalog details for the given business object, field and a certain locked status.

IsLoggedIn

```
public bool IsLoggedIn()
```

`IsLoggedIn` indicates if a user is currently logged in.

Login

```
public string[] Login(string username, string password, long languageId)
```

`Login` performs a login.

Logout

```
public void Logout()
```

`Logout` performs a log out.

Merge

```
public string[] Merge(string tablename, string mode, string verbose, string sourceRecId, string sourceExtSystem, string sourceExtKey, string destinationRecId, string destinationExtSystem, string destinationExtKey)
```

`Merge` allows for merging of two records. The records can be defined either by their record Id or by an External System and Key combination. The tablename is the name of the business objects being merged. The mode can be source or destination.

MetaInfo

```
public string[] MetaInfo(string tableName)
```

`MetaInfo` returns the meta information for a business object (table). The `MetaInfo` contains full details of the contents of a business object, its fields (type, size, etc...), indexes, parents, children, etc....

SendMessage

```
public string[] SendMessage(string repId, string message)
```

`SendMessage` sends a message to the Rep with the given Id.

Status

```
public string[] Status()
```

`Status` returns information about the CRM.interface settings, logged in users, active processes on the server and other information related to the server.

TableNames

```
public string[] TableNames()
```

`TableNames` returns a list of all business objects (tables).

XML Service

The XML Service acts as a SOAP wrapper for CRM.interface XML.

The service takes an XML structure in the CRM.interface "dialect" as input, and returns the raw XML response for the given operation.

This service is fully compatible with CRM.interface and might be the right choice in performance-intensive scenarios (i.e. when you need maximum control over query syntax).

Please refer to the Technical Guide of CRM.interface for a detailed description about the architecture and a full syntax reference for the interface XML dialect.

```
ProcessStringResponse ProcessString(ProcessStringRequest request)
```

```
ProcessNodeResponse ProcessNode(ProcessNodeRequest request)
```

You can either pass plain text (using `ProcessString`) or an XML document to the XML service (using `ProcessNode`) as illustrated in the code fragment of our test client tool.

```
XmlNode nodeResponse;  
if (_properties.TransferMode == TransferMode.String) nodeResponse =  
xmlClient.ProcessString(request);  
else  
  
{  
    var requestDoc = new XmlDocument();  
    requestDoc.LoadXml(request);  
    nodeResponse = xmlClient.ProcessNode(requestDoc.DocumentElement);  
}
```

Custom Services (CRUD Services)

Learn about the custom services.

The CRUD (**Create**, **Read**, **Update** and **Delete**) services are basic functionality of CRM.webservices. The purpose of the CRUD services is to expose update.CRM business logic and data via SOAP or RESTful services. CRUD Services are not delivered out-of-the-box. Instead, they can be generated as needed with just a few mouse clicks, using the CRM.webservices administration interface ([Service Generator](#)). It ensures that any customization of the data model is reflected in the interfaces you deploy.

Any entity in the update.CRM database can be exposed as a complex SOAP data type, and accessed by third-party applications for both read and write operations using specialized SOAP methods. As of Service Pack 2 of CRM.webservices it is additionally possible to generate REST service endpoints.

In addition, you can create custom CRUD services which you can customize the data types they contain. For further information see <https://support.aurea.com>.

Webservices configuration to activate HTTPS

Learn how to activate https for CRM webservices.

CRM.webservices is pre-configured for plain HTTP only.

Below is the configuration to activate HTTPS.

web.config (server-side)

- **HTTP (default)**

```
<security mode="None"/>
<endpoint.... binding="mexHttpBinding">
<serviceMetadata httpGetEnabled="true">
```

- **HTTPS example**

```
<security mode="Transport"/>
<endpoint.... binding="mexHttpsBinding" (2 times)
<serviceMetadata httpGetEnabled="true" httpsGetEnabled="true">
```

update.web.services.test.exe.config (client-side test tool)

- **HTTP (default)**

```
<basicHttpBinding>
<binding name="Basic" allowCookies="false" />
<binding name="BasicHttpBinding_IXml" maxReceivedMessageSize="5000000"
/>
</basicHttpBinding>
```

- **HTTPS example**

```
<basicHttpBinding>
<binding name="Basic" allowCookies="false" />
<binding name="BasicHttpBinding_IXml" maxReceivedMessageSize="5000000">
<security mode="Transport">
<transport clientCredentialType="None" proxyCredentialType="None" realm=""/>
<message clientCredentialType="Certificate" algorithmSuite="Default" />
</security>
</binding>
</basicHttpBinding>
```

5

Tools

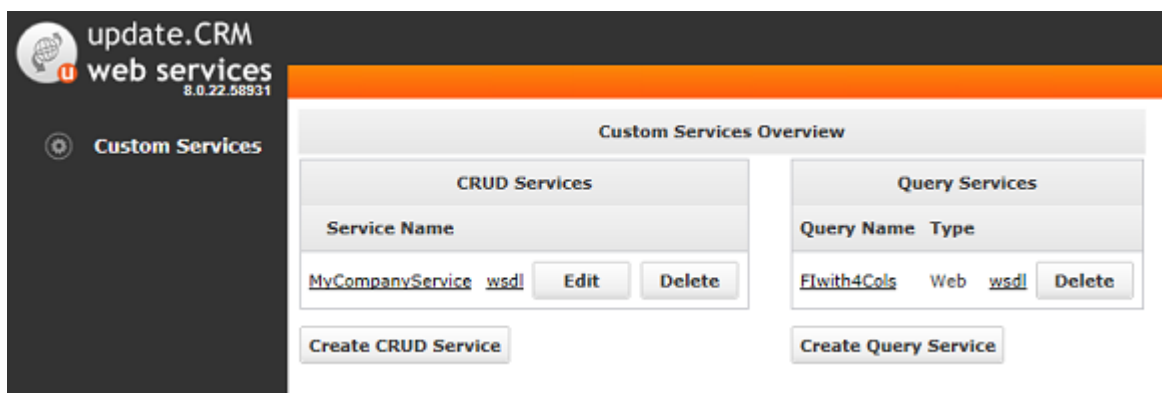
Learn about the tools used in CRM.webservices.

Below topics describe the tools:

Service Generator

Learn about the service generator tool.

The (CRUD) Services Generator is a web application which allows for generating CRUD Services. The web-based administration interface gives a rapid overview of all deployed services, the opportunity to test individual services as well as access to service generation functionality.

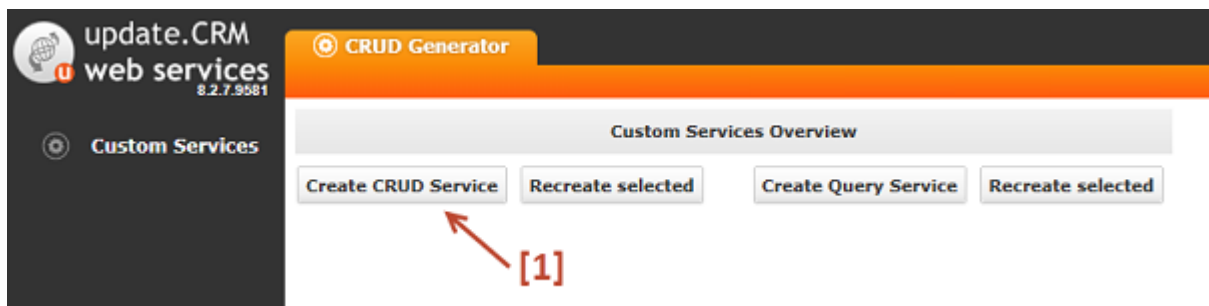


Service generator

The Custom Service page lists the currently configured Custom Services. With this page you can create, modify and delete services.

Creating a new CRUD service

To create a custom CRUD service, click on the "Create CRUD Service" button [1].



Generating a CRUD service (1)

update.CRM web services 8.2.7.9581

Custom Services

CRUD Generator

Create a new service

General Settings

MyCompanyService Service Name

☒ Generate abstract types

☐ Generate methods with credentials

☐ Generate simple conditions

☒ REST support

Auto REST messages format

☒ Generate service namespace

Cancel Continue [2]

Generating a CRUD service (2)

Then enter the Service Name, e.g. MyCompanyService.

The CRUD generator provides a couple of options to determine how the service should be created.

Option	Remark
Generate abstract types	This option indicates if abstract base class types are used in the generated service code. Some programming languages do not support the use of abstract data types. See chapter Conditions for further information on this topic.
Generate methods with credentials	This option indicates if additional CRUD methods should be created, which implement the CRM Login (user, password, language id) as additional parameter.
Generate simple conditions	This option indicates if conditions without recursions should be created.
REST support	This option indicates if an additional RESTful service endpoint should be created.

Option	Remark
REST message format	The option allows for choosing the message format (automatic, XML, JSON). "Automatic" means, that the response format is identical to the format of the request.
Generate service namespaces	Some Clients cannot handle an empty target namespace attribute in the WSDL file. With this option, a target namespace is generated.

CRUD service options

On the next page you can choose a business object (Info Area) [3]. If data type is left blank the data type has the same name as the Info Area.

Afterwards click "**generate type**" [4], then define the fields you want access with your service [5].

update.CRM web services 8.2.7.9581

Custom Services

CRUD Generator

Create a new service

Define data types for service: MyCompanyService

Company InfoArea [3]

DataType Name

If the datatype name is left blank the InfoArea name is used by default

Create type [4]

Currently defined types

Type Name	Base Info area	Fields	Links
Company	Company	Edit	Edit Delete

Back Cancel Continue [5]

Generating a CRUD service (3)

☐ Field Id	Name	Type	Custom Name
☒ 0	CoGrp	UInt16	
☒ 1	CoNo	UInt32	
☒ 2	CompanyField	String	
☒ 3	Synonym	String	
☐ 4	Division	String	
☒ 5	Country	Catalog	

Generating a CRUD service (4)

Clicking the check box in the header toggles on or off the selection for all fields. Selected fields can be given a custom name.

If required for your business needs add additional business objects (Info Areas), e.g. add the person info area as in our example below. Afterwards you can edit the links between the info areas [6].

Type Name	Base Info area	Fields	Links
Company	Company	Edit	Edit Delete
Person	Person	Edit	Edit Delete

Generating a CRUD service (5)

Data types may have many links to other data types, however if a data type is not defined in a service no links to that data type are possible. Via the "Show/hide disabled links" toggle button you can switch between displaying these disabled links and hiding them.

If no custom name for a link is specified a default link name is generated.

It is also possible to mark the link as optional; in this case the link record does not have to exist for the record to be created.

Generating a CRUD service (6)

Afterwards you have to define the methods of your service [7].

The service generator supports for different method types: Insert, Query, Update and Delete.

You can either generate the default methods [8] or generate custom methods. Each method name must be unique in the service. At least one method must be defined before moving on to the next step.

Method Name	Type	Data Type	Delete
QueryCompany	Query	Company	Delete
InsertCompany	Insert	Company	Delete
UpdateCompany	Update	Company	Delete
DeleteCompany	Delete	Company	Delete

Generating a CRUD service (7)

After creating the methods an overview page is displayed [9], where you finally can create your service [10].

update.CRM web services 8.2.7.9581

CRUD Generator

Create a new service

Review definition for service: MyCompanyService

Use abstract types	True
Generate methods with credentials	False
Generate simple conditions	False
REST support	True
REST messages format	Auto
Generate service namespace	True

Data Types

Type Name	BaseInfoArea
Company	Company
Person	Person

Methods

Method Name	Type	DataType
QueryCompany	Query	Company
InsertCompany	Insert	Company
UpdateCompany	Update	Company
DeleteCompany	Delete	Company

Back Cancel **Generate Service** ← [10]

Generating a CRUD service (9)

update.CRM web services 8.2.7.9581

CRUD Generator

Custom Services Overview

CRUD Services

Service Name	Recreate	Edit	Delete
MyCompanyService singlewsdl	☐	☐	☐

Create CRUD Service Recreate selected [11]

Create Query Service Recreate selected [12]

Generating a CRUD service (10)

After creation of the service you can view the service definition and the WSDL via clicking the links in the overview page [11].

Creating a new query service

In order to create a Query Service click the "**Create Query Service Button**" [12].

The custom query service allows for execution of either a web query or a win query. Queries with parameters are supported, too. If a non-existing query is referenced the service is not created.

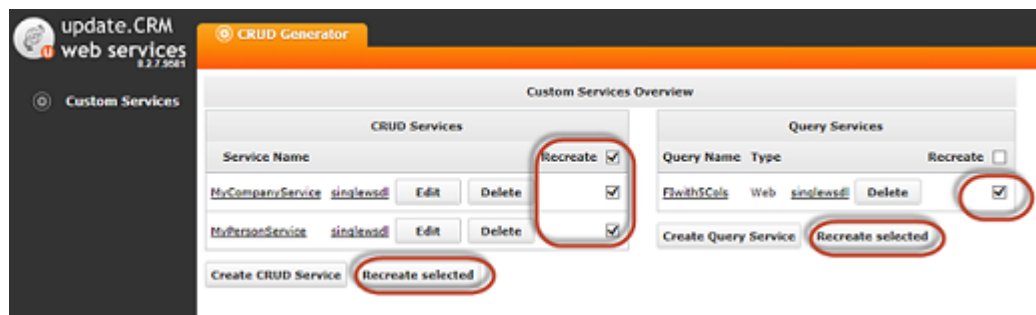
Query definition in web

Generating a query service

Note: Queries stored in web designer cannot be used by webservice.

Recreating Services

Under some circumstances it is unavoidable that CRM.webservices contain breaking changes and therefore it might be necessary to recreate the services. To make the process easier, it is possible to select multiple services and recreate them all at once.



Recreating services

Test Client

Learn about the test client tool.

The CRM.interface and CRM.webservices test client allows for testing the *ProcessXml* Service (and CRM.interface). Additionally it is possible to test RESTful services.

In the property pane you can basically define to which service endpoint the request should be sent and how to authentication mode you want to use.

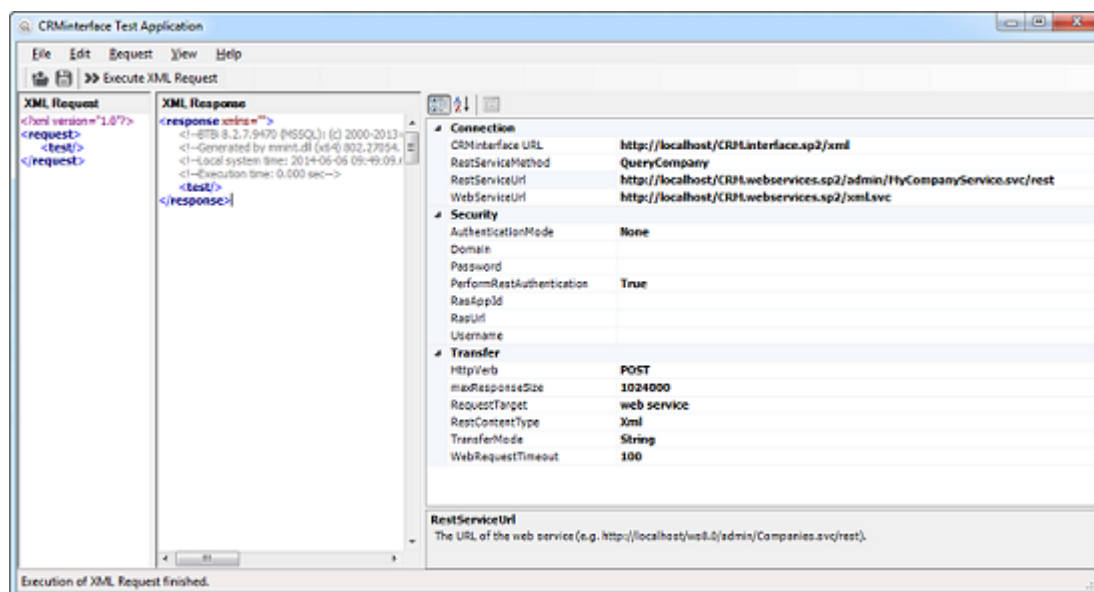


Figure 17, Test Client (1)

It is possible to insert Query/Import and Condition fragments, which might be helpful if you are not familiar with the CRM.interface query language.

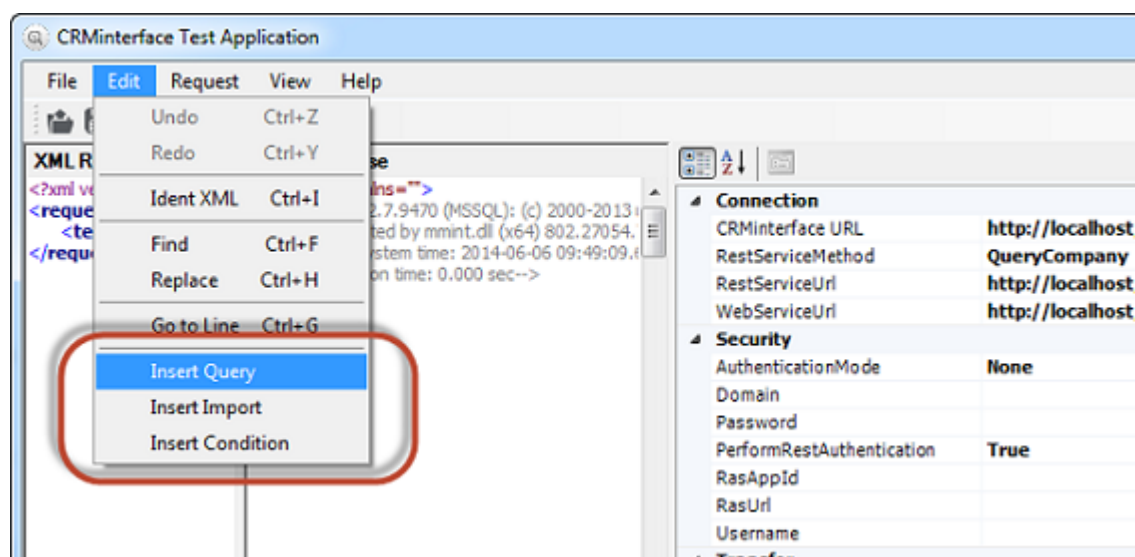


Figure 18, Test Client (2)

Note: When testing **XML Services** you have to specify the URL of the XML service (e.g. `http://localhost/CRM.interface.sp2/xml`) and to set the request target to "web service".

When testing **REST Services** you have to specify:

- RestServiceUrl (e.g. `http://localhost/CRM.webserv.sp2/MyCompanyService.svc/rest`)
- RestServiceMethod (e.g. QueryCompany)
- Http Verb (set it to POST, for create/update/delete and to GET otherwise)
- Request Target (set it to REST service)
- Request Content Type (set it to XML if you want to use the example infrastructure)
- Username, Password (username, password for update.CRM)

You can use the request templates -which are provided with the RESTSample- in order to test your REST service. These templates are intended to work with a CRUD service named "Companies", which contains Company and Person objects.

Hint: By default the maximum message size which can be received by the test client is set to 50000000. If a test call fails with an error message "The maximum message size quota for incoming messages (50000000) has been exceeded." you need to change that limitation by modifying the value of `maxReceivedMessageSize` in the configuration file (`update.web.services.test.exe.config`) of the test client.

Further Tools for testing and troubleshooting

Some other tools which can be used to test CRM.webservices.

Visual Studio

The most comfortable way to consume CRM.webservices is by using **Visual Studio** and **C#**. We provide samples for all CRUD methods.

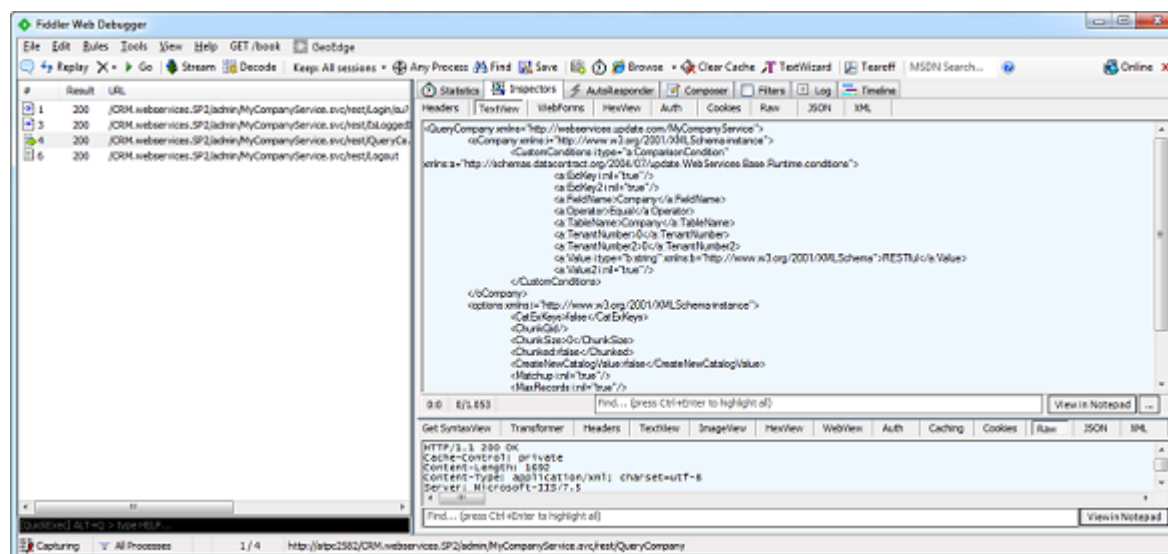
Soap UI

SoapUI is a helpful tool for testing

- REST services
- SOAP webservices based on a WSDL file
- Pure Soap Requests

Fiddler

One of the most important tools for getting an overview what's really going on between a client and the webservice is Fiddler.



6

Examples

Examples for webservices configuration.

In the "Samples" subdirectory you can find a Visual Studio 2012 solution containing six sample projects covering how to use CRUD web services (Insert, Query, Update and Delete), Custom Query service and an example illustrating the chunked read feature.

Additionally the solution contains the `CookieManagerLib` project with classes allowing for passing cookies between different instances of services (WCF equivalent of passing `CookieContainer` between instances in previous version).

General settings

To use these projects you have to follow the instruction below:

1. The service generator can be found at the following url:

`http://<your server>/<your path to web services>`

Open the service generator and create a Custom CRUD service, naming it "Companies" and using "Abstract Types".

Create "Company" and "Person" data types.

Edit the fields for both data types and select all available fields.

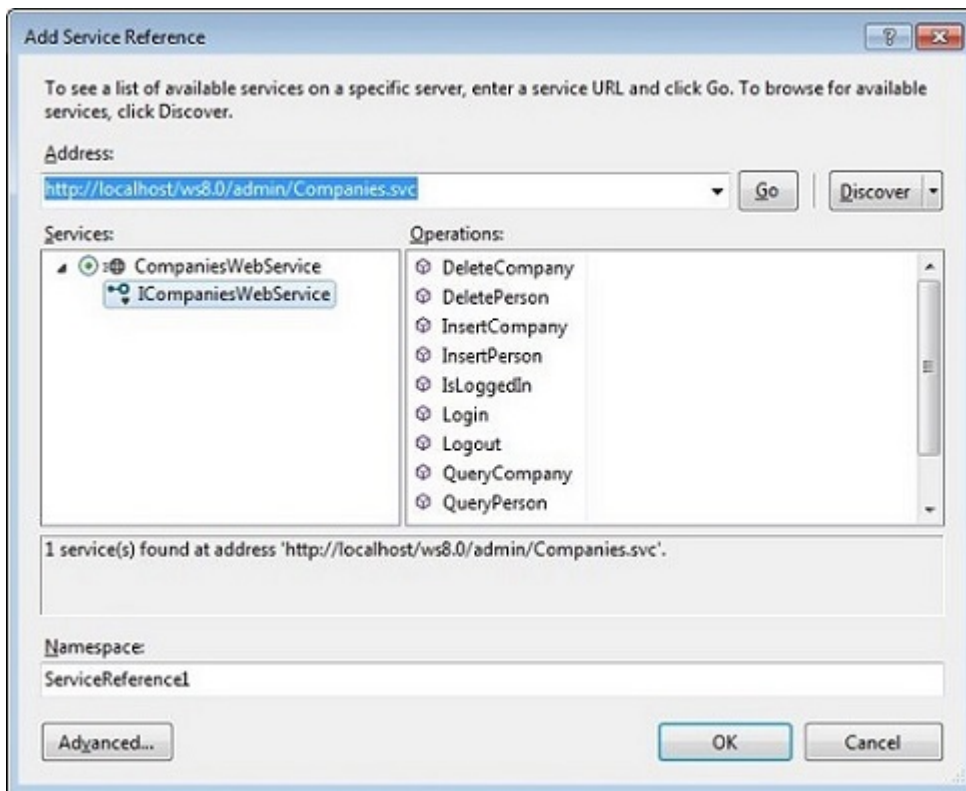
Edit the links for the Person data type and create a link to Company (Link Id, 0; Type, Parent).

Create the default methods for the Person and Company data types.

2. You need to update/recreate service references of each project to point them to Companies service you generated in previous step.

You can do it by one of the following ways:

- Deleting existing reference and using Visual Studio wizard via the "Add service reference" context menu where you have to enter URL to svc file of your newly created service.



Adding a Service Reference

- Modification of service configuration in `app.config` file (changes are in green font color)

```
<client>
  <endpoint address="http://localhost/ws8.0/admin/Companies.svc"
    binding="basicHttpBinding"
    bindingConfiguration="BasicHttpBinding_ICompaniesWebService"
    contract="CompaniesServiceReference.ICompaniesWebService"
    name="BasicHttpBinding_ICompaniesWebService"/>
</client>
```

- Setting the URL to the entry point of service in code

```
var client = new CompaniesServiceClient();
client.Endpoint.Address = new
EndpointAddress("http://localhost/ws8.0/admin/Companies.svc");
```

You can get the URL of your generated service by clicking the one of the links in the CRUD generator as depicted below.



Retrieving the URL of a generated service

Login and session cookie management

Each CRUD service and `CommonServicesWcf` has

- `Login`
- `Logout`
- `IsLoggedIn`

methods. But since WCF services using basic http protocol are stateless the next method called after `Login` returns error message "Session is null. Login missing?".

To have a possibility of sustaining session cookies between subsequent calls into the services we can use two approaches:

1. Configure the service to use cookie transmission between server and client. In `app.config` file we need to add the attribute `allowCookies="true"` attribute in binding section.

```
<bindings>
  <basicHttpBinding>
    <binding name="BasicHttpBinding_ICompaniesWebService" allowCookies="true"/>

  </basicHttpBinding>
</bindings>
```

2. Use WCF service's behavior extension infrastructure to intercept cookies returned by server and pass them between each service client configured to use that extension.

Note: This mechanism is equivalent of passing `CookieContainer` instances between Common Services and other services in `update.seven` webservice.

An example of the implementation of such behavior extension is located in the `CookieManagerLib` project and it's usage is illustrated in the `Custom Query` sample project.

Note: You have to remove the `allowCookies` attribute or set it to `false` otherwise behavior extension won't be able to intercept cookies.

Insert

This example shows how to create a company record and a person record linked to that company.

The program contains the following distinct parts.

1. We create an instance of the web service (`CompaniesWebServiceImpl`) which is used to perform a login and create records.
2. We create a `Company` object and assign it Company Name, any other data you require can also be added to the appropriate fields. The default Company name is "Test Company".
3. Using the `InsertCompany` method of `CompaniesWebServiceImpl`, we insert the Company into the database.
4. After the insert is completed we check that it was successful and if so we display the Id of the new Company record and continue creating a Person record to link to the Company. If the insert failed we display the error message.
5. We create a `Person` object and assign it a First and Last name, any other data you require can also be added to the appropriate fields. The default name is "Test Person". Set the link object `Company_0` to our Test Company.
6. Using the `InsertPerson` method of `CompaniesWebServiceImpl`, we insert the Person into the database.
7. After the insert is completed we check that it was successful and if so we display the Id of the new Person record. If the insert failed we display the error message.

Query

This example returns all company records matching the defined condition; as a search value we use "Test Company", the company we created in the insert example above.

1. We create an instance of the web service (`CompaniesWebServiceClient`) which is used to perform a login and to query for the records.
2. We create a `Sort` object; the example's default sort order is by the "Street" attribute of the returned Companies.
3. We create a `ComparisonCondition` object; this defines the query conditions we require. The sample's default condition is to find any companies called "Test Company". You can customize the condition to suit your database contents.
4. We create a `Company` object, for our query and assign our sort and condition objects to the `Sort` and `CustomCondition` attributes.
5. Using the `QueryCompany` method of `CompaniesWebServiceClient`, we query the database for Companies that match our condition.
6. After the query is completed we check that if it was executed successfully and if any matching companies are returned we display their Name and Street. If the query failed we display the error message.

Note: If only an object (Company, Person etc.) is specified without any conditions, ALL available records of this Info Area (table) are returned.

This sample has a simple condition; more complex conditions can also be used. See the [Conditions](#) section later in this chapter.

Update

In this example we update a company which satisfies the search condition. If more than one record is found, none of the records are updated and an error message, "Parent record not unique" is returned. If no record is found, a new record with the specified fields is created.

1. We create an instance of the web service (`CompaniesWebServiceClient`) which is used to perform a login and to update the record.
2. We create a `ComparisonCondition` object; this defines the query conditions we require. The example's default condition is to find any companies with a Synonym called "Test Synonym".
3. We create a `Company` object, for our update and assign our condition object to the `CustomCondition` attribute and set the street field to "New Test Street" to update the value.
4. Using the `UpdateCompany` method of `CompaniesWebServiceClient`, we update the Company that matches our condition.
5. The response indicates if the record that was updated or contains an error message if no record or more than one record was found.

To update multiple records it would be necessary to update every record individually.

Delete

This example deletes all matching company records from database.

1. We create an instance of the web service (`CompaniesWebServiceClient`) which is used to perform a login and to delete the records.
2. We create two instances of `ComparisonCondition`. We search for a company with a company name equal to "Test Company" and synonym equal to "Test Synonym". Combine the comparison conditions using a `LogicalCondition` with an And operation.
3. We create a `Company` object, for our delete and assign our logical condition object to the `CustomCondition` attribute.
4. Using the `DeleteCompany` method of `CompaniesWebServiceClient`, we delete Companies that match our condition from the database.
5. After the deletion is completed we check if it was successful and if any companies have been deleted we display their Record Ids. If the query failed we display an error message.

Note: Note: Use the delete method with caution, since – as the method name implicates – all matching data are deleted.

Custom Query

This example shows how to execute a (web) query via webservices. It is possible to pass parameters to the query.

1. Please create a query called "CompanyTest". Add the Company Info Area to this query, including some output fields (Company, Synonym, Street and City are used in the example). Add a (parameter) condition for companies by company name and click "Use as Parameter (requires user input)".
2. In the example we create a query service "CompanyTest" with the service generator create.
3. We create an instance of `CommonServicesWcfClient` and perform a login.
4. We create an instance of the web service (`CompanyTestQueryServiceClient`) which is used to execute the query. Due to usage of behavior extension intercepting cookies (see `app.config` file) the session created by Login method of `CommonServicesWcfClient` is maintained for the query service client.
5. We create an instance of `CompanyTest` which represents the query and set the value of our parameter to find the company called "Test Company".
6. Using the Query method of `CompanyTestQueryServiceClient`, we execute the CompanyTest query.
7. After the query is completed we check if it was executed successfully and if any companies are returned we display their Company, Synonym, Street and City attributes. If the query failed we display the error message.

Chunked Query

A chunked query allows for splitting the results of a query into parts and process them as such. The chunked query is initialized by setting the "Chunked" request option property to true; you can either provide your own unique identifier or retrieve the automatically assigned one from the response. To retrieve any subsequent chunks of the query you need to provide the identifier in the query request options.

Note: If you use the same `RequestOptions` object that is used for creating the chunked query, for a retrieving more chunks, you need to set the "Chunked" property to false and update the unique identifier if one has been automatically assigned. Otherwise you get an error saying that a Chunked operation with that identifier already exists.

The Chunked Query example shows how to get all company records in chunks of 10 records.

1. We create an instance of the web service (`CompaniesWebServiceClient`) which is used to perform a login and read the data.
2. We create a `Company` object, for our query.
3. We create `RequestOptions` object with `Chunked` property set to true and `ChunkSize` property set to 10.
4. Using the `QueryCompany` method of `CompaniesWebServiceClient` we query the database for first chunk of records (pass `RequestOptions` object as second parameter of `QueryCompany` method).
5. After the query is completed we check if it was executed successfully. If any matching companies are found we display their Name and Street. If the query failed we display the error message.
6. Before we can query for next chunk of records we need to set `ChunkQid` property of `RequestOptions` to the value of `ChunkQid` from response object and set `Chunked` property of `RequestOptions` object to false.
7. We continue to query for chunks of records and displaying Name and Street values until query won't return more data.

Tenant Catalogs

To create business objects with tenant catalogs you may use the following example code as a template:

```
// create tenant objects
Tenant oCountryTenant = new Tenant();
oCountryTenant.Field = "Country";
oCountryTenant.TenantNumber = 100;

Tenant oAreaTenant = new Tenant();
oAreaTenant.Field = "Area";
oAreaTenant.TenantNumber = 123;

// the tenant objects are put in an array
Tenant[] oTenants = new Tenant[] { oCountryTenant, oAreaTenant };

// create an object assigning the tenants array
// to the CustomTenants property
Company oCompany = new Company();
oCompany.CompanyField = "Test Company";
oCompany.Synonym = "Test Synonym";
oCompany.Country = "Austria";
oCompany.Area = "Test Area";
oCompany.CustomTenants = oTenants;
```

Conditions

CRM.webservices support two types of conditions

- Condition and
- ConcreteConditions.

Condition is an abstract base class type for ComparisonCondition and LogicalCondition, which implements the functionality to define complex conditions.

ConcreteConditions offer the same functionality but without the use of abstract classes.

Some program languages do not support the use of abstract types. You can therefore use this option to omit abstract types in your custom CRUD services.

The following example illustrates to implement conditions using both condition types. The example condition is to find Person objects with a First Name “test” and Last Name “person”.

The main difference between the conditions is ComparisonCondition and LogicalCondition have an Operator property, whereas ConcreteConditions has ComparisonOperator and LogicalOperator **properties**, only on these must be set to indicate what kind of condition the ConcreteConditions represents.

Note: The description for services with simple conditions is provided in a future revision of this document.

Using Condition

```
ComparisonCondition oCondition1 = new ComparisonCondition();
oCondition1.TableName = "Person";
oCondition1.FieldName = "FirstName";
oCondition1.Value = "test";
oCondition1.Operator = ComparisonOperator.Equal;

ComparisonCondition oCondition2 = new ComparisonCondition();
oCondition2.TableName = "Person";
oCondition2.FieldName = "LastName";
oCondition2.Value = "person";
oCondition2.Operator = ComparisonOperator.Equal;

LogicalCondition oConditions = new LogicalCondition();
oConditions.Operator = LogicalOperator.And;
oConditions.Conditions = new Condition[] { oCondition1, oCondition2 };

Person oPerson = new Person();
oPerson.CustomConditions = oConditions;

Response oResponse = companiesService.QueryPerson(oPerson, null);
```

Using ConcreteConditions

```
ConcreteConditions oConcreteCondition1 = new ConcreteConditions();
    oConcreteCondition1.TableName = "Person";
    oConcreteCondition1.FieldName = "FirstName";
    oConcreteCondition1.Value = "test";
    oConcreteCondition1.ComparisonOperator = ComparisonOperator.Equal;

ConcreteConditions oConcreteCondition2 = new ConcreteConditions();

    oConcreteCondition2.TableName = "Person";
    oConcreteCondition2.FieldName = "LastName";
    oConcreteCondition2.Value = "person";
    oConcreteCondition2.ComparisonOperator = ComparisonOperator.Equal;

ConcreteConditions oConcreteConditions = new ConcreteConditions();

    oConcreteConditions.LogicalOperator = LogicalOperator.And;
    oConcreteConditions.Conditions = new ConcreteConditions[] {
        oConcreteCondition1, oConcreteCondition2 };

Person oPerson = new Person();
oPerson.ConcreteConditions = oConcreteConditions;

Response oResponse = companiesService.QueryPerson(oPerson, null);
```

Parent Catalog condition

To create a condition to find a record by a catalog and its parent catalog use the value2 attribute of the comparison condition to specify the parent catalog value.

The following example code can be used as a template:

```
// condition with a catalog and it's parent catalog
ComparisonCondition oComparisonCondition = new ComparisonCondition();
oComparisonCondition.Operator = ComparisonOperator.Equal;
oComparisonCondition.TableName = "Interests";
oComparisonCondition.FieldName = "Interest";
oComparisonCondition.Value = "catalog value";
oComparisonCondition.Value2 = "parent catalog value";
```

External Key Catalogs

Allows catalog values are specified by external key (as opposed to their textual value). This can be set at request level and/or for individual conditions.

The following example shows how to use external catalog keys at the request level. All catalog values must be specified as external keys and all catalog values returned are their external key values.

The 2nd condition (oCondition2) in the example illustrates how to override that behavior for a single condition.

```
// create request options
RequestOptions oRequestOptions = new RequestOptions();
oRequestOptions.CatExKeys = true;
oRequestOptions.VisibleFields = "my catalog field";

// create conditions
ComparisonCondition oCondition1 = new ComparisonCondition();
oCondition1.TableName = "Company";
oCondition1.FieldName = "my catalog field";
oCondition1.Value = "external_key_1";
oCondition1.Operator = ComparisonOperator.Equal;

ComparisonCondition oCondition2 = new ComparisonCondition();
oCondition2.TableName = "Company";
oCondition2.FieldName = "my catalog field";
oCondition2.Value = "my catalog text value";
oCondition2.Operator = ComparisonOperator.Equal;
oCondition2.ExtKey = false;

LogicalCondition oConditions = new LogicalCondition();
oConditions.Operator = LogicalOperator.Or;
oConditions.Conditions = new Condition[] { oCondition1, oCondition2 };

// create an object to query
Company oCompany = new Company();
oCompany.CustomConditions = oConditions;

// create an object to query
Response oResponse = companiesService.QueryCompany(oCompany, oRequestOptions);
```

The following sample shows how to use external catalog keys at the condition level.

```
// create condition
ComparisonCondition oCondition = new ComparisonCondition();
oCondition.TableName = "Company";
oCondition.FieldName = "my catalog field";
oCondition.Value = "external_key_1";
oCondition.Operator = ComparisonOperator.Equal;
oCondition2.ExtKey = true;
```

Request Options

Chunked

Reading in chunks allows for more efficient retrieving of (possible) huge data sets and navigating in these data sets. First only the record-IDs are read (up to maxrecords). Subsequent queries return the next "page" (determined by `ChunkSize`) of records by referencing the original query via a query-ID (`ChunkQid`).

```
const long English = 1;
// Login
var companiesService = new CompaniesWebServiceImpl();
companiesService.Login("TestUser", "TestPassword", English);
Company company = new Company();
RequestOptions oRequestOptions = new RequestOptions();
oRequestOptions.Chunked = true;
oRequestOptions.ChunkSize = 10;
// get the first chunk of results
Response oResponse = companiesService.QueryCompany(company, oRequestOptions);
Console.WriteLine("Succeeded: " + !oResponse.Error);
// check the results, to see if the query has completed
if (!oResponse.Error)
{
    // output the first chunk
    int nCount = 0;
    const string outputFormat = "Company: {0}; Street: {1};";
    foreach (var companyInResponse in oResponse.Objects.Cast<Company>())
    {
        Console.WriteLine(outputFormat,
companyInResponse.CompanyField, companyInResponse.Street);
    }
    // **** important ****
    // update the request options for getting the next chunks
    // set returned qid and switch off Chunked attribute
    oRequestOptions.ChunkQid = oResponse.ChunkQid;
    oRequestOptions.Chunked = false;
    bool bContinue = true;
    // continue getting chunks while they are available
    while (bContinue)
    {
        // get the next chunk of results
        oResponse = companiesService.QueryCompany(company, oRequestOptions);
        Console.WriteLine("Succeeded: " + !oResponse.Error);
        // check the results, to see if the query has completed
        if (!oResponse.Error)
        {
            nCount = 0;
            foreach (var companyInResponse in oResponse.Objects.Cast<Company>())
            {
                Console.WriteLine(outputFormat,
companyInResponse.CompanyField, companyInResponse.Street);
            }
        }
        else
        {
            // if the query ends or there is an error output terminate the loop
            Console.WriteLine("ErrorText: " + oResponse.ErrorText);
            Console.WriteLine("Description: " + oResponse.Description);
            bContinue = false;
        }
    }
}
else
{
    // if there is an error with the initial query output the reason
    Console.WriteLine("ErrorText: " + oResponse.ErrorText);
    Console.WriteLine("Description: " + oResponse.Description);
}
```

Chunk size

Defines the maximum number of records to be read in one chunk.

For further details see chapter [Chunked](#) above.

ChunkQid

A unique ID for a "chunked read" operation, The `ChunkQid` (query id) can be used to read subsequent chunks of data.

For further details see chapter [Chunked](#) above.

CreateNewCatalogValue

An indicator of whether unknown catalog values should be automatically created. It is recommended that new catalog values is not created using this flag due to risk of unwanted duplicates. It works only in the catalog base language.

Matchup

An indicator of whether the matchup is to be done for records during an insert.

MaxRecords

The maximum number of records to be read in a query. The default value is 9999.

RecordId

A 64-bit record id specified in hexadecimal notation and prefixed with an 'x' character.

SkipRecords

Sets the number of records to be skipped, before returning records from a query.

The following example shows how to skip 10 records.

```
// create request options
RequestOptions oRequestOptions = new RequestOptions();
oRequestOptions.SkipRecords = 10;
// create an object to query
Company oCompany = new Company();
// create an object to query
Response oResponse = companiesService.QueryCompany(oCompany, oRequestOptions);
```

Note: A more performant way for paging through large result sets is using Chunked Queries (see chapter [Chunked](#) above).

Visible Fields

An indicator, which fields should be retrieved by the layer beyond web services.

RESTSample

This sample illustrates how to implement a client for RESTful service endpoints. The example loads the request templates (e.g. `queryrequest.xml`, `insertrequest.xml`, `json_queryrequest.txt` etc.) and sends it to the service.

Note: The example is intended to work with a CRUD Service called "Companies", which contains Company and Person objects.
