

POWER TO RUN YOUR BUSINESS



CRM

Sage CRM 6.0

Web Services Guide

© Copyright 2006 Sage Technologies Limited, publisher of this work. All rights reserved.

No part of this documentation may be copied, photocopied, reproduced, translated, microfilmed, or otherwise duplicated on any medium without prior written consent of Sage Technologies Limited.

Use of the software programs described herein and this documentation is subject to the End User Licence Agreement enclosed in the software package, or accepted during system sign-up.

Sage, and the Sage logo are registered trademarks or trademarks of The Sage Group PLC. All other marks are trademarks or registered trademarks of their respective owners.

Contents

Chapter 1: Introduction

Introduction to Sage CRM Web Services	1-1
Web Service Technology	1-1
CRM Web Service Capabilities.....	1-2
Chapter Summary	1-2

Chapter 2: Getting Started

Prerequisites.....	2-1
Platform Compatibility.....	2-1
Now you can... ..	2-1

Chapter 3: Working with CRM Web Services

Steps for Working with Web Services	3-1
Web Services User Setup.....	3-1
Specifying Web Service Configuration Settings.....	3-2
Accessing the WSDL File.....	3-4
Now you can... ..	3-5

Chapter 4: Objects and Functions Overview

Manipulating Records	4-1
Functions	4-1
Objects.....	4-1
Available Functions	4-2
Objects Exposed.....	4-4
Abstract Objects.....	4-5
crmrecordtype Object	4-6
Standard Objects	4-6
Selection Fields	4-7
Now you can... ..	4-10

Chapter 5: Preparing the Web Services Request: Examples

Complete Example	5-1
Sample Soap Requests	5-1
Sample Soap Request for Logon	5-1
Sample Soap Request for Logoff	5-2
Sample Soap Request for Delete	5-2
Sample Soap Request for Update	5-3
Sample Soap Request for QueryEntity	5-4
Sample Soap XML Representing a Company	5-4
Now you can	5-5

Chapter 6: Creating a CRM Web Services client with Microsoft Visual Studio 2005

The purpose of the client	6-1
Stage 1: Configuring your installation	6-1
Stage 2: Creating the project and adding a web reference	6-2
Stage 3: Using Visual C# to logon to CRM Web Services	6-3
Stage 4: Using Visual C# to query CRM Web Services	6-5
Stage 5: Using the information	6-7
Stage 6: Running the application	6-9
Now you can	6-10

Chapter 1

Introduction

This guide is for system administrators and experienced software developers.

Please note that while the document refers to Sage CRM, CRM, or the CRM system throughout, all functionality covered is also relevant to Sage Accpac CRM and Softline Accpac CRM.

We assume that you are:

- Familiar with modern programming principles.

Introduction to Sage CRM Web Services

Sage CRM's Web Service API (application programming interface) enables developers to manipulate CRM records remotely with SOAP (Simple Object Access Protocol) over HTTP using XML (Extensible Markup Language). It is possible to access a CRM server or a hosted system from a specified client machine (typically another server) in order to read, create, update, or delete records for each exposed entity, for example, Companies, People, Opportunities, or Cases.

The main steps involved in communicating with the Sage CRM Web Service are as follows:

- The WSDL (Web Service Description Language) is generated on the CRM server.
- The user then accesses the WSDL file from the client and prepares the request.
- The client machine passes the request with its parameters to the Web Service.
- The Web Service processes the request and sends a response to the client.
- The client receives the response synchronously, and it processes the data returned or it deals with the error.

Web Service Technology

Web Services represents a standardized method for integrating Web-based applications using XML, SOAP, and WSDL via an Internet protocol backbone. Web Service components work as follows:

- XML tags the data.

- SOAP transfers the data. (For a detailed account of SOAP, please refer to <http://www.w3.org/TR/SOAP>).
- WSDL describes the available services.

The technology allows organizations to exchange data without in-depth knowledge of each other's IT systems behind the firewall. It does not provide users with a GUI, which is the case with traditional client/server models. Instead, Web Services share business logic, data, and processes through a programmatic interface across a network. Developers can add the Web Service to a GUI, such as a Web page or an executable program, to provide users with the required functionality.

The technology makes it possible for different applications from different sources to communicate with each other without time-consuming custom coding. Due to the fact that all communication is in XML, Web Services do not limit the user to any one programming language.

CRM Web Service Capabilities

The ability to manipulate records remotely affords the following capabilities:

Hosted Environments. As well as manipulating records on a standard CRM server, Sage CRM Web Services is compatible with a hosted environment. Consequently hosted customers can leverage the technology and its capabilities.

Changing Data. The ability to add, update and delete records in the CRM database.

Integrate with third-party applications. Access to the Sage CRM Web Services API enables you to integrate third-party applications used within your organization, for example Accounting packages or ERP systems, with the Sage CRM server or hosted system.

Chapter Summary

The table below gives a summary of each chapter.

Chapter	Summary
Introduction	An overview of the CRM Web Service functionality.
Getting Started	Prerequisites for setting up Web Services.
Working with CRM Web Services	A step-by-step account of how to work with Web Services.

Chapter	Summary
Objects and Function Overview	An explanation of the Objects exposed by the Web Services API and Functions that can be invoked.
Preparing the Web Services Request: Examples	Sample Web Service requests.
Creating a CRM Web Service client with Microsoft Visual Studio 2005	A walkthrough of the process for creating a CRM Web Services client using Microsoft's Integrated Development Environment.

Chapter 2

Getting Started

In this chapter you will learn about:

- Prerequisites for Web Services.
- Platform compatibility.

Prerequisites

To set up Web Services, you will need:

- CRM installed on a server with a standard license key.

Platform Compatibility

All up-to-date development environments that are compatible with Soap 1.1 are compatible with Sage CRM Web Services. Supported environments include:

- Microsoft Visual Studio 2003 and later (C#, J#, VB.NET)

Now you can...

- Explain prerequisites for Web Services.
- List which platforms are compatible with Sage CRM Web Services.

Chapter 3

Working with CRM Web Services

In this chapter you will learn about:

- Setting up a user for Web Services.
- Specifying Web Services configuration settings.
- Accessing the WSDL file.

Steps for Working with Web Services

The following steps are involved in working with Web Services:

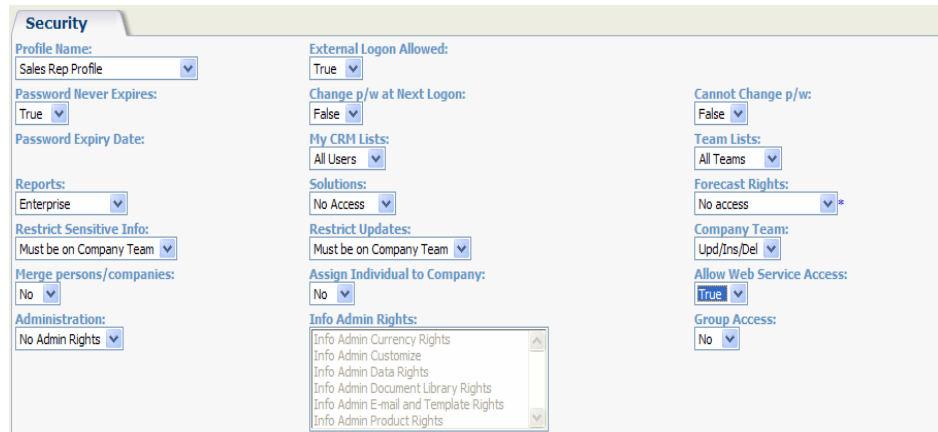
1. Setting up a Web Services user on the server. (Described in this chapter.)
2. Specifying Web Services configuration settings. (Described in this chapter.)
3. Accessing the WSDL file. (Described in this chapter.)
4. Preparing the request and submitting it to Web Services. (Described in the next chapter.)
5. Handling the response—returned values or error message. (Described in the next chapter.)

Web Services User Setup

Before Web Services can be accessed, a user needs to be set up for Web Services on the server.

To set up a user for Web Services:

1. Select Administration | Users | Users and find the user who you want to be able to access Web Services.
2. Select the hypertext link for the user and select the Change action button.
3. Scroll down to the Security Profile panel, set the Allow Web Service Access field to True.



User Details – Security panel

4. Select the Save button.

Note: Only one web service user can log on with the same ID at any given time. If a user tries to log on as another application, an error will be displayed informing the user that they should first log out. However, it is possible to log on to the desktop or from a device with the same ID while a web service application is running.

Moreover, it should be noted that the 6.0 feature, field level security, affects which entity fields can be accessed or updated using web service methods. So, for example, if a user is denied read access to a field by field level security, methods called by a web service session using that same user’s login details cannot return, update, or delete that field’s values. For more information on field-level security, refer to “Chapter 13: Field Customization” in the System Administrator Guide.

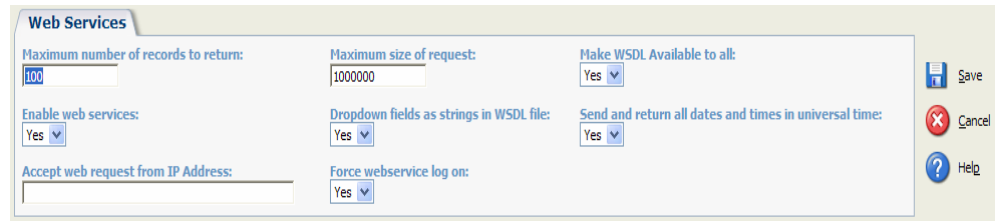
Specifying Web Service Configuration Settings

There are a number of Web Service configuration settings you need to be aware of.

To access Web Service configuration settings:

1. Select Administration | System | Web Services.

The Web Service setting page is displayed.



Web Services page

2. Select the Change action button.
3. Make the changes you require and select Save.

The table below explains the fields on the Web Service settings page.

Field	Description
Maximum Number Of Records To Return	The maximum number of records you want Web Services to be able to return at one time. This is used in conjunction with the query and queryrecord methods. The figure you set here is the number of records that will be returned in any one batch in response to a query. As each batch is returned, you will be prompted to call the next batch, until all of the records matching the query have been returned. If this field is set to 0, all records matching the query will be returned in a single batch.
Maximum Size Of Request	The maximum number of characters you want users to be able to send to Web Services.
Make WSDL Available To All	When set to Yes, the WSDL file can be viewed by anyone from: http://CRMservername/CRMinstallname/eware.dll/webservice/webservice.wsdl Users will not need to be logged in to view the file. It is accessible to anyone.
Enable Web Services	Set to Yes to enable the Web Services functionality. Set to No to disable Web Services.

Field	Description
Dropdown Fields As Strings In WSDL File	<p>Default is Yes. Drop down fields are displayed in the WSDL as enumerated types, for example comp_status as an enumeration with the drop down values in it. Please refer to the Objects and Functions chapter for more details.</p> <p>When set to Yes, makes the enumerated types "Strings". This is the recommended setting. This means that, for example, within Company the field comp_status now has a type of "String".</p>
Send And Return All Dates And Times In Universal Times	<p>When this is selected, all dates coming from the server will be set to universal time. Also, all dates coming to the web server will be offset from universal time. This is primarily important for migrations to the hosting service from different time zones.</p>
Accept Web Request From IP Address	<p>Specify the unique IP address that you want the WSDL file to be accessible from. When you do this, the Make Web Services Available To All field should be set to No.</p>
Force Web Service Log On	<p>If the connection between the web service client and the service is unexpectedly broken, that client remains logged on to the server hosting the service. This means that a new instance of the client will be blocked from logging on to the server. However, if you set the Force Webservice Log On setting to Yes, the old instance of the client is automatically logged out when a new instance attempts to log on. By forcing new log ons, this field prevents users from being "locked out" of a web service following a failed connection or unsuccessful log out.</p>

Accessing the WSDL File

As is the case with typical SOAP Web Services, CRM provides a Web Services description language file called a WSDL file.

To access this file on the CRM server:

- From the client machine, open the CRMWEBSERVICE.WSDL file from your install address, or from SageCRM.com, for example:

`http://CRMservername/CRMinstallname/eWare.dll/webservice/webservice.wsdl`

If you are connecting to a server with a version of CRM prior to version 57 SP1, you need to use the following address:

`http://CRMservername/CRMinstallname/eWare.dll/webservices/CRMwebservice.wsdl?version=57`

Or

`http://CRMservername/CRMinstallname/eWare.dll/webservices/CRMwebservice.wsdl?version=56`

Note: Internet Explorer 5.5 or higher is required to view this file. If a blank page is displayed when you open the file, right click on it and select View Source.

The CRM WSDL file describes all the APIs that CRM exposes, as well all the XML types that the APIs expect. The file also describes the server and location where those specific services can be found. Once the client has read and parsed the WSDL file, it can call the APIs in the same way as any typical function call. Since this data is passed and returned as XML, data can be easily interpreted and manipulated by the client.

Now you can...

- Set up a user for Web Services.
- Specify Web Services configuration settings.
- Access the WSDL file.

Chapter 4

Objects and Functions Overview

In this chapter you will learn about:

- How Sage CRM Web Services uses Objects and Functions to interact with the client machine and manipulate records.
- Objects exposed by the Web Service API.
- Functions that can be invoked to manipulate data.

Manipulating Records

Before you start working with CRM Web Services, you need to be familiar with all of the Functions that you can invoke to manipulate records, as well as the Objects (on which the functions are invoked) that are exposed in the API.

Functions

Functions are actions invoked from the client machine to perform certain tasks, such as adding, updating, or deleting information, on the server. Sage CRM functions are synchronous requests, and they are committed automatically. Once committed, the Sage CRM Web Service handles the request and returns a response. The client application then handles the response accordingly.

Note: All inserts should typically be performed on an entity basis. However, you can update a company (or person) along with address, phone, and e-mail information. This is to facilitate integration. In many systems, a single contact record represents company, person, phone, e-mail, and address information.

Objects

Objects are programmatic representations of data in the system. In Sage CRM, Objects represent main entities such as companies and people, as well as secondary entities such as addresses and products. Data is manipulated when the Web Service API interacts with Object properties, which represent fields in the entities.

Available Functions

All of the following Objects exposed are defined in the WSDL file.

Function	Description
logon	Logs onto the server and begins a session.
logoff	Logs off the server and terminates the session.
query	Executes a query on a specified Object based on a where clause and returns a record or record set that satisfies the query. Returns results in batches (the size of which is set in the Maximum Number Of Records To Return field at Administration System Web Services). Each batch is accompanied by a flag called More. If More is 'True', then there are more records waiting on the server for that query. Call 'Next' to get the next batch of data. If anything other than Next is called, the query is closed.
next	Will return the next batch of records matching a query. Each batch is accompanied by a flag called More. While More is 'True', you can continue to call 'Next' until all batches have been returned (i.e. until More is 'False').
queryentity	Returns a record if you supply an Object (For example Company) and an id. For example, queryentity(company, 42)
queryid	Returns AIS ID. Query the database with a Where clause, and a date and a number of IDs are returned, along with a series of flags on each to denote whether that record was created, updated or deleted since that date. This is very useful for data synchronization.

Function	Description
queryidnodate	Returns AIS ID. Query the database with a Where clause. This is useful where you need, for example, a set of company IDs but you do not want the overhead of getting all of the company data.
getmetadata	When you pass in a table name, this returns a list of CRM field types to provide metadata (for example fieldname, type) about the requested table.
getdropdownvalues	When you pass in a table, this returns the list of the drop-down fields in that table and the list of values that CRM expects for that field. This is important because CRM expects a given set of values for drop-down fields, so you need to be able to get these values programmatically.
add	Adds records or lists of records to a specified Object (for example Company). For example, add("company", NewCompany1, New Company2, New Company3).
addresource	Adds a user as a resource. This user is not a fully enabled user. The functionality exists purely to facilitate data migration.
update	Updates records or lists of records for a specified Object, for example Company.
altercolumnwidth	Used to resize a column width to ensure compatibility with third-party databases, for example ACT!.
delete	Deleted records or lists of records for a specified Object, for example Company. Note that you cannot delete records from the following tables, as they contain historical data: newproduct, uomfamily, productfamily, pricing, pricelist.

Function	Description
addrecord	Same as the add function except it has a different signature and it uses the lists of fields in the crmrecord type. See the section “CRMRecord Type” in this chapter.
queryrecord	Same as the query function except it has a different signature and it uses the lists of fields in the crmrecord type. See the section “CRMRecord Type” in this chapter.
nextqueryrecord	Will return the next batch of records matching a queryrecord. Each batch is accompanied by a flag called More. While More is ‘True’, you can continue to call ‘Next’ until all batches have been returned (i.e. until More is ‘False’).
updaterecord	Same as the update function except it has a different signature and it uses the lists of fields in the crmrecord type. See the section “CRMRecord Type” in this chapter.
getallmetadata	Returns a list of fields associated with all tables along with some type information.
getversionstring	Returns the version of CRM. For example, Version 5.8.

Objects Exposed

The following Objects are representative of CRM entities (main and secondary). If any custom entities are added to the CRM system, these entities are also available. Due to the fact that the WSDL is generated dynamically, any customizations made to the system – such as adding a new entity – are picked up each time the WSDL is refreshed at the client side.

Abstract Objects

Object Name	Description
ewarebase abstract	This is an abstract declaration from which all of the other CRM objects inherit.
idbase abstract	This is an abstract declaration from which all ID types inherit.
ewarebaselist	This represents a list of the abstract Objects above.
crmrecordtype	An enumeration that represents the types of a CRM field, i.e. string, datetime, integer, decimal. The value multiselectfield denotes a nested array of strings that represent the values of a multi-select field. The last option is crmrecord. This denotes a field type that contains other fields.
crmrecord	Contains an entity name and a list of objects of type recordfield that represent one record in the CRM database.
aisid	Contains the ID of the record, the created and updated date, and a flag to say whether that record was added, updated or deleted since the token that was passed to queryid.
multiselectfield	This type represents a multi select field from CRM. It contains a field name and an array of strings representing the values of the field in CRM. Note that these values are translations, as with the other fields.
recordfield	This represents a field in a database record. It has a name value and a type of crmrecordtype. It can also represent a nested structure. For example, the name of the recordfield within a company crmrecord could be person. The type would be crmrecord and the record property would contain a list of crmrecords - one for each person in the company.

crmrecordtype Object

The `crmrecordtype` object (with its associated `add`, `update`, and `delete` functions) provides a dynamic and flexible programming environment. Instead of querying an entity (for example, a company) and getting back a strongly typed (company) object, using the flexibility afforded by the `crmrecordtype` object, it is possible to query an entity and get back a list of fields that you can iterate through. This means that it is possible to specify which fields you want to get back in your query.

The ability to iterate through records provides programmers with a powerful and flexible interface. It allows for the dynamic addition of fields to the web services entities, and it removes the need for strongly typed objects in client applications. Code samples should be followed closely when performing these tasks.

The following is a query example that specifies a field list and an entity name, a where clause and an order by. Note that if you enter an `*` or leave the field list blank you will get all of the fields back.

```
Private static void CallQueryRecordOnCompanyEntity()  
{  
    String companyid = ReadUserInput("Please enter a company name:  
    ");  
    Queryrecordresult arestult =  
    Binding.queryrecord("comp_companyid,address", "comp_name =  
    'compol'", "company", "comp_companyid");  
}
```

Standard Objects

Object Name	Description
company	This Object represents the Company entity in CRM.
person	This Object represents the Person entity in CRM.
lead	This Object represents the Lead entity in CRM.
communication	This Object represents the Communication entity in CRM.
opportunity	This Object represents the Opportunity entity in CRM.
cases	This Object represents the Cases entity in CRM.
users	This Object represents the Users entity in CRM.

Object Name	Description
orderquote	This Object represents the Orders / Quotes entity in CRM.
lineitem	This Object represents the Lineitems entity in CRM.
opportunityitem	This Object represents the Opportunity Item entity in CRM.
currency	This Object represents the Currency entity in CRM.
address	This Object represents the Address entity in CRM.
phone	This Object represents the Phone entity in CRM.
email	This Object represents the Email entity in CRM.
newproduct	This Object represents the New Product entity in CRM.
uom	This Object represents the Unit of Measure entity in CRM.
uomfamily	This Object represents the Unit of Measure Family entity in CRM.
pricing	This Object represents the Pricing entity in CRM.
pricinglist	This Object represents the Pricing List entity in CRM.
productfamily	This Object represents the Product Family entity in CRM.

Selection Fields

Note: If you have drop-down fields as strings, these fields will not appear in the wsdl. As strings are the default option, these fields will not appear in a standard setup.

The following Objects represent CRM selection field values—like Active and Inactive for the Status field for example. In the WSDL file, an enumerated type for each field that contains values represents these values. There are several fields like this for each entity.

Note: Enumerated values like Additional Software Required or Customer Knowledge (listed below) are returned in the default system language.

```
<s:simpleType name="case_problemtyp">  
<s:restriction base="s:string">  
<s:enumeration value="Additional Software Required" />  
<s:enumeration value="Software Bug" />  
<s:enumeration value="Setup/Installation" />  
<s:enumeration value="Customer knowledge" />  
</s:restriction>  
</s:simpleType>
```

Company Selection Fields

comp_employees

comp_indcode

comp_mailrestriction

comp_revenue

comp_sector

comp_source

comp_status

comp_territory

comp_type

Person Selection Fields

pers_gender

pers_salutation

pers_source

pers_status

pers_territory

pers_titlecode

Lead Selection Fields

lead_decisiontimeframe

lead_priority

Lead Selection Fields

lead_rating

lead_source

lead_stage

lead_status

Communication Selection Fields

comm_action

comm_hasattachments

comm_notifydelta

comm_outcome

comm_priority

comm_status

comm_type

Opportunity Selection Fields

oppo_priority

oppo_product

oppo_scenario

oppo_source

oppo_stage

oppo_status

oppo_type

Case Selection Fields

case_foundver

case_problemtyp

case_productarea

case_solutiontype

case_source

Now you can...

Case Selection Fields

case_stage

case_status

case_targetver

Address and Product Selection Fields

addr_country

prod_uomcategory

Now you can...

- Explain how Sage CRM Web Services uses Objects and Functions to interact with the client machine and manipulate records.
- Describe the Objects exposed by the Web Service API.
- List the Functions that can be invoked to manipulate data.

Chapter 5

Preparing Web Service Requests: Examples

In this chapter you will learn about:

- How to get a complete Web Services example.
- Sample Web Services requests.

Complete Example

A complete console application code sample is available. Please ask your vendor for a copy of the code files. The new code sample is only compatible with CRM version 5.8.

Sample Soap Requests

The following sections provide a number of sample Soap requests. Some of the request examples are in C# and some are in XML.

Sample Soap Request for Logon

This C# example illustrates how to log onto the server:

```
//An Instance of the web service.
private static WebService binding = null;
//Persistent for the duration of the program, maintain the logon results
private static logonresult SID = null;

private static void LogonToCRMSystem()
{
    try
    {
        SID = binding.logon("admin", "");
        binding.SessionHeaderValue = new SessionHeader();
        binding.SessionHeaderValue.sessionId = SID.sessionid; //Persistent SID
        return true;
    }
    catch (SoapException e)
    {
        Write(e.Message);
    }
    catch (Exception e)
    {
        Write(e.Message + "\n" + e.StackTrace);
    }
}
```

This is the XML request that Web Services processes:

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <logon xmlns="http://tempuri.org/type">
      <username>admin</username>
      <password />
    </logon>
  </soap:Body>
</soap:Envelope>
```

Sample Soap Request for Logoff

This XML example illustrates how to log off:

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <SessionHeader xmlns="http://tempuri.org/type">
      <sessionId>57240080053832</sessionId>
    </SessionHeader>
  </soap:Header>
  <soap:Body>
    <logoff xmlns="http://tempuri.org/type">
      <sessionId>57240080053832</sessionId>
    </logoff>
  </soap:Body>
</soap:Envelope>
```

Sample Soap Request for Delete

This C# example shows how to delete a company whose ID is 66:

```
ewarehouse[] idList = new ewarehouse[1];
companyId aCompanyId = new companyId();
aCompanyId.companyid1 = 66; //66 is id of company to delete
idList[0] = aCompanyId;
deleteresult aResult = binding.delete("company", idList);

if(aResult.deletesuccess == true)
    Console.WriteLine("Number deleted successfully : " + aResult.numberdeleted);
```

This is the XML request that Web Services processes:

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <SessionHeader xmlns="http://tempuri.org/type">
      <sessionId>127169567253830</sessionId>
    </SessionHeader>
  </soap:Header>
  <soap:Body>
    <delete xmlns="http://tempuri.org/type">
      <entityname>company</entityname>
      <records xsi:type="companyId">
        <companyId>66</companyId>
      </records>
    </delete>
  </soap:Body>
</soap:Envelope>
```

```

        </records>
    </delete>
</soap:Body>
</soap:Envelope>

```

Sample Soap Request for Update

This C# example shows how to change the company name for a company whose ID is 66:

```

private static void UpdateACompany()
{
    String idString = "66";
    String newName = "newName";
    ewarehouse[] companyList = new ewarehouse[1]; //can update a number of companies
    company aCompany = new company();
    aCompany.companyid = Convert.ToInt16(idString);
    aCompany.companyidSpecified = true;

    aCompany.name = newName;
    companyList[0] = aCompany;
    updateresult aresult = binding.update("company", companyList);

    if(aresult.updatesuccess == true)
    {}
else
    {}
}

```

This is the XML request that Web Services processes:

```

<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <SessionHeader xmlns="http://tempuri.org/type">
      <sessionId>12663708753831</sessionId>
    </SessionHeader>
  </soap:Header>
  <soap:Body>
    <update xmlns="http://tempuri.org/type">
      <entityname>company</entityname>
      <records xsi:type="company">
        <people xsi:nil="true" />
        <address xsi:nil="true" />
        <email xsi:nil="true" />
        <phone xsi:nil="true" />
        <companyid>933</companyid>
        <name>Design Wrong Inc</name>
      </records>
    </update>
  </soap:Body>
</soap:Envelope>

```

Sample Soap Request for QueryEntity

This example queries a company record whose ID is 66:

```
company aCompany = (company) binding.queryentity( 66, "company").records;
```

Sample Soap XML Representing a Company

The following is the XML representing a company whose ID is 65:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body><queryentityresponse xmlns="http://tempuri.org/type">
<result>
<records xsi:type="typens:company" xmlns:typens="http://tempuri.org/type">
<typens:companyid>65</typens:companyid>
  <typens:primarypersonid>79</typens:primarypersonid>
  <typens:primaryaddressid>77</typens:primaryaddressid>
  <typens:primaryuserid>9</typens:primaryuserid>
  <typens:name>AFN Interactive</typens:name>
  <typens:website>http://www.AFNInteractive.co.uk</typens:website>
  <typens:createdby>1</typens:createdby>
  <typens:createddate>2004-08-30T18:10:00</typens:createddate>
  <typens:updatedby>1</typens:updatedby>
  <typens:updateddate>2004-08-30T18:10:00</typens:updateddate>
  <typens:timestamp>2004-08-30T18:10:00</typens:timestamp>
  <typens:librarydir>A\AFN Interactive (65)</typens:librarydir>
  <typens:secterr>-1845493753</typens:secterr>
  <email>
    <entityname>email</entityname>
    <records xsi:type="typens:email"
  xmlns:typens="http://tempuri.org/type">
      <typens:emailid>120</typens:emailid>
      <typens:companyid>65</typens:companyid>
      <typens:type>Sales</typens:type>
      <typens:emailaddress>sales@AFNInteractive.co.uk</typens:emailaddress>
      <typens:createdby>1</typens:createdby>
      <typens:createddate>2004-08-
30T18:10:00</typens:createddate>
      <typens:updatedby>1</typens:updatedby>
      <typens:updateddate>2004-08-
30T18:10:00</typens:updateddate>
      <typens:timestamp>2004-08-30T18:10:00</typens:timestamp>
    </records>
  </email>
  <phone>
    <entityname>phone</entityname>
    <records xsi:type="typens:phone"
  xmlns:typens="http://tempuri.org/type">
      <typens:phoneid>211</typens:phoneid>
      <typens:companyid>65</typens:companyid>
      <typens:type>Business</typens:type>
      <typens:countrycode>44</typens:countrycode>
      <typens:areacode>208</typens:areacode>
      <typens:number>848 1051</typens:number>
      <typens:createdby>1</typens:createdby>
      <typens:createddate>2004-08-
30T18:10:00</typens:createddate>
      <typens:updatedby>1</typens:updatedby>
      <typens:updateddate>2004-08-
30T18:10:00</typens:updateddate>
```

```

30T18:10:00</typens:timestamp>
                                <typens:timestamp>2004-08-
                                </records>
                                </phone>
                                <address>
                                <entityname>address</entityname>
                                <records xsi:type="typens:address"
xmlns:typens="http://tempuri.org/type">
                                <typens:addressid>77</typens:addressid>
                                <typens:address1>Greenside House</typens:address1>
                                <typens:address2>50 Station Road</typens:address2>
                                <typens:address3>Wood Grn</typens:address3>
                                <typens:city>LONDON</typens:city>
                                <typens:postcode>N22 7TP</typens:postcode>
                                <typens:createdby>1</typens:createdby>
                                <typens:createddate>2004-08-
30T18:10:00</typens:createddate>
                                <typens:updatedby>1</typens:updatedby>
30T18:10:00</typens:updateddate>
                                <typens:updateddate>2004-08-
30T18:10:00</typens:timestamp>
                                <typens:timestamp>2004-08-
                                </records>
                                </address>
                                </records>
                                </result>
                                </queryentityresponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Now you can...

- Get a complete Web Services example.
- Explain sample Web Services requests.

Chapter 6

Creating a CRM Web Services client with Microsoft Visual Studio 2005

In this chapter you will learn about:

- Configuring your installation for a web services client.
- Creating a project in Visual Studio 2005 and adding a Web Reference.
- Using Visual C# to logon to CRM Web Services.
- Using Visual C# to query CRM Web Services.
- Using the information from CRM Web Services in a client application.
- Running the client application.

The purpose of the client

In this example the client application uses the information obtained from CRM Web Services to display the customer's location on a map. The client is a very simple example of a so-called "mashup" web application, which extracts data from the CRM system and then passes it to a function provided by the Google Maps API. As you'll see, by drawing on third-party resources and using Visual Studio 2005's web services tools, the developer can start to create real-world applications with surprisingly few lines of code.

As with the preceding samples, the language used is Visual C#.

As the focus is the functionality associated with CRM Web Services, details of the Google Maps API are treated in passing. However, to use the API in your site you need to provide Google with a URL starting with "http://" in order to obtain a free API key. You can find out more about conditions and usage at <http://www.google.com/apis/maps/>.

Stage 1: Configuring your installation

The first step is to ensure that your Sage CRM installation permits the client to call the web service.

1. With Sage CRM open, select Administration | System | Web Services .
2. Set the Enable Web Services setting to Yes.

3. Set the Make WSDL To All setting to Yes.
4. Set the Force Webservice Log On setting to Yes.

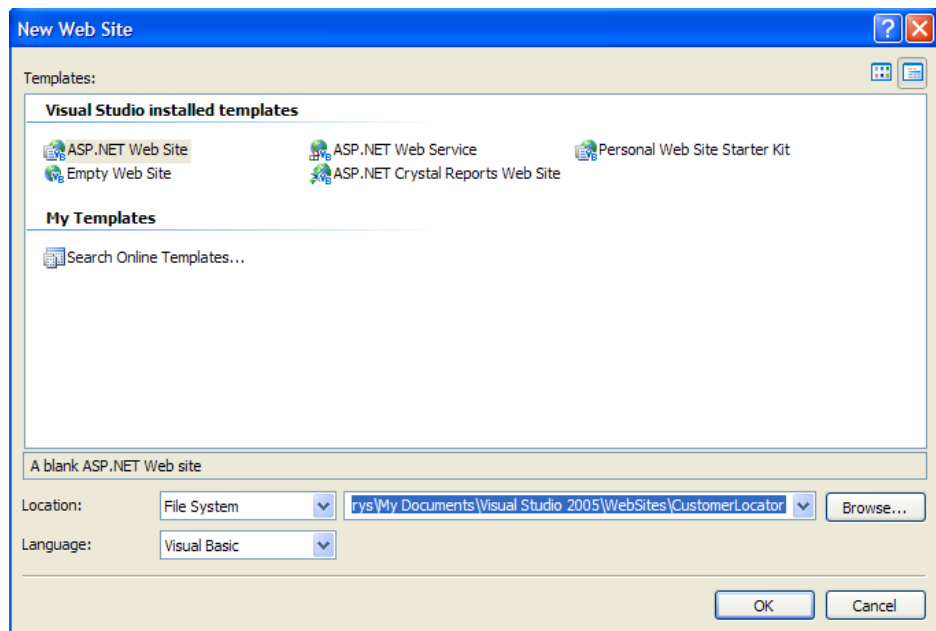
After you have finished testing the web service client, it is recommended that you switch the Make WSDL To All setting back to No to bolster security.

Stage 2: Creating the project and adding a web reference

1. With Visual Studio open, select File | New Web Site. . .

Note that a CRM Web Services client can also be a standard Windows application, a Console Application, or even a Web Service itself. Here we have chosen the ASPX option because of license requirements for the Google Maps API.

2. In the New Web Site dialog box, select the ASP.NET Web Site template.
3. Enter the name of the application at the end of the suggested file path. In this case, it's CustomerLocator.
4. Ensure the option selected in the Language drop-down field is set to Visual C#.



Information about the Web Site entered into the New Web Site dialog box

5. Click on the OK button to confirm your choices.

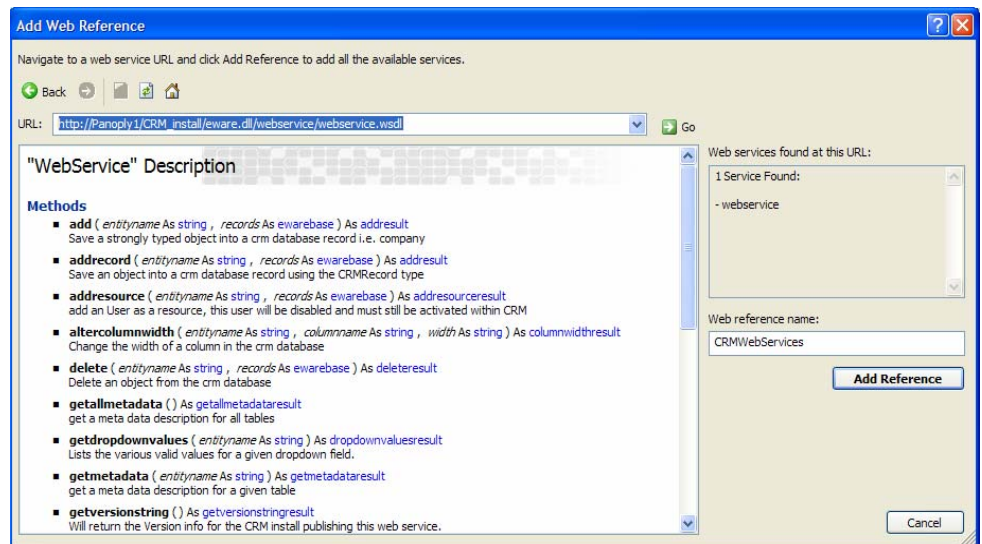
6. The generated files should be visible in the Solution Explorer window. Right-click on the path name at the top of the list and select Add Web Reference... from the drop-down list.
7. In the Add Web Reference dialog box, enter the URL of your CRM installation along with the path extension “/webservice/webservice.wsdl”.

In the screenshot below the URL entered is:

http://Panoply1/CRM_Install/eware.dll/webservice/webservice.wsdl

In this case, the server is called Panoply1 and the installation name is CRM_Install.

8. The main pane lists the methods available from the web service. You can now rename the service “CRMWebServices” and click on the Add Reference button to include the reference in your project.



Adding the Web Reference. Note the URL specifying the path to the WSDL file

A new folder called CRMWebServices, containing the files webservice.discomap and webservice.wsdl, has been added to your project. The “web service proxy” – a C# version of the wsdl file that handles the dispatch of data in SOAP format to the web service – is created automatically.

Stage 3: Using Visual C# to logon to CRM Web Services

1. Select a “code-behind” file for an ASPX file. For example, if the ASPX file is called “Locator.aspx,” the code-behind file is named “Locator.aspx.cs,” with

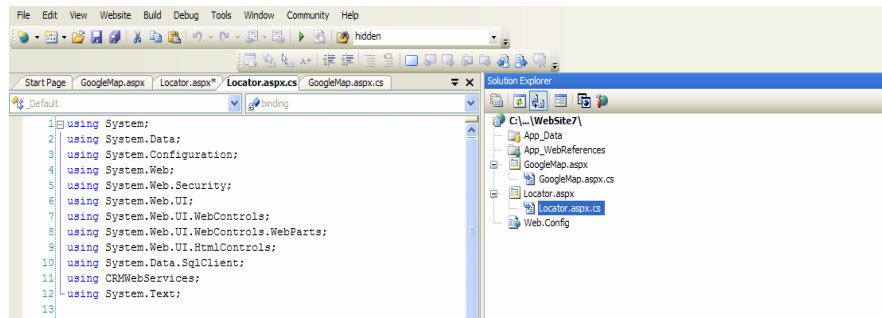
the “cs” suffix indicating “C-Sharp.” Code behind files enable you to separate code from scripting and design.

Note that if the name of your index page is changed to something non-standard such as “Locator.aspx,” you should remember to add this name to the list of documents recognized by Internet Information Services (IIS).

2. Implement the “using” directive so that types in the CRM web service can be used without being fully qualified. This means that you can refer to the “company” class directly rather than having to type “CRMWebServices.company”.

Under the list of other *using* directives add:

```
using CRMWebServices;
```

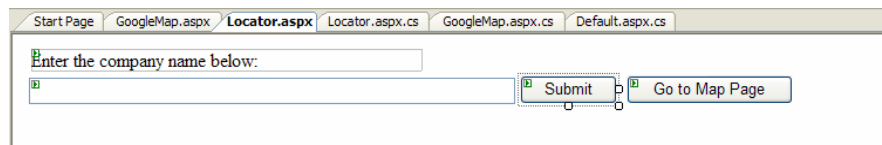


Referencing the CRMWebServices namespace with the “using” directive

3. Next, directly beneath the class declaration (public partial class _Locator : System.Web.UI.Page, for example), declare the types that will serve as the Web Service object and the logonresult object.

```
private static WebService binding = null;  
private static logonresult SID = null;
```

4. Now add a simple interface to allow the user to enter the name of the company to be located. It features a textbox, a Submit button wired to an onclick event handler, and a Go to Map Page button, which is initially invisible.



Interface to allow company names to be entered

5. The next step is to instantiate the binding object of type Webservice and logon with the service. We’ll do this in the event handler for the Submit button.

```
protected void SubmitButton_Click(object sender, EventArgs e)
{
    string compname = this.CompanyName.Text.ToString();

    this.Label1.Text = "The address details for " + compname + " are:";

    binding = new WebService();

    try
    {
        SID = binding.logon("admin", "");
        binding.SessionHeaderValue = new SessionHeader();
        binding.SessionHeaderValue.sessionId = SID.sessionid;
    }

    catch (SoapException ex)
    { Response.Write(ex.Message);}
```

Instantiating the web service object and logging on

You can see that the binding object is created by calling the `WebService` constructor. Next, within a try clause, the `logon` method is called, passing two parameters, the `userid` and the `password`. As you can see, in this example we have used a “magic number” approach by hard coding these `logon` values. In real-world applications, these parameters would probably be supplied by the user and securely stored.

The binding’s `SessionHeader` value collection is set by calling `SessionHeader()`. We then set `binding.SessionHeaderValue.sessionid` to the session id (SID) returned when we called the `logon` method. These lines of code are essential as the SOAP messages sent to the web service need to feature the `sessionid` as a parameter. However, once `sessionid` has been set, all future calls to the web service automatically include this required parameter.

Finally, a catch clause handles any SOAP exceptions triggered by `logon` problems. Note that you should add “`using System.Web.Services.Protocols;`” to the list of directives if you want to directly refer to any object of type `SoapException`.

Stage 4: Using Visual C# to query CRM Web Services

After creating the web service object and logging on, we can start calling web service methods to obtain data.

1. First, declare a variable, `ewbase`, which will contain an array of `ewarebase` objects. `Ewarebase` is the base object for all objects exposed by the CRM Web

Service, which means we can use casting operations to specify specific derived objects such as Company or Address

```

        ebase = qresult.records;
        company compdetails = (company)ebase[0];

        ewarebaselist anAddressList = compdetails.address;
        StringBuilder fulladdress = new StringBuilder();

        if(anAddressList != null)
        {
            for(int x = 0; x < anAddressList.records.Length; x++)
            {
                address anAddress = (address)anAddressList.records[x];

                if(anAddress.addressid == compdetails.primaryaddressid)
                {
                    fulladdress.Append(anAddress.address1.ToString());
                    fulladdress.Append(", ");
                    fulladdress.Append(anAddress.city.ToString());
                    fulladdress.Append(", ");
                    fulladdress.Append(anAddress.state.ToString());
                    fulladdress.Append(", ");
                    fulladdress.Append(anAddress.country.ToString());

                }
            }

            this.CompanyName.Text = fulladdress.ToString();
            this.SubmitButton.Visible = false;
            this.Button1.Visible = true;

        protected void Button1_Click(object sender, EventArgs e)
        {
            Server.Transfer("GoogleMap.aspx", true);
        }
    }

```

Calling the web service

2. We then construct a string, called "where," that will be used in the binding object's query. The compname variable is the value the name of the company the user entered in the textbox.
3. Next the query method is called, with two parameters specified. The first parameter is the WHERE clause we have just constructed. The second is the entity to query. Here, it's the Company entity. The returned result is an object of type queryresult.
4. We assign the queryresult.records value to the ebase object. We can "cast" the ebase object at index[0] to the derived object company using the statement :

```
company compdetails = (company)ebase[0];
```

5. We now assign a property of the compdetails object, address – which should contain location data relevant to the returned company – to the variable, anAddressList, of type ewarebaselist. Further down the code, we can then cast this abstract type’s record property to an address object named “anAddress.” Properties of this anAddress object (address1, city, state, and country) can then be assigned to a StringBuilder object to build up the company’s complete address.

Note that all these calls to web service methods should be wrapped in try-catch clauses. They are presented here without such exception handling code for the sake of brevity and legibility.

Stage 5: Using the information

Once we have the address details for a specified company, we can start preparing the string that will be fed to the Google Maps API Geocoder method (for more information on this feature, see here:

http://www.google.com/apis/maps/documentation/#Geocoding_Examples)

1. Google’s showAddress method expects an address in the form of a string, with the component elements separated by a comma and a space. One approach is the use .NET’s StringBuilder class to build up a string created by returning various properties of the address object.

In the example, we retrieve the address, city, state, and country values and append them to the StringBuilder object called “fulladdress”. (Note that strings are immutable in C#, hence the use of StringBuilder.)

```

        ebase = qresult.records;
        company compdetails = (company)ebase[0];

        awarebaselist anAddressList = compdetails.address;
        StringBuilder fulladdress = new StringBuilder();

        if(anAddressList != null)
        {
            for(int x = 0; x < anAddressList.records.Length; x++)
            {
                address anAddress = (address)anAddressList.records[x];

                if(anAddress.addressid == compdetails.primaryaddressid)
                {
                    fulladdress.Append(anAddress.address1.ToString());
                    fulladdress.Append(", ");
                    fulladdress.Append(anAddress.city.ToString());
                    fulladdress.Append(", ");
                    fulladdress.Append(anAddress.state.ToString());
                    fulladdress.Append(", ");
                    fulladdress.Append(anAddress.country.ToString());

                }
            }

            this.CompanyName.Text = fulladdress.ToString();
            this.SubmitButton.Visible = false;
            this.Button1.Visible = true;
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            Server.Transfer("GoogleMap.aspx", true);
        }
    }

```

Building up a StringBuilder object by returning address object properties

2. A string representation of the StringBuilder object is then assigned to the text box into which the user originally entered the company name. The SubmitButton now becomes visible allowing the user to find the displayed address on a Google map.
3. The submit button's event handler calls the Server.Transfer method to display the page that will use the Google API. Note that the Transfer method's second, optional parameter is set to true. This allows the new page to retrieve any form variables from this page. This is convenient as it provides an easy way of obtaining the full address from the textbox.
4. On the GoogleMap.aspx page, the fulladdress string can be retrieved from the submitting form using the Request.Form method and passed as an argument to the showAddress method. The code for showAddress is cut and pasted from the Google Maps API. Remember, however, that it is necessary

to include the required Maps API key, which is specified using the `src` attribute of the script element. See the relevant Google documentation for the code required.

```
function showAddress(address) {
    if (geocoder) {
        geocoder.getLatLng(
            address,
            function(point) {
                if (!point) {
                    alert(address + " not found");
                } else {
                    map.setCenter(point, 13);
                    var marker = new GMarker(point);
                    map.addOverlay(marker);
                    marker.openInfoWindowHtml(address);
                }
            }
        );
    }
}
```

Google-supplied `showAddress` code. The argument passed into the method is the text representation of the `fulladdress StringBuilder` object.

Stage 6: Running the application

Once the application has been debugged and hosted with IIS, it can be tested.

1. The basic interface allows the user to enter the company name. When the submit button is clicked, the web service object is created, logged on to, and the required query methods called.



Entering the Company name.

2. Assuming no exceptions are triggered (and that the company exists in the CRM database), the company's address should be displayed in the text box.

The address details for Verity Signs are:

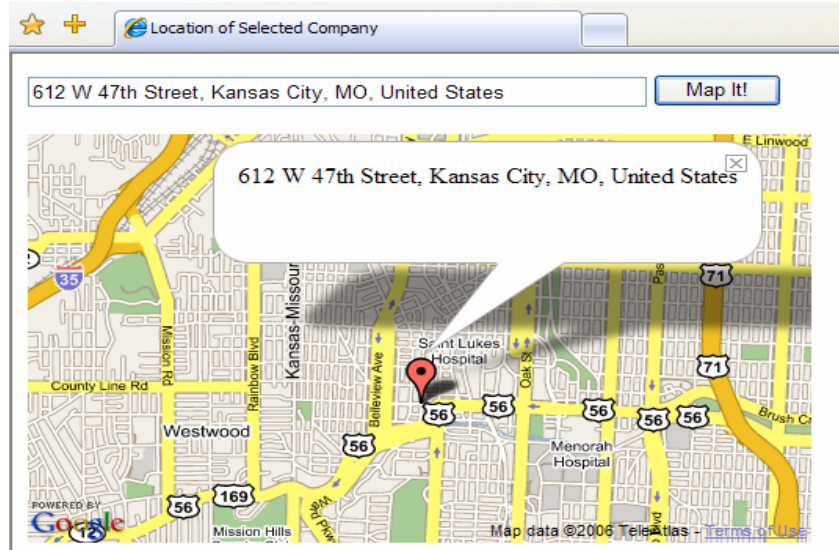
612 W 47th Street, Kansas City, MO, United States

Go to Map Page

Returned address for Verity Signs.

Now you can...

3. Now click on the Go to Map Page button to open the GoogleMap.aspx page. On this new page, click on the Map It! button to see the company's location on a Google map.



Location of the selected company mapped using the Google API

Now you can...

- Create a CRM Web Services client in Visual Studio 2005.
- Call CRM Web Service methods using Visual C#
- Use the information obtained from CRM Web Services in a sample application.

