Sage CRM

v7.1 Developer Guide

Contents

Chapter 1: Introduction 1-1

Chapter 2: Overview 2-1

Web Architecture 2-1

Extensibility Architecture 2-2

.NET Architecture 2-3

.NET vs Classic ASP 2-4

Security 2-5

Application Level Security 2-5

Server Level Security 2-5

Database Level Security 2-6

Customization Overview 2-6

Extending CRM with Custom ASP Pages 2-6

Overview of Customizable Areas 2-7

Database Overview 2-7

CRM Entities 2-7

Metadata 2-8

Using SQL and Triggers 2-8

Chapter 3: Getting Started 3-1

Building an ASP Page 3-1

Integrating the ASP Page Into CRM 3-4

Creating Custom Queries 3-5

Understanding "Context" and the GetContextInfo Method 3-7

Client-Side Scripting with JavaScript and DOM 3-7

Accessing User Information 3-9

Scripting in the CRM Interface 3-10

Field Level Scripting 3-10

Table Level Scripting 3-12

Chapter 4: Customization Basics 4-1

CRM Blocks Overview 4-1

Block Naming and Custom Blocks 4-1

Adding Help to Custom Pages 4-3

Step 1: Create a Custom Help Page 4-3
Step 2: Add a Help Button to your Custom Page 4-3
Limitations 4-3
Lists 4-3
How to Create a List 4-3
How to Display a List 4-4
Display a List Using an ASP Page 4-4
Display a List using Runblock and the List Name 4-5
Screens 4-5
How to Create a Screen 4-5
How to Display a Screen 4-6
Display a Screen Using Runblock and Screen Name 4-7
Display a screen using Runblock with a Custom Block 4-7
Display a screen with an ASP page 4-7
Buttons 4-8
Creating Button Groups 4-8
Adding Buttons to Button Groups 4-8
Viewing Button Groups 4-9
Restricting Access to Button Groups 4-9
CRM Classic Dashboard 4-9
Customizing The Classic Dashboard 4-9
Adding a List Block To The Classic Dashboard 4-11
Adding a Content Block To The Classic Dashboard 4-11
Adding a Chart To The Classic Dashboard 4-12
CRM Interactive Dashboard 4-13
Customizing the Interactive Dashboard 4-13
Example: Adding a Content Block to the Interactive Dashboard using the Contents field 4-13
Example: Adding a Content Block To The Interactive Dashboard based on an ASP page 4-15
Adding a Third-party gadget to the Interactive Dashboard 4-16
scrmPublishEvent 4-17
scrmRegisterEvent 4-17

scrmGetGadgetProperty 4-17
scrmSetGadgetProperty 4-18
scrmSaveGadgetProperties 4-18
scrmGetSageCRMOwner 4-18
Blocks 4-18
Creating a New Block 4-18
Customizing a Block 4-19
Displaying a Block 4-19
System Menus 4-20
Modifying System Menus 4-20
Creating a Main Menu Button 4-20
Creating an Admin Menu Button 4-21
Creating an External Link on the Main Menu 4-21
Tabs 4-24
Creating a New Tab Group 4-24
Editing the Main Menu Tab Group 4-25
Adding a Tab that Links to an ASP Page 4-25
Restricting Access to the Tab 4-26
Tab Actions 4-27
Chapter 5: Database Customization 5-1
Introduction to Database Customization 5-1
Creating a New Table 5-1
Creating a New Database Connection 5-3
Creating a New Table Connection 5-4
Example: Creating a Tab to Display a List of Invoices 5-4
Example: Displaying an Individual Invoice from a List 5-6
Example: Adding New Data Entry and Maintenance Screens 5-7
Table and Entity Scripts and Functions 5-12
Creating a Table Level Script 5-14
Detached Table Level Scripts 5-14
Creating an Entity Level Script 5-14
Example: UpdateRecord in an Entity Level Script 5-15
Example: InsertRecord 5-16

Example: PostInsertRecord 5-16
Example: UpdateRecord 5-16
Example: DeleteRecord 5-17
Advanced Customization Wizard 5-17
Creating a New Main Entity 5-17
Advanced Customization Wizard Parameters 5-18
Enabling Company and Person Deduplication 5-21
Custom Files and Metadata 5-22
Making Custom Entities Available for Reassignment 5-24
Customizing a New Main Entity 5-25
Advanced Customization Wizard Example 5-26
Chapter 6: Component Manager 6-1
Introduction to Component Manager 6-1
Recording 6-1
Scripting 6-1
Installing 6-1
What Types of Customizations can be Recorded? 6-1
Recording Customizations 6-2
The Component Details Screen 6-2
Starting Component Manager 6-3
Stopping Component Manager 6-3
Adding Customizations to an Existing Component 6-3
Changing the Current Component 6-4
Scripting Customizations 6-4
Previewing Changes 6-4
Scripting Changes 6-4
Scripting Multi-Stage Customizations 6-5
Scripting Workflows 6-6
Saving a Component 6-7
Installing a Component 6-7
Component Manager Log File 6-8
Advanced Component Options 6-9
How to Generate a Component Script using Advanced Component Options 6-9

How to Create a New Component Using Advanced Component Options 6-10
Modifying Component Manager Scripts 6-11
Error Handling 6-12
Referential Integrity 6-12
Script Parameters 6-13
Phone and E-mail Changes 6-14
Component Manager Methods 6-14
Component Manager Scripting Examples 6-36
Chapter 7: Graphics and Charts 7-1
Introduction 7-1
Charts Overview 7-1
Fusions Charts System Parameters 7-2
Pie Chart Only Effects 7-2
Special Effects For Charts 7-3
Using External Data for Charts 7-3
Chart Examples 7-4
Example: Adding a New Chart 7-4
Example: Organization Chart 7-6
Graphics Overview 7-8
Graphics Formats 7-8
Graphics Performance Tips 7-9
External Images 7-10
Graphics Examples 7-10
Example: Adding a New Graphic 7-10
Example: Pipeline 7-11
Graphic Effects Basics 7-14
Change Image Color 7-14
Clear An Image 7-14
Display Errors 7-14
Drawing Functions 7-14
Merging 7-16
Special Effects 7-17
Animation 7-17

Adding Frames 7-17
Delay 7-17
Loops 7-17
ASP Example 7-18
Chapter 8: ASP Object Reference 8-1
Introduction to the ASP Object Reference 8-2
Quick Reference Table 8-3
Examples 8-7
Sample CRMGridColBlock ASP page 8-7
Sample CRMListBlock ASP page 8-7
Using the New Workflow Properties in an ASP page 8-8
AddressList Object 8-9
Methods 8-9
Properties 8-10
Attachment Object 8-10
Methods 8-10
Properties 8-11
AttachmentList Object 8-11
Properties 8-11
Email Object 8-12
Methods 8-12
Properties 8-13
MailAddress Object 8-15
Properties 8-15
MsgHandler Object 8-15
Methods 8-16
Properties 8-16
CRM Object 8-17
Methods 8-17
Properties 8-22
CRMBase Object 8-22
Methods 8-23
Properties 8-27

CRMBlock Object 8-27
  Methods 8-28
  Properties 8-29
CRMChartGraphicBlock Object 8-33
  Methods 8-33
  Properties 8-35
CRMContainerBlock Object 8-36
  Methods 8-37
  Properties 8-39
CRMContentBlock Object 8-43
  Properties 8-43
CRMEntryBlock Object 8-43
  Methods 8-44
  Properties 8-44
CRMEntryGroupBlock Object 8-55
  Methods 8-56
  Properties 8-57
CRMFileBlock Object 8-57
  Properties 8-58
CRMGraphicBlock Object 8-59
  Methods 8-59
  Properties 8-68
CRMGridColBlock Object 8-69
  Properties 8-70
CRMListBlock Object 8-72
  Methods 8-73
  Properties 8-75
CRMMarqueeBlock Object 8-77
  Properties 8-77
CRMMessageBlock Object 8-79
  Properties 8-80
CRMOrgGraphicBlock Object 8-82
  Methods 8-82

CRMPipelineGraphicBlock Object 8-82
Methods 8-83
Properties 8-85
CRMQuery Object 8-85
Methods 8-86
Properties 8-87
CRMRecord Object 8-89
Methods 8-89
Properties 8-92
CRMSelfService Object 8-95
Self Service Method and Property Differences 8-95
Note on Instantiating the CRMSelfService Object 8-96
Methods 8-97
Properties 8-98
CRMTargetListField Object 8-99
Properties 8-99
CRMTargetLists Object 8-99
Methods 8-100
Properties 8-100
Example: Creating and Saving a Target List 8-102
Example: Retrieving a Target List 8-103
CRMTargetListFields Object 8-103
Methods 8-104
Properties 8-104
Chapter 9: Web Services 9-1
Introduction to Web Services 9-1
Setting Up CRM Web Services 9-2
Objects and Functions Overview 9-5
List of Web Services Functions 9-5
List of Web Services Objects 9-7
The CRM RecordType Object 9-10
Selection Fields in Web Services 9-11
Introduction to Web Services Examples 9-13

Sample SOAP Requests 9-15
Chapter 10: SData Read-only 10-1
Introduction to SData 10-1
Overview of SData within Sage CRM 10-1
SData Prerequisites 10-2
Switching on SData 10-2
Constructing SData URLs 10-3
SData URL Examples 10-3
SData Authentication 10-3
Chapter 11: .NET 11-1
Extending Sage CRM With .NET 11-1
Sage CRM .NET API and ASP.NET 11-2
Programming Languages 11-2
Component Manager 11-2
Connection Pool 11-2
Using .NET Application Extensions 11-2
The CustomDotNet Folder 11-2
Changing the Build Location of the DLL 11-2
Copying the DLL file to the CustomDotNet Folder 11-3
Calling the .NET Application Extension 11-3
Calling the .NET Application Extension From Tabs/Menus 11-3
Calling the .NET Application Extension From List Block Hyperlinks 11-4
Calling the .NET Application Extension From Within Another Custom .NET Application 11-4
Calling the .NET Application Extension From ASP Pages 11-4
.NET API Debugging 11-4
Method 1: Change IIS Security 11-5
Method 2: Use COM+ 11-5
Troubleshooting 11-6
.NET API Class Library Documentation 11-6
Sage Namespace 11-6
Sage.CRM.WebObject Namespace 11-7
Sage.CRM.Controls Namespace 11-7

Sage.CRM.Data Namespace 11-7
Sage.CRM.Utils Namespace 11-7
Sage.CRM.Blocks Namespace 11-8
Sage.CRM.HTML Namespace 11-8
Sage.CRM.UI Namespace 11-8
Installing the .NET SDK 11-8
Pre-Installation Checklist 11-8
Installing the SDK and .NET Templates 11-9
Installation Troubleshooting 11-9
Manually registering the DLL 11-9
Manual Installation of the CRM Visual Studio .NET Templates 11-10
Uninstalling the .NET SDK 11-10
.NET Examples 11-10
Creating a Simple Sage CRM Interface 11-11
Creating a More Complex CRM Interface 11-12
Creating a Project Based on a Sage CRM Template 11-16
CRM Basic Template 11-16
CRM Entity Template 11-17
Index **i**

Chapter 1: Introduction

This Developer Guide is for CRM Implementers and Programmers. We assume that you are a confident CRM user and are fully conversant with the topics covered in the System Administrator Guide.

We also assume that you are fully conversant in the areas of:

- SQL views, tables, databases, data relationships, and normalization.
- Programming concepts.
- Internet technologies, specifically Active Server Pages (ASPs).

Sage CRM is extensible using the Extensibility Module (EM). EM technology is also known as CRM Blocks. As well as the ability to extend your own system, EM enables you to create custom pages which you can add to other CRM systems using the Component Manager feature—enabling you to extend your customers' CRM systems.

The focus of this guide is on extending your Sage CRM system using CRM Blocks, ASP pages and the .NET SDK. The Sage CRM system can also be extended using Web Services (page 9-1).

Please note that while this guide refers to Sage CRM, CRM, or the CRM system throughout, regional products may use different brand names.

Also note that CRM Blocks used to be known as eWare Blocks. Existing code that refers to eWare Blocks or the eWare object will still work, and CRM and eWare are interchangeable (except in the context of CRMSelfService). Please see the ASP Object Reference for more information.

Chapter 2: Overview

In this chapter you will learn how to:

- Get an overview of the Web Architecture.
- Get an overview of the Extensibility Architecture.
- Get an overview of the .NET Architecture.
- Compare .NET and ASP as customization methods.
- Discuss Security in Sage CRM.
- Get an overview of customization options.
- Discuss database properties.

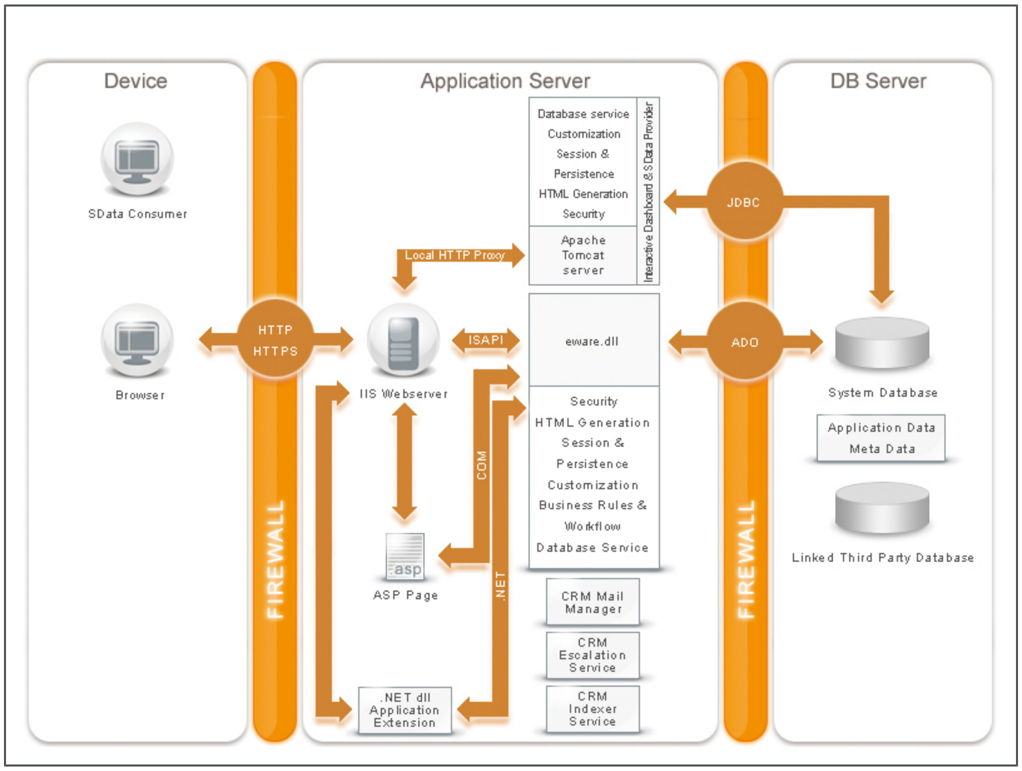
Web Architecture

CRM is based on a standard Web applications structure. The application server sits into an existing intranet environment.

Web Browser. A standard browser such as Internet Explorer. As CRM is a thin-client configuration, a browser is all that is required on each user's computer.

Firewall (optional). You may have a firewall installed between your network and the Internet for network security. This is not a required component for CRM.

Web Server. The CRM Application Server works with Microsoft Internet Information Server. Please refer to the latest Product Support Matrix for supported versions.



Web Architecture

CRM Application Server. The CRM Application Server comprises a number of subsystems, which work together to coordinate the delivery of information and functionality to clients. These include objects for checking user security, maintaining user persistence, getting information from the database, generating Web pages from data, and processing business rules and logic. This guide describes each of these objects and outlines their properties and methods.

Database. CRM has native support for Microsoft SQL Server and Oracle. The database is used to store corporate data and metadata, which define the system customization, security, and business rules.

E-mail Server. The E-mail Management functionality enables the Application Server to be integrated with the E-mail server to automate the sending of e-mails and SMS messages as part of the CRM implementation. The CRMEmail object and the CRMMsgHandler object and it's child objects are used to customize the E-mail Management functionality. Note that the CRMMessageBlock object can also be used to send messages in SMS and e-mail format.

As CRM is designed for a multi-tier environment, you can have all of the above components on the same or separate machines.

Note : IIS and the CRM DLL have to be on the same server, so as a general rule IIS and CRM are installed on the same server. We would recommend that the database is installed be on a dedicated database server.

DLL. The CRM DLL (Dynamic Link Library) runs on a Windows Server with Microsoft Internet Information Server (IIS) Web Server software. The IIS Web Server communicates with CRM through the CRM DLL and Internet Server Application Programming Interface (ISAPI) technologies.

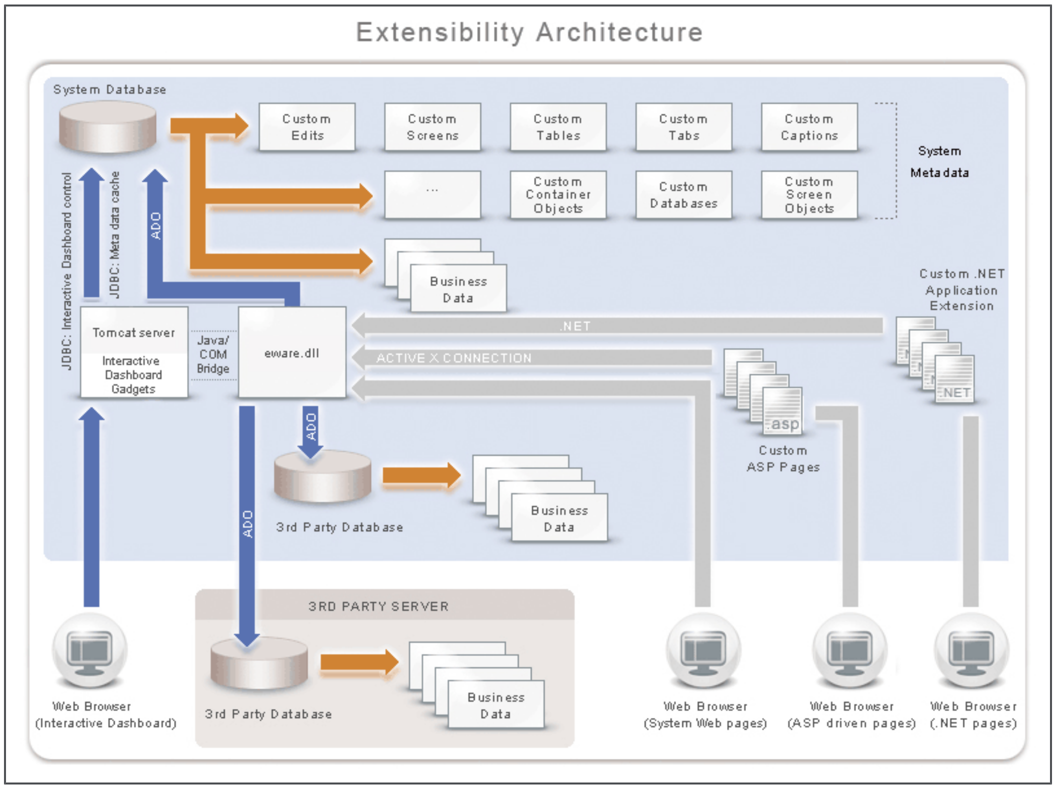
Apache Tomcat Redirector. The redirector in v7.1 is an ASP.NET Reverse Proxy, which is a 32-bit/64-bit ASP.NET web application configured in standard Apache format. This ASP.NET rewriter uses the HTTP protocol instead of binary socket and can be accessed directly in a browser, so you can easily check whether Tomcat is working. Please refer to the *Installation and Upgrade Guide* and the latest articles on the *Sage CRM Ecosystem* for more information.

Extensibility Architecture

The Extensibility Module ('EM') provides users with a range of powerful functions that allow them to customize and extend the existing CRM product.

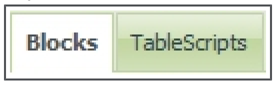
These functions are made available through the CRM ActiveX object, which is made up of a number of CRM methods and properties. The CRM object components have a variety of functions, which render HTML and display Screen and List objects-previously defined in the CRM system itself.

There are also a number of database connectivity options available. These include searching, inserting, updating, and deleting data to and from CRM data, as well as data from external tables and databases.



Extensibility Architecture

You can establish if the Extensibility Module is present in your CRM installation as follows: Select **Administration | Customization | Company**.
If you have the Extensibility Module, you will see Blocks and TableScripts tabs:



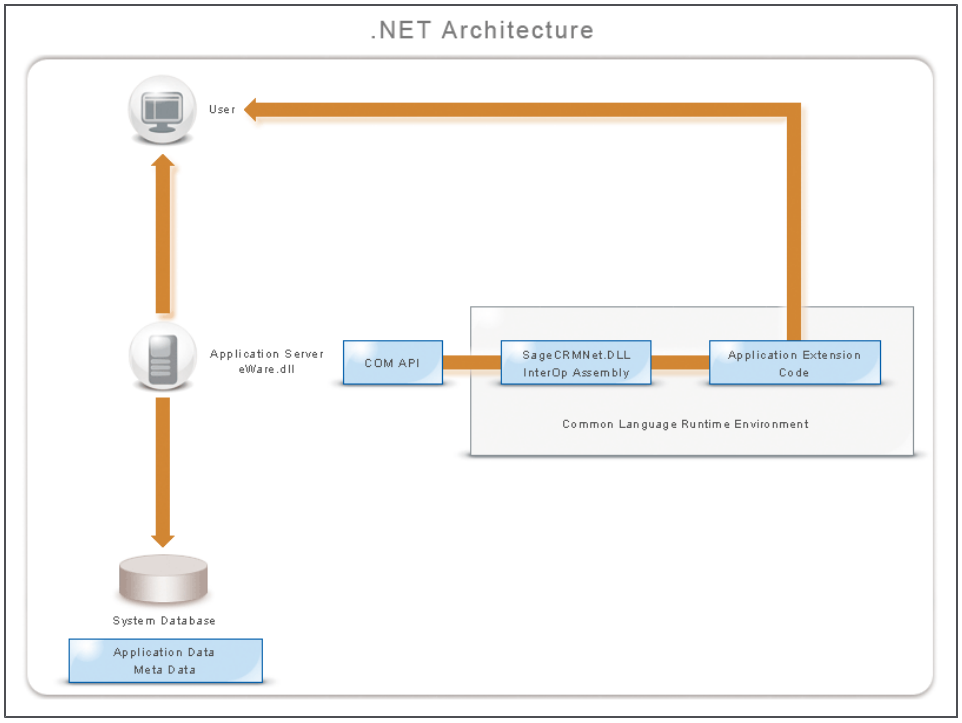
.NET Architecture

As the core of the Sage CRM solution is represented by an ActiveX component, Sage has leveraged Microsoft’s Interop technology to expose this existing COM component to managed code (that is code executed by Microsoft's .NET Framework Common Language Runtime).

The process of exposing COM components to the .NET Framework can be challenging, involving steps such as converting the coclasses and interfaces contained in a COM type library to metadata and deploying Interop applications as strong-named, signed assemblies in the global assembly cache.

The Sage CRM SDK for .NET handles these low-level implementation details by providing a redistributable package that installs the .NET component onto your system and makes it readily available within the Visual Studio environment.

The diagram below shows this architecture.



.NET Architecture

1. CustomDotNetDll action calls Application extension.
2. CustomDotNetDll action uses COM interop to trigger behaviour in CRM .NET Component.
    - Passes CRM Application Extension DLL name and session information.
    - Calls CRM Application Extension.
3. CRM Application Extension processes data and generates and returns HTML.

The Sage CRM .NET API conforms to the .NET 2.0 Framework. It provides a type library that exposes the Sage CRM objects, properties, and methods. Through its core libraries the Sage CRM .NET Component manages both data access and web interface generation. Projects developed using the Sage CRM .NET Component will be compiled into a dll and called directly from within Sage CRM. By using Sage CRM meta data Application Extensions constructed using the Sage CRM .NET API will look, feel and perform exactly like core system pages.

Reference to the Sage CRM .NET component from within ASP.NET projects is not supported.

Any programming language that conforms with the .NET 2.0 Framework can be used for the development of Sage CRM Application Extensions (e.g. J#, C# VB.NET etc).

.NET vs Classic ASP

Before the release of Sage .NET SDK, the main method for developers who wished to add custom pages to the CRM environment was through the creation of ASP pages. As these pages were coded in simple text editors—typically Notepad or WordPad—developers could not use features offered by IDEs, such as IntelliSense (providing drop-down lists of available objects, properties, and methods), syntax checking, and debugging tools. The absence of IntelliSense, in particular, usually required developers accessing functionality provided by the Sage CRM API to refer repeatedly to the relevant documentation.

In addition, the absence of meaningful syntax checking and debugging tools in text editors meant that developers could not gauge the correctness of their ASP coding until the page was upload to the Sage

CRM environment and running in the context of new tab. Developers reload pages in the Sage CRM's Custom Pages folder to check whether their ASP code is working as intended.

The simplicity of the ASP/Sage CRM API/Text Editor combination makes it appropriate to solutions that require simple coding implementations and rapid deployment. However, given that coding separate ASP pages is not conducive to best programming practice (specifically, OOP principles and associated practices, such as refactoring) and that creating multi-file projects can demand more advanced project tools, developers should appreciate a more sophisticated solution for certain customization tasks.

By allowing developers to access the Sage CRM through the .NET framework, several of the issues associated with earlier customization solutions are comprehensively addressed.

Instead of coding in a basic text editor, developers can now use the suite of features provided by Microsoft's Visual Studio. With the Sage CRM .NET component properly referenced by the application, Visual Studio treats objects from the Sage CRM API in the same way as any other initialized C# object. This means when, for example, you are using a CRM object and are calling one of its methods, the IntelliSense prompts you with a list of the object's properties and methods when you type the "." operator after the object name.

Also, rather than coding in a scripting language that initializes and uses both ASP and Sage CRM API objects, the .NET framework enables developers to write in C# or Visual Basic.NET, highly object-oriented languages that can draw on the resources of .NET's class library.

In addition, whereas the source code in ASP pages can be accessed by users selecting the View–Source option from the browser menu, .NET applications are compiled into binary DLLs before deployment. Preventing access to source code means improved security and better protection of intellectual property.

Security

CRM is modelled on an n-tier architecture. Each tier includes a number of security mechanisms.

Application Level Security

Every user is assigned a valid username and password. The only person allowed to add or remove users is the System Administrator. Within the system, each user can be assigned different levels of access security depending on their job role. When IIS uses SSL encryption, the system is aware of this, and when the client attaches any documents to a form in the system, it sends it through the encrypted sessions.

- **User Authentication/Password Setup**. A user requires a Logon ID and password to access the system. The user's password is encrypted both within the system and in the database for maximum security.
- **Tab and Team Restrictions**. Access to individual tabs within CRM can be restricted by the System Administrator, as can the level of access that each user is allocated. The System Administrator can also assign users to teams, which further categorizes and restricts levels of access to individual tabs.
- **Security Profiles and Territories**. The System Administrator can manage security access rights across the organization by setting up Territory Profiles and, if required, security Territories. A Profile is a way of grouping users together when defining access rights (View, Update, Insert, Delete). A Territory is a way of further dividing user rights by location or other criteria. For example, you may want users in the Europe territory to view all Opportunities within the USA territory but not to be able to update them. Complex inter-territory security rights and exception handling are also catered for using Territory Policies. Profiles and Territories are set up from Administration | Users | Security, please refer to the System Administrator Guide for more information.

Server Level Security

You can use all three or a combination of the following methods to secure the CRM server:

- **NT Challenge/Response**. This allows access to clients with a valid domain login.
- **SSL Encryption**. This secures your data sessions with client users.
- **A Firewall**. This restricts unauthorized access from outside the network and allow only authorized users through.

Database Level Security

CRM users do not have direct access to the database. The CRM DLL accesses the database by using a predefined login. When a user requests data, the CRM DLL connects to the database using Microsoft Data Access Components (MDAC) and retrieves the required data. For extra security, the CRM DLL can be configured to access the database using a login with limited access.

Customization Overview

Sage CRM can be customized in a number of different ways:

- By development of custom ASP pages which can perform a wide variety of functions including for example displaying data entry screens or search and list pages.
- By using the system interface to add fields, tables, tabs etc (see table below).
- Through use of the .NET SDK.

Extending CRM with Custom ASP Pages

The ability to add a custom ASP page to CRM and display that page on a user-defined tab allows developers to extend and modify system functionality. As well as allowing developers to leverage their expertise in technologies such as Active Server Pages (ASP), JavaScript, and W3C's Document Object Model (DOM), the Extensibility Module (EM) provides an extensive library of existing functionality represented by the classes, methods, and properties provided by the CRM Block architecture.

The Block architecture serves as a highly accessible and flexible SDK (Software Development Kit) by enabling the developer to use the CRM Object, which is initialized by standard CRM include files referenced at the top of a custom file. As well as providing a range of useful functions and methods itself, the CRM Object allows you in turn to initialize further blocks that build up the screen interface, generate lists, manipulate database records, and modify scripts.

As CRM blocks reside on the install server, code using the CRM SDK typically also has to run on the server before the compiled page is dispatched to the client. ASP is the technology of choice for server-side coding, as it is supported by Internet Information Services (IIS) and provides six built-in objects that ease the creation of dynamic web pages.

However, after the page has been downloaded from the server and displayed within the CRM system, it is possible to handle data and respond to user-generated events using client-side JavaScript and the DOM.

Server-side and client-side scripts can coexist in the same ASP file, with each approach playing to the strengths of that particular technology. Server-side ASP code can be used to draw on the prebuilt functionality of the CRM Block architecture. In contrast, client-side JavaScript offers immediate responses to user actions and enables you to access the DOM to navigate the screen interface.

Overview of Customizable Areas

Aside from integrating ASP pages into CRM, you can also customize the following areas by using the system interface to enter scripts and change settings:

Customizable Area	**Without EM**	**Additional Customization with EM**
Fields	Create new fields and customize existing ones.	
Screens	Customize existing screens that are in the CRM database. Add and remove fields that were created in the system.	Add new screens.
Lists	Customize existing lists that are in the CRM database.	Add new lists.
Tabs	Add new tabs to tab groups that are in the CRM database. Customize main menu buttons.	Add new tab groups. Link tabs to custom files or URLs. Link tabs to runblock and runtabgroup functionality.
Views	Add new views. Change some existing views.	
Blocks		Add Blocks.
Table Scripts		Add Table and Entity Scripts.
Tables and Databases*		Connect to a new table. Connect to a new database. Create a new table.
Button Groups*		Add new button groups.
Component Manager**	Upload a component.	Record and script components.

* Tables And Databases and Button Groups are customized from Administration | Advanced Customization.

** Component Manager is accessed from Administration | Customization.

Database Overview

CRM Entities

In Sage CRM, an "Entity" is a representation of a real world entity such as a Company, a Person, or a Sales Opportunity. The entity may have relationships with other Entities. For example, a Company can have many People working for it. Each of these Entities may then have other entities in relationships with them, for example People can have Communications and Sales Opportunities associated with them.

CRM entities are made up of information from several different tables linked in a logical way. For example, the Company entity is made up of information from the Company, Address, Phone, and Person tables.

CRM includes the following primary Entities:

- Company
- Case
- Opportunity
- Person
- Communication
- Leads
- Quotes
- Orders

Unlike database entities, CRM entities are several tables linked together in a logical business grouping.

Metadata

In CRM, Metadata encompasses all of the information that is required to make sense of the stored business data. For example, there are metadata tables that contain field captions, or convert code values from drop-down fields into meaningful information.

The system database contains CRM metadata tables. These are prefixed with the word 'custom'.

Using SQL and Triggers

You can use the CRM Query object's SQL method to include SQL statements in an ASP page.

You can also use SQL conditional clauses within the system in **Administration | Customization | <Entity> | Tabs**. For more information please refer to the System Administrator Guide.

CRM also includes Table and Entity level scripting functionality that you can include as an alternative to SQL triggers.

Chapter 3: Getting Started

In this chapter you will learn how to:

- Build an ASP page.
- Integrate an ASP page into CRM.
- Create custom queries.
- Understand "Context" and the GetContextInfo method.
- Get an overview of client-side scripting with JavaScript and DOM.
- Access user information.
- Enter script in the CRM interface.
- Get an overview of field-level scripting.
- Get an overview of table-level scripting.

Building an ASP Page

Sage CRM ASP pages leverage the properties and methods of the CRM object to connect to the system database and produce formatted output to the web browser. Standard ASP scripting conventions are observed.

To illustrate the creation of a simple ASP page, we will create a custom search page which will allow the user to search contacts in the CRM database (the Person Entity) and display the results in a list.

The first step in creating a page is to use an include statement to ensure that the CRM object is instantiated:

```
<!-- #include file ="sagecrm.js"-->
```

Note that this Developer Help assumes that the CRM object will be instantiated as "CRM". In some older versions of the include file, the object is instantiated as "CRM"' or "eWare". If you are using an older version, simply use "eWare" when referring to the object, or modify the include file to set: CRM = eWare;

Aside from instantiating and initializing the CRM Object, the include file also:

- References the Sage CRM CSS Stylesheet
- Defines constants
- Checks for errors

The include file referenced in your file depends on the language you're scripting in:
SAGECRM.JS - referenced in JavaScript-based ASP pages. This file sets the default language to JavaScript.
SAGECRMNOLANG.JS - this file does not set the default language.
SAGECRM.VBS - referenced in Visual Basic-based ASP pages. This sets the default language to VB Script.
eWare.JS - for backward compatibility with pre 5.6 versions of CRM.
ACCPACCRM.JS - for backward compatibility with pre 7.0 versions of CRM.
ACCPACCRMNOLANG.JS - for backward compatibility with pre 7.0 versions of CRM.
ACCPACCRM.VBS - for backward compatibility with pre 7.0 versions of CRM.

After the include file, you must open the ASP delimiters <% %> to tell the ISAPI.DLL that the contained ASP code will execute on the server, and then call the CRM object methods required for your page.

In the example below, we call the CRM object's GetBlock method to initialize one of the child blocks that implement core CRM functionality. This method is likely to feature at the start of most of your custom ASP pages, and often several more times throughout, as you build up a CRM screen using several components.

```
<%
// get an empty container block
var SearchContainer = CRM.GetBlock('Container');
```

In this example, the GetBlock's parameter is "Container" which indicates the CRMContainerBlock object. This object is used to group and ensure proper display of the output from the other objects on the page. The container block also provides default onscreen elements, such as buttons, that make it easier to format and support custom layouts.

The returned CRMContainerBlock object has been assigned the variable name SearchContainer. Aside from default buttons, this container screen is empty. To make it useful, we now need to add some blocks.

```
var SearchBlock = CRM.GetBlock('PersonSearchBox');
```

We again use the CRM objects GetBlock method to retrieve a block and its associated functionality. In this case, we have specified *PersonSearchBox*. PersonSearchBox is an instance of the CRMEntryGroupBlock, and you can see (and modify) the contents of PersonSearchBox by going to:

Administration | Customization | Person | Screens | Person Search Screen

Other standard screens based on CRMEntryGroupBlock include CompanySearchBox, PersonEntryBox, CaseDetailBox and other entity search and entry screens.

We now add SearchBlock (our instance of the PersonSearchBox) to the screen container with the statement:

```
SearchContainer.AddBlock(SearchBlock);
```

By default, the container object will supply a 'Save' button. However in this case we are going to create a Search page, so we will change the attributes of the default button with the following code:

```
//Change the label and image on the default button
SearchContainer.ButtonTitle='Search';
SearchContainer.ButtonImage='Search.Gif';
```

The next section of code specifies what happens when the user submits a search. To determine whether we should display a search results grid, we must check the Mode property of the CRM object. CRM.Mode=Save is true when we have just submitted a form to either save an entry or perform a search.Save is a constant defined in the include file (see above).

```
//Show results of search
if (CRM.Mode == Save)
{
       var resultsBlock = CRM.GetBlock('PersonGrid');
       resultsBlock.ArgObj = SearchBlock;
       SearchContainer.AddBlock(resultsBlock);
}
```

Other possible values for the Mode property areView (the screen is in read-only state) and Edit (the screen is in an editable state).

In this case, we have specified what action to take after the search has been submitted. Again, the CRM.GetBlock method has been called, here to return a PersonGrid block. PersonGrid is and instance of the CRMListBlock object. The fields displayed in this list can be viewed at:

Administration | Customization | Person | Lists | Person Grid

The returned PersonGrid block has been assigned the variable name resultsBlock. Next, the CRMBlock property, ArgObj, which is a base property implemented by all subclasses (such as the PersonGrid block in this case) is used to pass the SearchBlock as a parameter for populating the list. In other words, we have specified that the list resultsBlock is to take the search screen SearchBlock as a parameter, and therefore the contents of the list generated by resultsBlock will be determined by the values of the fields on the search screen SearchBlock.

Finally, we used the AddBlock method of the container object to add the resultsBlock object.

Next we must set the CRM.Mode property. As this is a search form (rather than a data-entry form), we need to ensure that the page is not displayed in read-only mode.

```
if (!Defined(Request.Form))
{
       // first time - display mode
       CRM.Mode = Edit;
}
else
{
       // mode is always Save
       CRM.Mode = Save;
}
```

Finally we call SearchContainer.Execute which returns the HTML code for the container we have created. This HTML code is passed as a parameter to the CRM.AddContent method.

Multiple pieces of content can be stored to the AddContent method. Finally, all of the content is rendered to the browser by calling CRM.GetPage(). CRM.GetPage() produces properly formatted HTML and incorporates the tabs at the top of the page. We can store this HTML text output into a variable, sHTML, and then use Response.Write() to send this to the browser.

```
CRM.AddContent(SearchContainer.Execute());
var sHTML = CRM.GetPage();
Response.Write(sHTML);
```

The AddContent and GetPage methods will ensure that the correct format is sent to the browser (i.e. it will be rendered properly if the browser client is a mobile device).

Note: If a coaching caption is associated with the returned page, and the coaching caption settings are enabled, the displayed ASP page will feature coaching text. However, it is not possible at present to specify coaching text via the API.

Here is the complete code for the page:

```
<!-- #include file ="sagecrm.js"-->
  <%
// get an empty container block
var SearchContainer = CRM.GetBlock('Container');
// add the Person Search Box
var SearchBlock = CRM.GetBlock('PersonSearchBox');
SearchContainer.AddBlock(SearchBlock);
//Change the label and image on the default button
SearchContainer.ButtonTitle='Search';
SearchContainer.ButtonImage='Search.Gif';
//if button has been pressed then add the list block to show
```

```
//results of search
if (CRM.Mode == 2)
{
       var resultsBlock = CRM.GetBlock('PersonGrid');
       resultsBlock.ArgObj = SearchBlock;
       SearchContainer.AddBlock(resultsBlock);
}
if (!Defined(Request.Form))
{
       // first time - display mode
       CRM.Mode = Edit;
}
else
{
       // mode is always Save
       CRM.Mode = Save;
}
CRM.AddContent(SearchContainer.Execute());
var sHTML = CRM.GetPage();
Response.Write(sHTML);
```

Integrating the ASP Page Into CRM

Once you've finished writing the ASP page, making it accessible within CRM is a two-stage process:

- Placing the ASP file in the Custom Pages folder.
- Specifying details of the ASP file in the Administration section of Sage CRM.

Placing the ASP file in the Custom Pages folder:

- Copy the saved ASP file into the Custom Pages folder. In a typical install, the path name will be ...\Program Files\CRM\(Your Install Name)\WWWRoot\CustomPages\

Enabling a CRM tab to display the ASP file:

1. Select **Administration | Customization**. A list of primary entities that can be customized are displayed. Secondary entities available for customization can be accessed from the drop-down list at the bottom of the screen.
2. Select the **Company** entity. A sequence of tabs indicating customizable areas of the selected entity is displayed.
3. Select the **Tabs** tab.
4. In the table displayed, click the **Customize** icon beside the hyperlink for Company. The Customize tabs page, which allows you to customize existing tabs or create new ones, is displayed.
5. In this case, we want to add a new tab to display the ASP page we've created. So in the Properties panel, enter the name **Searchbox** in the Caption field and click on the **Add** button. A tab named **Searchbox** has been added to the list under the heading Desktop HTML Tab Group Contents. You can adjust this new tab's position in the tab sequence by using the up and down arrows beside the list.
6. Next, we need to specify that the new tab displays the file copied to the Custom Pages folder. In the Actions drop-down field, select the **Customfile** option. This indicates to the system that the tab is using a file from the Custom Pages folder.
7. Next, you type your file name (i.e. **MySearchBox.asp**) into the Custom File field.
8. Click on the **Update** button to add these details to Searchbox tab.
9. Click on the **Save** button to confirm the changes.

To view the ASP page within the tab, click into any company details screen. A new tab, SearchBox, is now visible.

Creating Custom Queries

Creating a RecordSet using CreateQueryObj

The CRM API allows easy access to the database for selecting and updating data. The CreateQueryObj method can return a record set that can be viewed and manipulated in the same way as, for example, an ADO record set.

Here is an example of an ASP page that uses the CRMQuery object to present the user with a filtered list of companies. In this case we will show only those companies from the software sector that are ranked as prospects.

```
<!-- #include file ="sagecrm.js"-->
<%
Query= CRM.CreateQueryObj("SELECT * FROM Company WHERE Comp_Deleted IS NULL AND
Comp_Type = 'Prospect'
AND Comp_Sector = 'Finance'");
Query.SelectSql();
While (!Query.eof)
{
       CRM.AddContent(Query.FieldValue("comp_name")+'&nbsp;&nbsp');
       CRM.AddContent(Query.FieldValue("comp_address")+'<BR>');
       Query.NextRecord();
}
Response.Write(CRM.GetPage());
%>
```

The CRM object's CreateQueryObj method returns a CRMQuery object by specifying a valid SQL statement as a parameter. The default database is used in this example, but it is possible to specify another database by adding a second parameter to the CreateQueryObj method call.

You can expand the scope of your query using such relational database features as Joins and Views. You can also examine, or copy if you wish, system and custom views either by going to Administration | Customization | <EntityName> | Views or scanning the list of Views in a database management tool such as SQL Server Enterprise Manager.

In this case the returned query object is assigned the name Query. The Query.SelectSql method executes the SELECT query. It is also possible to use the Query.ExeqSql method to run queries that do not return records, such as DELETE, UPDATE, and INSERT.

As well as encapsulating the components to access and update the database, the Query object also stores the returned data.

In this example we use a Javascript WHILE statement to iterate through the returned records until an end-of-file marker is found. Within the loop, the values stored for each company's name and e-mail address are retrieved using the Query object's FieldValue property.

We then use the AddContent method to build up a block of HTML that will be displayed when we write the output of the CRM.GetPage method to our ASP page using the Javascript Response.Write.

Using a ListBlock to provide better format

Although this query generates a filtered list containing information the developer specified, it would be better to have data formatted in the same way as the CRM interface.

This can easily be done by using the CRMListBlock Object, which can be initialized by supplying the CRM.GetBlock method with the appropriate parameter. In the code below we create the new object, NewList, by calling CRM.GetBlock("list").

```
Please refer to Developer Help files for code sample
```

In the example above, the SELECT query for retrieving a filtered list of company records remains unchanged from that used by a CRMQuery block. However, in this case, the query string is being passed to the CRMListBlock object's SelectSql property. As always, the CRMListBlock, assigned the name NewList, was obtained using the CRM.GetBlock method.

We can then specify which columns to display by using the AddGridCol method. In this case, we're going to show values relating to the company name and company e-mail address.

You can pass only field names that are returned by the SQL query. You can also enter optional addition parameters for this method that specify the position of the column in the tabular list and whether the column contents are ordered.

The Execute method returns HTML to display the selected columns in a properly formatted list.

Note: The list object has abstracted the process of looping through the available records, so a WHILE statement testing an EOF marker is not needed.

The returned list is passed to the CRM AddContent method. Again, CRM.GetPage is used in conjunction with Response.Write to send the output onto the client's screen.

Using the CreateRecord method for easier Recordset manipulation

The ability to construct SQL strings and pass them to CRM objects enables developers to apply relational database concepts to the presentation and manipulation of CRM data. However, developers not familiar with SQL or requiring a more abstract approach to handling data can use the CRMRecord Object which can be used to create an updateable record object.

```
<!-- #include file = "sagecrm.js"-->
<%
Comp = CRM.CreateRecord('company');
Comp.item('comp_Name') = '4D Communications International';
Comp.item('comp_emailaddress') = 'www.4DCommInter';
Comp.SaveChanges();
block=CRM.GetBlock("companygrid");
CRM.AddContent(block.execute(''));
Response.Write(CRM.GetPage());
%>
```

In this example, the CRM object's CreateRecord method is called, with the company table specified as an argument.

Rather than having to use a SQL INSERT statement, new values can now be assigned to fields using the following syntax:

Comp.item ('fieldname') = 'strvalue';

This is similar to opening an ADO updateable recordset. For example, Comp.item ('comp_Name') = '4D Communications International' is equivalent to the SQL statement: "INSERT INTO Company (Comp_Name) Values ('4D Communications').

Note: You are not required to specify the item property (because it is the default property for this object), so
Comp.item('comp_Name') = '4D Communications International' and Comp('comp_Name') = '4D Communications International' are treated in the same way.

The SaveChanges method must then be called to updates the database with the insertions.

Finally we can view the updated records by selecting an appropriate block to display company details. In the above example, we have used a company grid.

Understanding "Context" and the GetContextInfo Method

In CRM development terms, 'Context' refers to information about the current situation of the user within the software interface. As a very simple example, if the user is looking at the Company summary screen, then he is said to be in the context of that company.

This means that it will be possible for the developer to easily access contextual information, such as data from the Company and Person tables, that relate directly to what the user is looking at. Data from the User table concerning the current user is also available within the context framework.

The GetContextInfo method is used to access information about the current context. In the sample below, the GetContextInfo method and the CaseList block are used to retrieve and display the cases in progress associated with the company currently being viewed by the user:

```
<!-- #include file = "sagecrm.js"-->
<%
ThisCompanyId = CRM.GetContextInfo('Company','Comp_CompanyId');
SearchSql = "Case_PrimaryCompanyId="+ ThisCompanyId;
SearchSql += " and Case_Status='In Progress'";
CaseListBlock = CRM.GetBlock('CaseList');
CRM.AddContent(CaseListBlock.Execute(SearchSql));
Response.Write(CRM.GetPage());
%>
```

GetContextInfo accepts two parameters - entity (e.g. Company, Person, Opportunity, Case, Solution, or Lead or marketing- and user-related entities) and the fieldname that we want to access. In this example, we're obtaining the unique company ID.

This information is then used in a SQL statement, which will be used to select all cases that match the context company's ID and have a "In Progress" status.

Next, the CRM GetBlock method is used to return a CRMListBlock object of type "CaseList," which is assigned the variable name CaseListBlock.

In the next line, we call the CaseListBlock's Execute function, passing in the SELECT statement used to extract the required cases. The populated CaseList is in turn passed as an argument to the CRM.AddContent method to store the page in memory.

Finally, we call Response.Write to output the generated HTML to the screen.

Client-Side Scripting with JavaScript and DOM

Although server- side ASP offers full access to the CRM API, it is sometimes more convenient and quicker to handle certain tasks with code that runs on the client.

Obviously, client-side JavaScript can be used in the same ASP page that contains server-side code. Such scripts are processed by the browser, thereby eliminating round-trips to the server.

Client-side scripting in CRM ASP pages is handled in the normal way using the JavaScript scripting language and the Document Object Model (DOM). JavaScript is a full-featured programming language that allows developers to use an extensive library of classes, methods, properties, and event handlers defined. The DOM provides a tree-form representation of a HTML page that can be searched up or down to locate specific nodes.

Example: Client-side Validation

The following example shows a client-side JavaScript code that accesses the DOM of a CRM generated web page in order to capture certain events and validate them. In most cases you will be able to achieve your validation objectives by using the tools available in the Field Customization chapter in the System Administrator Guide, or failing that using Field Level Scripting (page 3-10).

One of the most common tasks for client-side scripts is validation. Client-side validation saves time by checking, for example, whether a required field contains information before the page is sent to the server for server-side validation.

The script below is a very basic example of validation as it merely checks whether the last name field in the "PersonSearchBox" is empty before the search details are submitted to the server. However, it is a useful example of how JavaScript in conjunction with DOM enables the developer to handle events and manipulate the interface after the page has been loaded into the browser's memory.

```
<script language="javascript">
document.attachEvent("onclick", Validate);
function Validate()
{
       var oSource = window.event.srcElement;
       if((oSource.className ==
"ButtonItem")||(oSource.parentNode.className=="ButtonItem"))
       {
               if(document.forms[0].pers_lastname.value=="")
               {
                       alert("Please specify a last name"); return false;
               }
               else
               {
                       return true;
               }
       }
}
</script>
```

In this example we have set up an event handler with the attachEvent method of the document object, which is the highest-level node in the DOM representation of a HTML page.

The method's two parameters, "onclick" and Validate, specify that when a click event occurs on the document, the Validate function will handle that action. The attachEvent method, which is Internet Explorer-specific syntax, is the recommended approach for setting up event handlers as it allows for events to have multiple handlers. Attaching handlers to objects directly (with window.onload, for example) has the disadvantage of allowing you to associate only one function with an event.

In the Validate function, window.event.srcElement retrieves the object that fired the event and assigns it the variable name oSource.

```
if((oSource.className ==
"ButtonItem")||(oSource.parentNode.className=="ButtonItem"))
```

The subsequent IF statement checks whether the object that fired the event was a button. The className property employed to check the object is typically used to associate a particular style rule in a style sheet with the element. In this case, we're interested in an object assigned the ButtonItem style. In this case, we've also specified "oSource.parentNode.className" because the user can click on a *.gif image of a button that is nested in an element with the className attribute set to

ButtonItem. The parentNode, as the name indicates, allows us to check the className attribute at the next level of the document hierarchy.

Incidentally, one way of examining the document structure in detail is to use the JavaScript alertbox to display the HTML of a section of the document that is clicked. The innerHTML property of the srcElement object facilitates this:

```
alert(window.event.srcElement.innerHTML);
```

The statement document.forms[0].pers_lastname.value=="" uses DOM in a straightforward fashion (each node after the dot "." represents a further "level" down in the DOM hierarchy). So this statement drills down to the pers_lastname field and checks whether it is empty. If it is blank, the event handler displays a warning alertbox and returns "false," indicating that the form cannot be submitted

Note that CRM provides via the Customization interface, and in most cases it will be preferable to use those when working with CRM entry blocks. Please refer to Field Level Scripting (page 3-10). See also the Field Customization chapter in the System Administrator Guide.

Accessing User Information

The CurrentUser object

Client-side scripting can actually leverage important information about the current user of the system with the CurrentUser object. This is made available to the client through the code contained in the include file, referenced as <!-- #include file ="sagecrm.js"-->. Behind the scenes, the supporting code parses the query string shown in the browser's status bar to define the Session ID that identifies the user.

The CurrentUser object provides access to extensive information about the user, stored in the following columns in the "Users" table in the CRM database.

> user_userid, user_primarychannelid, user_logon, user_lastname, user_firstname,user_language, user_department, user_resource, user_externallogonallowed,user_isterritorymanager, user_per_user, user_per_product, user_per_currency,user_per_data, user_offlineaccessallowed, user_per_customise, user_minmemory,user_maxmemory, user_title, user_location, user_deskid, user_per_infoadmin, user_device_machinename, user_per_solutions, user_prf

The combination of JavaScript, the DOM, and CurrentUser information enable the developer to configure screens and create event handlers that respond to the current user's profile.

In this example, we show how to extend a search screen by configuring the search options depending on the user's profile. For this scenario, we assume that all users attached to the Telemarketing department are focused on customers belong to the "Europe" territory. So it would be quite useful if "Europe" was automatically selected in the drop-down list of Territories if this customized Search page is accessed by a member of the Telemarketing department.

The JavaScript below uses the CurrentUser object to ascertain the user's department.

```
<script language="Javascript">
var channeldept = CurrentUser.user_department;
window.attachEvent("onload", Populate);
function Populate()
{
        if (channeldept=="Telemarketing")
        {
                var oSource = document.all.item("pers_secterr");
                var dropdownloop = document.all.item("pers_secterr").length;
                var setIndex = 0;
                for (i=0;i<dropdownloop;i++)
                {
                        checkText = oSource.options[i].text;
                        if(checkText.indexOf("Europe") != -1)
```

```
                    {
                            setIndex = i;
                            break;
                    }
            }
            document.all.item("pers_secterr").selectedIndex =setIndex;
    }
    else
    {
            document.all.item("pers_secterr").selectedIndex = 0;
    }
}
</script>
```

The information contained by "CurrentUser.user_department" is assigned to the variable channeldept. Next an attachEvent method specifies that a function called "Populate" is called when the page is loaded ("onload").

The first line in the Populate event handler is to test whether the user's department is actually "Telemarketing." If it is, we use item from the all() collection of the Document object to find the "pers_secterr" field, which is the drop-down list featuring the available territories. (A more advanced solution could be to use a switch statement to indicate a range of actions depending on a variable's value).

The rest of the code in the if statement loops through the options in the list in search of the entry for "Europe". The JavaScript method indexOf checks whether the current option text contains "Europe". If it does, the setIndex variable is set at this point in the loop and ensures that the Europe option is visible when the page is loaded.

One of the limitations of using client-side scripting, even given the flexibility provided by the DOM, is that we are limited to checking and modifying information that is available on screen. If we want to validate user-input based on data stored in the CRM database, we need to execute on the server.

Fortunately, as well as allowing ASP pages to define custom behavior, CRM enables you to enter code directly through the interface, enabling developers to see clearly the connection between interface elements and their event-handling code.

Scripting in the CRM Interface

Once the limitations and benefits associated with server-side and client-side scripting in a custom ASP file are clear, the scripting options available within the CRM interface become easy to grasp. Some scripting fields expect server-side code, others are designed to accommodate client-side script. The easiest approach to deciding what to enter is to judge whether you will need to use objects from the CRMBlocks hierarchy. If you do, you'll probably write server-side code. On the other hand, code that primarily interacts with the interface can run on the client side. Scripts, such as SQL queries, that manipulate databases need server resources to access the specified records.

There are several areas where you can enter scripts in CRM, with the chosen location reflecting the scope of the event to be handled (a change in a single field all the way up to the addition or deletion of a entire entity) and the action to be taken (performing a simple validation up to adding an entity).

Field Level Scripting

CreateScript, OnChangeScript, and ValidateScript

On the lowest level, when managing the display of fields on an Entity screen (by choosing Administration | Customization | <EntityName> | Screens| <ScreenName>), you can associate behaviors with individual fields. There are three "boxes" that allow you to write code that is executed when an event affects the field selected in the contents screen: CreateScript, OnChangeScript, and ValidateScript.

Note: A simplified code-free method for defining access rights to fields has been available since release 6.0 of Sage CRM. See the section on field security in the Field Customization chapter in the System Administrator Guide.

Scripts written in the CreateScript and ValidateScript sections run on the server, whereas OnChangeScript code runs on the client. We have already seen in the discussion of client-side scripting in ASP pages how we can handle events such as the user clicking a button. We can enter in the OnChangeScript box similar scripts, which are executed when the JavaScript event OnChange occurs on the specified field. For example, if we have added code for the Company Type drop-down list, it is executed when the user clicks to view options.

One of the convenient aspects of entering code in the interface is that it's not necessary to add include statements or <script> tags. In addition, we can use the JavaScript "this" keyword to refer to the object that triggers the code. For example, this is a simple if statement for the comp_type field that disables another field if it contains the "Partner" value.

```
if(this.value =='Partner')
{
       comp_revenue.disabled = 'true';
}
```

Another simple OnChange script can alert the user to change an entry before validation is triggered.

```
if(this.value.toUpperCase() == this.value)
{
       window.alert(this.name + ' has been entered all uppercase');
}
```

Running on the server, scripts that execute when the page is loaded into CRM (CreateScript) or when the user clicks the submit button (ValidateScript) can access the CRM API without having to include files such as SAGECRM.JS. When writing validation scripts that execute on the server, it is worth being aware of three system variables that greatly ease the process of trapping information and providing feedback: Values(), Valid, and ErrorStr:

- Values( ) collection method: This system variable holds the inbound values of data coming into the system. This is an obvious choice for validation scripts. Values() allows you to read any variable in the QueryString in CreateScripts. You can test for the dominant key and therefore context simply like this: var x = Values("Key0"); Valid = false; ErrorStr = x;

- Valid: This system variable has a default value of "True", but when set to "False" marks the current entry as invalid. This determines whether the ErrorStr value should be displayed on the screen. Note: The effect of Valid being set to false has slightly different behavior in parts of the system. It just displays an ErrorStr in Create scripts and blocks the commitment of data in Validate scripts.
  In more advanced situations, this variable controls the display of workflow rule buttons in Primary, Transition, and Global workflow rules. In Conditional rules, setting the value of Valid to false causes the rule to execute an alternative set of actions. In Table Level and Entity Level scripts, which update records in response to an action, the Valid variable may simply cause the ErrorStr to be set to the browser, or it can block the whole transaction in a way similar to the Validation rules.

- ErrorStr: This system variable returns the string in the error content (red bar at the top of the screen).

We can see how these variables can be used in a simple code snippet used to validate the fields when specifying details for an opportunity.

```
if (Values('oppo_type')="Mix")
{
        Valid = false;
        ErrorStr = 'This Mix type is temporarily not supported';
}
```

In the sample above, the code could be placed in the Validate script box for the Opportunity Type field on the Opportunity Detail Screen, which is accessed by selecting Administration |Customization | Opportunity | Screens | Opportunity Detail Screen.

The code sample checks the information being returned using the Values ( ) collection, specifying that the value in the oppo_certainty field will be tested. Here we are specifying that some action will be taken if "Mix" is selected as the Opportunity type.

Note: You can't put the script on any field if the Opportunity Type field is the one you want to validate.

To halt validation, Valid is set to false and a message is assigned to the ErrorStr. When triggered, the ErrorStr text appears in a red bar above the screen.

It is worth noting that although this simple example is confined to testing information on the screen, it is possible to use other blocks accessible through the API so you can base validation on entity information not currently displayed onscreen. Moreover, the CRMEntryBlock object also provides a variety of properties-such as Required, ReadOnly, MaxLength, CreateScript, OnChangeScript, and ValidateScript-that enable you to control and respond to the values entered into a screen.

The CreateScript, OnChangeScript, and ValidateScript boxes for custom event handling are also available elsewhere in the system, specifically when defining the rules for workflow (which manages the progress of an Entity such as an Opportunity, for example, from lead stage to completed) and escalations (which are chiefly used to remind users of upcoming events and tasks to be completed or to notify them about the impending expiry of, say, an order). For these aspects of the system, which will mainly use SQL queries to check database information, refer to the relevant chapters in the *System Administrator Guide*.

Table Level Scripting

Another significant area where you can enter code using the interface is the Table Script screen, accessed by selecting Administration | Customization | <EntityName> | TableScripts.

This screen allows you to specify actions to be executed when a record is inserted, updated, or deleted into a specified table (Table Scripts) or when an entity is updated, inserted, or deleted (Entity Scripts).

For detailed information on writing such scripts, refer to Introduction to Table and Entity Level Scripts (page 5-12). However, if you recognize that the developer's task chiefly consists of responding to four "triggers"-InsertRecord(), PostInsertRecord(), UpdateRecord(), and DeleteRecord()-using server-side scripts that leverage the API, the process of drilling down to actual implementations should be straightforward.

Chapter 4: Customization Basics

In this chapter you will learn how to:

- Get an overview of CRM blocks.
- Get an overview of CRM objects.
- Add Help to custom pages.
- Customize lists.
- Customize screens.
- Customize buttons.
- Customize the classic dashboard.
- Customize the interactive dashboard.
- Customize blocks.
- Customize system menus.
- Customize tabs.

CRM Blocks Overview

The Extensibility Module is required to create and customize blocks in Administration | Customization | <Entity> | Blocks.

- All lists, screens, and fields that you create are actually blocks within the system (ListBlock, EntryGroupBlock and EntryBlock).
- Any new blocks you create must be based on one of the existing standard CRM blocks, for example Container, EntryGroup, or Marquee.
- You associate all new blocks created with the entity to which they relate.
- You can create and display new blocks of data from external tables and databases that connect to CRM. For more information on Customizing Tables refer to Database Customization (page 5-1).
- Once you create a new block, you can reference and run the block from within CRM and in ASP pages.

Block Naming and Custom Blocks

Aside from referring to this guide, you can use several methods to check the names of blocks referenced in an ASP page. The approach taken depends on the type of block used. Block names can be any of the following:

- The generic name of the block.
- The name of any custom screen (Entry block).
- The name of any custom list (List block).
- The name of any block that you have created.

Generic Block Names

All blocks are based on one of the CRM basic block types.

You can reference CRM's generic block names from the interface.

To find out the generic block names:

1. Select **Administration | Customization | <Entity>**.
2. Select the **Blocks** tab.
3. Click on the **New** button. All the custom block names are listed in the Block Type list. The block names correspond to the main block names in the CRM block hierarchical diagram. All other blocks are based on one of these block types.

Name of Custom Screen (EntryGroupBlock)

A standard installation includes a number of pre-defined custom screens that EM enables you to manipulate or copy for your own implementation. Once you have determined the exact block names, you can call these blocks from your ASP page.

You can determine the predefined custom screen names:

- From the system interface.
- From your database tables.

From the System Interface

To find out the name of the company entry screen:

1. Select **Administration | Customization | Company**.
2. Select the **Screens** tab. The screen names are displayed in the Screen Caption column. In this example, the screen name is CompanyBoxLong.

From the Database Tables

The other way of finding a full list of Custom Screen names is from the Custom Tables in your database. For example, if you are using SQL Server, you can find the block names from the database in Enterprise Manager. In this example, the custom screen names are listed under Custom_Screens in the SeaP_SearchBoxName field.

Name of a Custom List (ListBlock)

The installation includes a number of pre-defined custom lists that EM enables you to manipulate or copy for your own implementation.

There are two ways of determining predefined custom list names:

- From the system interface.
- From the database tables.

From the System Interface

To find out the name of the company list (ListBox):

1. Select **Administration | Customization | Company**.
2. Select the **Lists** tab. The list names are displayed in the List Name column. In this example, the list name is CompanyGrid.

From the Database Tables

The other way of finding a full list of block names is from Custom Tables in the database. For example, if you are using SQL Server, you can find the block names from the database in Enterprise Manager. In this example, the custom list names are listed under Custom_Lists in the GriP_ GridName field.

The Name of a Block that you Have Created

You can create your own blocks from the system interface. Once you have created a new block you can refer to it from an ASP page using the name you assigned it.

Adding Help to Custom Pages

This section describes how to add a help button and a link to context-sensitive help page from an ASP page. Please refer to the System Administrator Guide for more information on customizing help in the standard CRM system.

Step 1: Create a Custom Help Page

Use any HTML editor to create a HTML help page and save it to the following folder:

<Install Path><Install Name>\WWWRoot\HELP\EN\Main Menu\Content\User

For example:

C:\Program Files\Sage\CRM\CRM\WWWRoot\HELP\EN\Main Menu\Content\User

Here is an example of a complete custom help file:

```
Please refer to Developer Help files for code sample
```

Note: Make sure that you maintain backup copies of your custom help page away from the installation. Also note that any changes you make to existing CRM Help files may be overwritten when you upgrade your CRM installation to a new version or patch release.

Step 2: Add a Help Button to your Custom Page

Once you have saved your Custom Help Page, you can add a Help Button to your ASP page. This is the code to create the button (assuming your Custom Help Page is called FI_ SearchingForCompany.htm):

```
Please refer to Developer Help files for code sample
```

When you run the ASP page within CRM, and click the Help button, the CRM Help system will be launched in a new window and your page will be displayed within it.

Limitations

It is a limitation of this method that the new help page(s) that you create cannot be included in the CRM Help Table of Contents, Index or Search feature.

Lists

How to Create a List

You can create a new list in the following ways:

- In CRM from Administration | Customization | <Entity> | Lists.
- In an ASP page linked to CRM through the customfile action in Administration | Customization | <Entity> | Tabs.

This page describes how to create a list within CRM. For more information on creating a list in an ASP page refer to Building an ASP Page (page 3-1) and ASP Object Reference (page 8-1), specifically CRMListBlock object and CRMGridColBlock object.

You can create your own lists from columns in existing tables or external tables connected to CRM. Once you create a new list you can use different methods to display it.

Creating a New List

To create a new list:

1. Open **Administration | Customization**, and select an entity, for example **Company**.
2. Select the **Lists** tab, and click on the **New** button. The New List Definition page is displayed.
3. Enter the details in the **New List Definition** page.
4. Choose the **Save** button. The new list is displayed.

Field	Description
Name	Name you assign to the List. This is also the block name that you use to reference the list in ASP pages.
Table or View to Select Fields From	The table or view that the fields (columns) on the list are in.
Filter Box Name	The name of the filter box by which you wish to search the list.

To customize a list:

1. Select **Administration | Customization**, and select an entity, for example **Company**.
2. Select the **Lists** tab, and click on the **Customize** button beside the list you want to customize. The Maintain List Definition screen is displayed.

   Note: Selecting the Change button next to the list name allows you to view the New List Definition page.

3. Choose the **Save** button.

For more information on List Customization please refer to the System Administrator Guide.

How to Display a List

New lists can be displayed as follows:

- By creating a tab and using the page that references the list For more informatin, refer to Display a List Using an ASP Page (page 4-4). This option enables you to set the properties of the list before displaying it.
- On a new tab using the runblock action to directly run the list. For more information, refer to Display a List using Runblock and the List Name (page 4-5). Note: If you use the runblock action to run a list, the tab group to which you add the list must be the tab group for the entity on which the list is based. This ensures that when the block is used it will maintain the context for the current entity.

Display a List Using an ASP Page

To display a list created from an ASP file by linking to the file from a custom tab:

1. Create the ASP file. For more information refer to Building an ASP Page (page 3-1) and Introduction to the ASP Object Reference (page 8-2) in this guide.
2. Select **Administration | Customization**, and select an entity, for example, **Company**.
3. Select **Tabs**.
4. Click on the hyperlink of the tab group that you want to add the tab to. The Customize Tabs For screen is displayed.
5. Enter a name for the tab, select the **customfile** option from the Action list and enter the name of the ASP page in the Custom File field.

6. Click on the **Add** button, and choose the **Save** button.
7. To display the list, click the new tab

Display a List using Runblock and the List Name

To display a list directly from a tab using the runblock tab action:

1. Open **Administration | Customization | <Entity>**, and select **Tabs**.
2. Click the hyperlink of the tab group that you wish to add the tab to. The Customize Tabs For screen is displayed.
3. From the **Properties** panel, type a name for the tab in the Caption field.
4. Select the **runblock** option from the Action list and enter the list name you created earlier in the Block Name field.
5. Click the **Add** button and then the **Save** button. To view the list, you select the new tab.

Screens

How to Create a Screen

You can create a new screen in two different ways:

- In CRM within Administration | Customization | <Entity> | Screens.
- In an ASP page linked to CRM through the customfile action within Administration | Customization | <Entity> | Tabs.

This page describes how to create a screen within CRM. For more information on creating a screen in an ASP page refer toIntroduction to the ASP Object Reference (page 8-2), especially CRMEntryGroupBlock Object (page 8-55) and CRMEntryBlock Object (page 8-43).

You can create your own screens from columns in existing tables or external tables connected to CRM. Once you create a new screen you can use different methods to display it. You can also create a block from the screen and use the block properties to customize the appearance of the screen.

Creating a New Screen

To create a new screen:

1. Select **Administration | Customization**, and select an entity, for example **Company**.
2. Select the **Screens** tab and click on the **New** button. The New Screen Definition page is displayed.
3. From the Screen Type list select a screen type, for example Entry Screen or Search Screen. If you select Search Screen, an additional field called Associated List is displayed on the New Screen Definition page. This enables you to associate a list with the screen.
4. Enter the details in the **New Screen Definition** page. See the table below for descriptions of the fields on the screen.
5. Choose the **Save** button. The new screen name is displayed in the list of screens.

The table below explains the standard fields in the New Screen Definition input form.

Field	Description
Screen Name	Name you assign to the screen. This is also the block name that you reference the screen from in ASP pages, but it is not the name that is displayed on the actual screen.

Field	Description
Screen Caption	The actual name that is displayed at the top of the screen that you create, for example the My New Entry Screen.
Screen Type	You can either create an Entry screen or a Search screen.
Associated View	The name of the view that contains the screen fields.
Foreign Table	If using fields from a foreign table, enter the table name here.
Foreign Table Column	The column that uniquely relates the foreign table to the CRM table and that has a corresponding field on the CRM table.
Associated List	Only applicable if it is a search screen. The name of the list you want to associate the search screen with.

Customizing the New Screen

Once you have created a screen, you need to define fields for it.

To customize the new screen:

1. From **Administration | Customization**, select the entity in which you created the new screen.
2. Select the **Screens** tab.
3. Click on the **Customize** button beside the new screen you created. The **Desktop HTML Screen Contents** page is displayed.

   Note: Selecting the Change button next to the screen name allows you to view the **Maintain Screen Definition** page.

4. Add the fields you wish to display on the screen.
5. When you have added all the fields that you wish to display on the screen, click the **Save** button.

How to Display a Screen

New screens can be displayed in a number of different ways:

- On a new tab using the runblock action to directly run the screen. See Display a Screen Using Runblock and Screen Name (page 4-7).

  Note: If you use the runblock action to run a screen, the tab group to which you add the screen must be the tab group for the entity on which the screen is based. This ensures that when the block is used it will maintain the context for the current entity. For example, if you create a new entry screen for the Company table to edit extra data relating to companies, this can be added to the Company tab group, but will not work on any other tab group.

- By creating a block from the screen, then creating a new tab and using the runblock action to run the block. This option enables you to set the properties of the block before displaying it. **Note**: Similar to running a screen directly, if you use the runblock action to run a block, the tab group to which you add the block must be the tab group for the entity on which the block is based. See Display a screen using Runblock with a Custom Block (page 4-7).
- By creating a new tab and using the customfile action to link to an ASP page in which the screen is referenced. This option enables you to set the properties of the screen before displaying it. See Display a screen with an ASP page (page 4-7).

Display a Screen Using Runblock and Screen Name

To display a screen directly from a tab using the runblock tab action:

1. Open **Administration | Customization | <Entity>**, and select **Tabs**.
2. Click the hyper link of the tab group that you wish to add the tab to. The Customize Tabs For screen is displayed.
3. From the Properties panel, type a name for the screen in the Caption field.
4. Select the **runblock** option from the Action list and enter the screen name you created earlier in the Block Name field.
5. Click the **Add** button and then the **Save** button.
6. To view the screen, you select the new tab.

Display a screen using Runblock with a Custom Block

To create a block from a screen, change the properties, and use the runblock tab action:

1. Select **Administration | Customization | <Entity> | Blocks**, and click on the **New** button. The New Block screen is displayed.
2. Enter the block details and select the name of the screen you created from the Use Group list.
3. Choose the **Save** button. The block is displayed in the list of blocks for the entity.
4. Click the hyperlink of the block or select the **Customize** button. The Properties screen for the block is displayed.
5. Enter the properties and then choose the **Save** button. Please refer to Customizing a Block (page 4-19) for more information on block properties.
6. Select **Administration | Customization | <Entity> | Tabs**.
7. Select the context to which the new block belongs from the list in the context area of the screen.
8. Click the hyperlink of the tab group that you wish to add the tab to. The Customize Tabs For screen is displayed.
9. On the Properties panel, select the **runblock** option from the Action list and enter the block name in the Block Name field.
10. Click the **Add** button and choose **Save**.
11. To view the screen, select the new tab.

Display a screen with an ASP page

To display a screen created from an ASP file by linking to the file from a custom tab:

1. Create the ASP file and save it in the Custom Pages folder of your CRM installation directory. For more information refer to Building an ASP Page (page 3-1) and Introduction to the ASP

Object Reference (page 8-2).

2. Select **Administration | Customization | Tabs** and select a context.
3. Click the hyperlink of the tab group that you wish to add the tab to. The Customize Tabs For screen is displayed.
4. From the Properties panel, select the **customfile** option from the Action list and enter the name of the ASP page in the Custom File field.
5. Click the **Add** button and choose **Save**.
6. To view the screen, select the new tab.

Buttons

Creating Button Groups

You can define button groups that will appear on specific CRM screens and which will give users access to custom functionality created by you. For example, say you want to add custom ASP pages to the Cases Summary screen so you can give users access to extra information relating to the management of cases in your company. You can create a button group that will act as a place holder for displaying buttons on the Case Summary screen.

To create a button group:

1. Select **Administration | Advanced Customization | Button Groups**. The Button Groups list is displayed.
2. Click on the **New** button. The New Button Group Definition page is displayed.
3. Type the name of the new button group in the Name field. For example, **Case Summary Button Group**.
4. Select the CRM screen you want the button group to display on from the Select Action drop-down list. For example, **casesummary**.
5. Click on the **Save** button. The new button group is displayed on the Button Groups list.

Adding Buttons to Button Groups

Once you have created a button group you can add buttons to it that will display on the relevant CRM screen. For example, say you have created two ASP pages relating to case management, one that will provide a current list of who is responsible for all live cases and a second one that provides a current picture of all recently closed cases. You have created a button group called Case Summary Button Group and you would like to add two buttons to this group, each of which will bring the user to one of the newly created ASP pages.

To add a button to a button group:

1. Select **Administration | Advanced Customization | Button Groups.** The Button Groups list is displayed.
2. Click on the **Customize** button for the Case Summary Button Group. The Customize Button Group page is displayed.
3. Type the name of the new button in the Caption field, for example **Closed Case List**, and select **Customfile** from the Action drop-down list.
4. Type the name of the associated ASP file in the Custom File field, for example, **CLOSEDCASELIST.ASP**.
5. Select the image you would like to display with the button caption from the Bitmap drop-down list. For example, **DEFAULTBUTTONGROUP.GIF**.

6. Click on the **Add** button. The new button is displayed in the Desktop HTML Button Group Contents area. You can use the up and down arrows to position a new button relative to other buttons you have created.
7. Click on the **Save** button. The new buttons are displayed on the summary screen for any case.

Viewing Button Groups

Once you have added buttons to a button group the buttons will appear automatically on the relevant CRM screen. Say you have added two new buttons to a newly created button group on the Case Summary screen. You can view the new button group by opening the Case Summary page for any case.

To view a button group:

1. Select **Find** | **Case**.
2. Search for, and open any case. The Case Summary page is displayed.

The new button group is displayed on the right-hand side of the page above the Help button.

Restricting Access to Button Groups

When creating or editing a button group, you can use an SQL statement to limit access to individual buttons in that group. For example, you might want to restrict access to buttons in the Case Summary Button Group on the Case Summary screen to members of the Customer Service team. In this example, the Case Summary Button Group would be displayed on the Case Summary page for members of the Customer Service team only.

Note: Buttons in the button group take a user's existing security rights into account. For example, a user must have at least View rights to Cases to be able to open a button group which displays a list of cases.

To limit access to a button group:

1. Select **Administration** | **Advanced Customization** | **Button Groups**. The Button Groups list is displayed.
2. Click on the **Customize** button for the Case Summary Button Group.
3. Select the button to which you want to limit access in the Desktop HTML Button Group Contents area.
4. Type the limiting SQL statement in the SQL field. For example,

```
User_PrimaryChannelId=3
```

5. Click on the **Update** button.
6. Select **Save**.

The button group is available to members of the Customer Service team only.

CRM Classic Dashboard

Customizing The Classic Dashboard

This section provides details on how to create and customize content for CRM Classic Dashboards. An introduction to both Classic and Interactive Dashboards can be found in the *CRM User Guide*. Information on how to setup standard classic dashboards for users can be found in the *CRM System Administrator Guide*.

There are three types of Blocks that can be used on the Classic Dashboard:

- List Block
- Chart Block
- Content Block

There are a number of fields on the Maintain Block Definition page that allow you to set the block up for use on the Classic Dashboard. For an example of the Maintain Block Definition page, go to Administration | Customization | Cases | Blocks and click My Cases.

The fields on the Maintain Block Definition page that have specific relevance to the Classic Dashboard are as follows:

Field	**Description**	**Applies To:**
Width	Recommended that this be left blank so that the block will fill the available space	List Block Chart Block Content Block
Height	Recommended that this be left blank so that the block will fill the available space	List Block Chart Block Content Block
Interactive/Classic Dashboard Level	For a block to be displayed in the list of Available Content in the Classic Dashboard, select one of the following:Available In Dashboards: Block will be in the list of Available Content for all users.Dashboard - Info Manager: Block will be in the list of Available Content for all users with Security Administration rights set to Info Manager or System Administration.Dashboard - Administration Only: Block will be in the list of Available Content for all users with Security Administration rights set to System Administration.	List Block Chart Block Content Block
Double Width Block	When this option is checked, the block is spread over two of the three columns on the Classic Dashboard.	List Block Chart Block Content Block
Long Block	When this option is checked, a longer maximum length for the block is set.	List Block Chart Block Content Block

Field	Description	Applies To:
Dashboard Conditional	This allows you to enter a filter so that only records meeting the condition are met. For example, the 'My Cases' block contains the following conditional to limit records to those that are 'In Progress' and assigned to the current user : `case_assigneduserid=#U and case_status='In Progress';`	List Block
Contents	This field can contain a wide variety of custom content, including for example, content from external web sites. Enter HTML (or Javascript within <script> tags) and the resulting content will be displayed on the Classic Dashboard.	Content Block

Adding a List Block To The Classic Dashboard

This section explains how to make a List Block available to users via the Classic Dashboard Content page (see the CRM User Guide for more information on the Classic Dashboard Content page).

The following is an example of how to add a new Case List Block for use in the Classic Dashboard:

1. Select **Administration | Customization | Cases | Blocks**. A list of existing blocks for the Cases entity is displayed.
2. Select the **New** button. The New Block page is displayed.
3. Type in a name for the block.
4. Set the Block Type to List Block.
5. If you wish to copy an existing block, select it from the Copy Existing Block drop-down. Alternatively, choose a list from the Use Group list.

   If you wish to order the items in your list, you must open the list definition, select the field that you wish to order by, and choose 'Yes' in the Default Order By drop-down. Note that the 'Allow Order By' setting has no effect when the list is viewed on the Classic Dashboard.

6. Click the **Save** button. The new block is displayed in the list of existing blocks for the current entity.
7. Click on the name of the new block you created (or select the **Customize** button). The Maintain Block definition page is displayed.
8. Select **Available In Dashboards** from the Dashboard Level list.
9. Click **Save**. Now when users click on the Modify Dashboard button within the Classic Dashboard tab, the newly created block is displayed in the list of Available Content.

Adding a Content Block To The Classic Dashboard

This section provides an example of how to make a Content Block available to users via the Classic Dashboard Content page (see the CRM User Guide for more information on the Classic Dashboard Content page).

The following example shows how to add a new Content Block for use in the Classic Dashboard. The Content Block will display the contents of the ASP page 'Test.asp'.

1. Create the ASP page that will display the content that you want for the Classic Dashboard. It should be saved in the CRM application's custom pages folder (see Building an ASP Page (page 3-1) for more information about creating ASP pages in CRM).

   If you are creating an ASP page that uses the GetPage() function to display content, you may wish to supress the tab group. In that case you should pass the "none" parameter to the GetPage function:

   ```
   Response.Write(CRM.GetPage("none"));
   ```

2. Select **Administration | Customization | Secondary Entities | System | Blocks**. A list of existing blocks for the System secondary entity is displayed. Note that the System secondary entity contains a number of lists, views, and blocks for CRM components that are not linked to any other CRM entities, for example the News block.
3. Select the **New** button.
4. Enter the Block Name and set the Block Type to **Content Block**. You could use the Copy Existing Block drop-down to make a copy of an existing block, but in this example we will leave the drop-down blank.
5. Select the **Save** button. The new block is displayed in the list of existing blocks for the selected entity.
6. Click on the name of the new block. The Maintain Block Definition page is displayed.
7. Select **Available In Dashboards** from the Dashboard Level list.
8. In the Contents field add the following:

   ```
   Please refer to Developer Help files for code sample
   ```

   This example code contains a Javascript function called 'callPage'. We pass the name of the ASP page ("tesp.asp") as a parameter to the function and it will build the path including the correct session and context information that will allow the ASP page to use CRM blocks.

9. See Customizing The Classic Dashboard (page 4-9) for details of other fields that are relevant to the Classic Dashboard on this screen.
10. Click **Save**. Now when users click on the **Modify Dashboard** button within the Classic Dashboard tab, the newly created block is displayed in the list of Available Content.

Adding a Chart To The Classic Dashboard

This section explains how to make a Chart Block available to users via the Classic Dashboard Content page (see the *CRM User Guide* for more information on the Classic Dashboard Content page).

The following is an example of how to add a new Chart Block (relating to the Cases Entity) for use in the Classic Dashboard:

1. Select **Administration | Customization | Cases | Blocks**. A list of existing blocks for the Cases entity is displayed.
2. Select the **New** button. The New Block page is displayed.
3. Type in a name for the block.
4. Set the Block Type to **Chart Block**.

5. Select an existing block to copy from the **Copy Existing Block** drop-down, or leave the field blank.
6. Choose the **Save** button.The new block is displayed in the list of existing blocks for the selected entity.
7. Click on the name of the new block you created. The Maintain Block definition page is displayed.
8. Select **Available In Dashboards** from the Dashboard Level list.
9. Click **Save**. Now when users click on the Modify Dashboard button within the Classic Dashboard tab, the newly created block is displayed in the list of Available Content.

CRM Interactive Dashboard

Customizing the Interactive Dashboard

This section provides details on how to create and customize content for CRM Interactive Dashboards. An introduction to both Classic and Interactive Dashboards can be found in the *CRM User Guide*.

Content Blocks can be used on the Interactive Dashboard:

There are a number of fields on the Maintain Block Definition page that allow you to set the block up for use on the Interactive Dashboard. For an example of the Maintain Block Definition page, go to Administration | Customization | System | Blocks and click News.

The fields on the Maintain Block Definition page that have specific relevance to the Interactive Dashboard are as follows:

Field	Description
Dashboard Level	For a block to be displayed in the list of Available Content in the Dashboard, select one of the following:Available In Dashboard: Block will be in the list of Available Content for all users.Dashboard - Info Manager Only: Block will be in the list of Available Content for all users with Security Administration rights set to Info Manager or System Administration.Dashboard - Administration Only: Block will be in the list of Available Content for all users with Security Administration rights set to System Administration.
Contents	This field can contain a wide variety of custom content, including for example, content from external web sites. Enter HTML (or Javascript within <script> tags) and the resulting content will be displayed on the Interactive Dashboard.

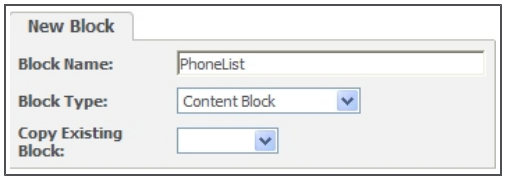
Example: Adding a Content Block to the Interactive Dashboard using the Contents field

This example shows you how to make a simple content block available to end users by pasting HTML into the Contents field on the Maintain Block Definition page.

To add a content block:

1. Copy some simple HTML to your clipboard. For example, if you have a company phone list available in MS Word, you can Save As .HTM, then edit in Notepad, and select and copy the HTML to your clipboard.

2. In CRM, select **Administration | Customization | Secondary Entities | System | Blocks**, and click **New**.
3. Give the block a name, and select **Content Block** from the Block Type drop-down.

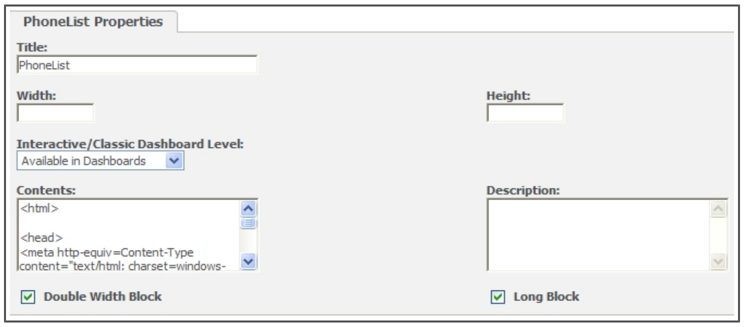


New Block

Block Name:	PhoneList
Block Type:	Content Block
Copy Existing Block:	

New Block page

4. Click **Save**.
5. Click the pencil icon next to the new block you have created.
6. Set the Interactive/Classic Dashboard Level to **Available in Dashboards**.
7. Paste the HTML into the **Contents** field.



PhoneList Properties

Title: PhoneList

Width:

Height:

Interactive/Classic Dashboard Level: Available in Dashboards

Contents:

```
<html>

<head>
<meta http-equiv=Content-Type
content="text/html; charset=windows-
```

Description:

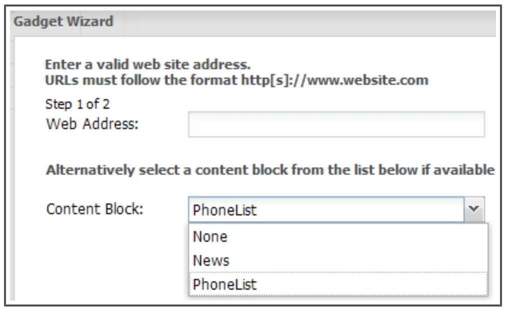
✓ Double Width Block

✓ Long Block

Maintain Block Definition page

The Double Width Block and Long Block properties apply to the Classic Dashboard only. Set these if you want the block to display as a larger sized block on the Classic Dashboard.

8. Click **Save**.
9. To test the block, go to **My CRM | Dashboard**, and either on an existing or new dashboard, create a new **Web Site gadget**.
10. Select the new content block from the **Content Block** drop-down and complete the gadget wizard steps.



Gadget Wizard

Enter a valid web site address.
URLs must follow the format http[s]://www.website.com

Step 1 of 2

Web Address:

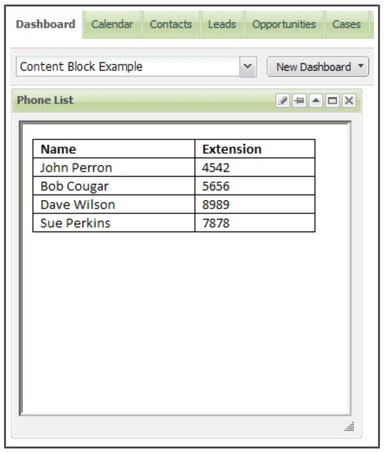
Alternatively select a content block from the list below if available

Content Block: PhoneList

None
News
PhoneList

Web Site gadget

The gadget is displayed on the dashboard, showing the content block that you created.



Name	Extension
John Perron	4542
Bob Cougar	5656
Dave Wilson	8989
Sue Perkins	7878

Content block in Web Site gadget on the Interactive Dashboard

Example: Adding a Content Block To The Interactive Dashboard based on an ASP page

This section provides an example of how to make a Content Block available to users via the Interactive Dashboard Content page (see the CRM User Guide for more information on the Interactive Dashboard Content page).

The following example shows how to add a new Content Block for use in the Interactive Dashboard. The Content Block will display the contents of the ASP page 'Test.asp'.

1. Create the ASP page that will display the content that you want for the Interactive Dashboard. It should be saved in the CRM application's custom pages folder (see Building an ASP Page (page 3-1) for more information about creating ASP pages in CRM).

   If you are creating an ASP page that uses the GetPage() function to display content, you may wish to suppress the tab group. In that case you should pass the "none" parameter to the GetPage function:

   ```
   Response.Write(CRM.GetPage("none"));
   ```

2. Context is available to Content Block.

   ```
   Please refer to Developer Help files for code sample
   ```

3. Select **Administration | Customization | Secondary Entities | System | Blocks**. A list of existing blocks for the System secondary entity is displayed. Note that the System secondary entity contains a number of lists, views, and blocks for CRM components that are not linked to any other CRM entities, for example the News block.
4. Select the **New** button.
5. Enter the Block Name and set the Block Type to **Content Block**. You could use the Copy Existing Block drop-down to make a copy of an existing block, but in this example we will leave the drop-down blank.
6. Select the **Save** button. The new block is displayed in the list of existing blocks for the selected entity.
7. Click on the name of the new block. The Maintain Block Definition page is displayed.
8. Select **Available In Dashboards** from the Dashboard Level list.

9. In the Contents field add the following:

   ```
   Please refer to Developer Help files for code sample
   ```

   This example code contains a Javascript function called 'callPage'. We pass the name of the ASP page ("test.asp") as a parameter to the function and it will build the path including the correct session and context information that will allow the ASP page to use CRM blocks.

10. See Customizing The Interactive DashboardCustomizing The Classic Dashboard (page 4-9) for details of other fields that are relevant to the Interactive Dashboard on this screen.
11. Click **Save**. Now when users click on the **New Gadget :Create Gadget** button within the Interactive Dashboard tab, then selects Web Site as the type of gadget the newly created block is displayed in the list of Content Blocks available.

Adding a Third-party gadget to the Interactive Dashboard

This section explains how to make a 3rd party gadget available to users via the Interactive Dashboard.

A number of methods have been exposed to let you write gadgets in JavaScript, which can communicate with other CRM gadgets via the Event Channel that is used by native CRM gadgets. The data is passed in JSON format.

- scrmGetSageCRMOwner
- scrmRegisterEvent
- scrmPublishEvent
- scrmGetGadgetProperty
- scrmSetGadgetProperty
- scrmSaveGadgetProperties

These third-party gadgets are set up as Web Site gadgets, which link to a file in the wwwroot\staticcontent folder. They can be used in My CRM context or the Company context.

Like List and SData gadgets, third-party gadgets can send or receive data from other gadgets. In other words, they can set the filter or be filtered by other gadgets.

For example, to link a sample third-party gadget to a company list and a company summary gadget:

1. Copy the sample third-party gadget file to ..\ProgramFiles\Sage\CRM\[installname]\WWWRoot\StaticContent
2. Set up a List gadget and a Record Summary gadget – both using company data.
3. Add a Web Site gadget which links to third-party gadget file, for example: #crm_server#/StaticContent/[*gadgetfilename*].html
4. Click on the **Links** button on the Web Site gadget, and link to the company list and Record Summary gadgets.
5. Scroll through the Company List and watch the data on the third-party gadget change.
6. Enter a CRM Company ID on the third-party gadget, click **Publish**, and watch the Record Summary populate with the company data.

scrmPublishEvent

Description	Publishes event
Parameters	gadget: Gadget that calls the method fieldName: field name that is being published jsonData: data to publish, the first property must be called "entityRecordId" and must contain value of field called fieldName

Example	`gadget = parent.scrmGetSageCrmOwner(window.frameElement.id);`
	`parent.scrmRegisterEvent(gadget, "Test Field", "5", null, "BOTH");`
	`message = '{"entityRecordId":"' + document.forms[0].elements["companyId"].value`
	`+ '"}';`
	`parent.scrmPublishEvent(gadget, "Test Field", message);`

scrmRegisterEvent

Description	Registers event classes the gadget will post and/or listen to
Parameters	gadget: Gadget that publishes/receives event; Must not be null. May be obtained by calling scrmGetSageCrmOwner method entityId: ID (custom_tables.bord_tableid) of entity that gadget publishes/may listen to. May be null; fieldType Type of field the gadget publishes/may listen to. May be null; fieldName: Name of field the gadget publishes/may listen to. Must not be null or empty; direction: one of: "PUBLISH" (when gadget publishes information), "LISTEN" (when gadget responds for events in other gadgets), "BOTH"; Must not be null or empty;

Example	`gadget = parent.scrmGetSageCrmOwner(window.frameElement.id);`
	`parent.scrmRegisterEvent(gadget, "Test Field", "5", null, "BOTH");`

scrmGetGadgetProperty

Description	Gets gadget properties
Parameters	gadget: Gadget that calls the method propertyName: name of the property to read

Example	`gadget = parent.scrmGetSageCrmOwner(window.frameElement.id);`
	`propValue = parent.scrmGetGadgetProperty(gadget,`
	`document.forms[0].elements["propertyName"].value);`

scrmSetGadgetProperty

Description	Sets gadget properties so they can be saved
Parameters	gadget: Gadget that calls the method propertyName: name of the property to read propertyValue: value of the property to read

Example	`gadget = parent.scrmGetSageCrmOwner(window.frameElement.id);`
	`parent.scrmSetGadgetProperty(gadget,`
	`document.forms[0].elements["propertyName"].value,`
	`document.forms[0].elements["propertyValue"].value);`
	`parent.scrmSaveGadgetProperties(gadget);`

scrmSaveGadgetProperties

Description	Saves gadget properties on the server so they can be used again
Parameters	gadget: Gadget that calls the method

Example	`gadget = parent.scrmGetSageCrmOwner(window.frameElement.id);`
	`parent.scrmSetGadgetProperty(gadget,`
	`document.forms[0].elements["propertyName"].value,`
	`document.forms[0].elements["propertyValue"].value);`
	`parent.scrmSaveGadgetProperties(gadget);`

scrmGetSageCRMOwner

Description	Finds the web site gadget that owns the current page
Parameters	iframeDomElementId : Id of the iframe element that is stored by the url gadget
Example	`gadget = parent.scrmGetSageCrmOwner(window.frameElement.id)`

Blocks

Creating a New Block

There are two ways to create a new block:

- In Administration | Customization | <Entity> | Blocks.
- In an ASP page linked to CRM through the customfile action in Administration | Customization | <Entity> | Tabs.

This section describes how to create a block within CRM. See also CRM Blocks Overview (page 4-1).

All new blocks you create in CRM are based on one of the CRM custom blocks. You can then customize the properties of the block.

Any lists or screens that you create in Administration | Customization | <Entity> | Lists and in Administration | Customization | <Entity> | Screens can be customized by creating a block from the list or screen and editing the properties of the block. These blocks are also available for use within ASP pages.

When a block is created, it is given a name and values for its properties. The blocks can then be accessed within an ASP page by using the CRM.GetBlock('blockname') method. The object returned by the GetBlock method has the specified properties already set.

To create a new block:

1. Select **Administration | Customization | Company | Blocks**.
2. Click on **the** New button. The New Block page is displayed.
3. Add the information for the new block. The fields are described in the table below.
4. Choose the **Save** button. The new block appears in the list of available blocks for the entity.

The table below describes each of the fields in the New Block page.

Field	**Description**
Block Name	Any text to describe the new block being created. This is the name that you use to reference the block from an ASP page.
Block Type	Defines the type of block. All the custom block types are available in a list.
Copy Existing Block	This is a list of blocks that already exist (including any new blocks you have created). Itis possible to make a new block that is a copy of an existing block and then further refine it.
Use Group	'EntryGroupBlock' or 'ListBlock', in which case it is a selection of the available screens and lists.One of these must be selected for the block to be associated with it.

Customizing a Block

To change the properties of an existing block or to specify the properties for a newly created block:

1. From **Administration | Customization |** <**Entity**> **| Blocks**.
2. Click on the hyperlink of the block name or select the **Customize** button. The Maintain Block definition page is displayed. The properties available vary depending on the type of block. As this is an EntryGroup block, the properties relate to block height and width as well as adding buttons.
3. Enter new properties or change the existing ones.
4. Choose the **Save** button.

Properties that can be specified on the Maintain Block definition page differ depending on the block type. Please refer to Introduction to the ASP Object Reference (page 8-2) for detailed information on specific properties that can be defined for each CRM block.

Displaying a Block

You display blocks that you create as follows:

- On a new tab using the customfile tab action to link to an ASP page that references the block.
- On a new tab using the runblock action to directly run the block.

Note: Only blocks of the following type can be run directly using runblock:

- Any EntryGroupBlock that is based on a screen of the current entity
- Content blocks

- Marquee blocks
- Message blocks
- Chart blocks

For more information on running blocks from tabs and ASP pages refer to Adding a Tab that Links to an ASP Page (page 4-25) and Tab Actions (page 4-27).

System Menus

Modifying System Menus

The Systems Menu functionality enables you to customize the following special types of tab groups:

- Administration menu.
- Main menu.
- Individual Administration work areas.
- Some User (Main Menu) work areas.

Note: In order to edit the tab groups for Primary and Secondary Entities, you access them via Administration | Customization, but you edit the tab groups for System Menus via Administration | Advanced Customization | System Menus.

For more information on System Menus please refer to the System Administrator Guide.

To access main menu buttons:

1. Select **Administration** | **Advanced Customization** | **System Menus**, and select **MainMenu**. The Customize Tabs For MainMenu page is displayed. You can now add, delete, and update the tabs in the Main Menu tab group.

Creating a Main Menu Button

Within System Menus, you can create new main menu buttons, and link them to custom pages.

To create a menu button that links to a custom page:

1. Select **Administration** | **Advanced Customization** | **System Menus**.
2. Click on the hypertext link of MainMenu. The Customize Tabs for MainMenu page is displayed.
3. From the Properties panel, enter a Caption name, for example **MyWeb**.
4. Select **Custom URL** (or **Custom File** if your search page is an ASP page that you have built) from the Action list.
5. Enter the file name in the Custom File field. If you want to link to a Web page as opposed to a custom page, type the URL in the URL Name field.
6. Select a GIF from the choices provided. If you have created your own graphic using a GIF editor, copy it to the following subdirectory of your install so that it is displayed as one of the choices: ..\Img\Menu.
7. Select **Yes** from the New Window list if you want the new page to be displayed in a new window.
8. Select the **Add** button and then choose the **Save** button. The **New Menu** button is displayed on the left-hand side of the screen. When you click on it, the custom page is displayed in a new window.

Creating an Admin Menu Button

You create new Administration menu (or home page) options in the same way as you add new main menu options.

To create a new Administration option that links to a custom page:

1. Select **Administration | Advanced Customization | System Menus**.
2. Click on any of the Admin hypertext links, for example **Admin** (the Administration main menu and home page) or **AdminUsers** (the Users home page). The Customize Tabs for MainMenu page is displayed.
3. From the **Properties** panel, enter a Caption name.
4. Select **Custom File** from the Action list.
5. Enter the file name in the Custom File field.
6. Select a GIF from the choices provided.
7. Select the **Add** button and then choose the **Save** button. The new menu Administration option is added.

Creating an External Link on the Main Menu

This example creates a new main menu option that opens your company's Internet home page in a new window.

To create the new menu option:

1. Select **Administration | System | System Behavior**.
2. Click on the **System Behavior** tab, and select the **Change** button. The System Behavior settings page is displayed.
3. In the Home page URL field type your company's home page URL, for example: **http://www.yourhomepage.com**.
4. Click the **Save** button.
5. When you log off and back on, the new menu option is displayed on the left hand side of the screen. Selecting it opens your company's home page in a new window.

CRM Objects Overview

The following is a brief description of the full range of CRM objects.

Dispatch Object

Controls all Web requests and responses, and finds the relevant UserSession and sets the session keys. When the user requests something to be done, for example, enter a new company or show communications, the request is sent to the application's Web module, which creates an object to process the request and output the relevant response. You do not have any access to the Dispatch object as it is internal to CRM.

CRM Object

Provides basic access to CRM objects and functionality. You use the methods of this object to create new objects, get existing objects, and execute objects.

CRMBase Object

Provides functionality that is only applicable in the CRM environment, such as the company currently being viewed. This object is often used to set up the current context information for the current view and to display tabs that apply to that view.

CRM TargetLists Object

The CRM TargetList Object is used to create Target Lists.

Note: In version 6.0 and above, target lists are referred to as "groups." However, to ensure that legacy code continues to work with new installations, the older term, "target lists" is maintained in the API terminology.

CRM TargetListFields Object

This is a container for a list of TargetListField objects.

CRM TargetListField Object

The fields that are displayed on the target list.

CRMSelfService Object

Contains methods and properties that enable self service users to access information relevant to them from the Self Service Web site.

MsgHandler Object

Used to customize the script deployed by E-mail Management, the MsgHandler object provides access to the Email Object.

Email Object

Used to customize the scripts deployed by E-mail Management, the Email Object provides access to the e-mail itself.

AddressList Object

Used to customize the scripts deployed by E-mail Management, the AddressList Object provides access to To, CC, and BCC lists of addresses.

Mail Address Object

Used to customize the scripts deployed by E-mail Management, the MailAddress Object provides access to individual addresses from the AddressList Object.

AttachmentList Object

Used to customize the scripts deployed by E-mail Management, the AttachmentList Object provides access to e-mail attachments.

Attachment Object

Used to customize the scripts deployed by E-mail Management, the Attachment Object provides access to individual e-mail attachments from the AttachmentList Object.

CRMRecord Object

Represents records in a table. This object is an enumerator that returns all the specified fields in a table. You use the CRM object's CreateRecord or FindRecord methods to get the record.

CRMQuery Object

Enters and executes SQL statements against a known CRM database. Used to perform more powerful queries than you can with the CRM Record object.

CRMBlock Object

The base of all CRM blocks. This block determines the actual implementation of each of the CRMBlock methods and properties.

CRMContainerBlock Object

Used to group other blocks on a screen. This block contains the standard CRM buttons Change, Save, Delete, and Continue. You can also configure Workflow buttons on screens where they are required to display. A linked search panel and related list is an example of a container block.

CRMEntryGroupBlock Object

Used to group entries to create a screen. You can generate many different kinds of entry groups, such as a Company Search Box, a Person Entry Box, and a Case Detail Box. This block also contains the standard CRM buttons.

CRMListBlock Object

Generates a custom list from columns in a CRM table, or a table connected to CRM through Administration | Advanced Customization | Tables And Databases.

CRMEntryBlock Object

Corresponds to a single field that is to be displayed or edited on-screen. There are many different entry types that you can create, such as text blocks, multi-select boxes, and currency input boxes. You typically add Entry blocks to EntryGroups or Containers. You can use JavaScript scripts on these blocks to perform tasks when they load, change, or are validated.

CRMGridColBlock Object

Related to the List block. Corresponds to a single column within the List block. You use the GridCol block to change properties on individual columns of a list.

CRMMarqueeBlock Object

Adds scrolling text to a page. The content of the text is maintained through Administration | Customization | Translations. You can use the properties of this block to control the direction, speed, and style of the scrolling text.

CRMFileBlock Object

Provides access to external files that are not part of CRM and enables the files to appear as if they are part of CRM.

CRMMessageBlock Object

Allows you to send messages in e-mail and SMS format. Include this block in ASP pages to display a simple e-mail form or to automate the message sending in response to a certain event.

CRMContentBlock Object

Simple block that takes a string of content (text) and displays it on the page. Used to write out a line of HTML to the browser.

CRMGraphicBlock Object

Enables the display of images through an ASP page. It is more powerful than standard static images, as variables can be used in their creation. These variables may represent live data from a database or incorporate details of the current user, such as their privileges or settings.

CRMChartGraphicBlock Object

Displays a variety of different charts. These charts may be generated from data retrieved via SQL or added through an ASP page. Inherits all the functionality of the Graphics block.

CRMOrgChartGraphicBlock Object

An implementation of the Graphics block being used for organizational charting. These charts may depend on data retrieved via SQL or added through ASP for their data. Inherits all the functionality of the Graphics block.

CRMPipeLineGraphicBlock Object

Creates cross-section diagrams representing data from an ASP page or stored in a table. Inherits all the functionality of the Graphics block.

Tabs

Creating a New Tab Group

When you link to a new table either in the CRM database or from an external database, you can create a new group of tabs to display the lists, screens, and charts for that table. You create new tab groups in Administration | Customization | <Entity> | Tabs. Once you create a new tab group you can display it by linking it to a main menu button.

Note: The following example assumes that a connection has already been made to an external table called InstallBase in a database called External. For information on making connections to external databases and tables, refer to Creating a New Database Connection (page 5-3).

To create a new tab group from a newly connected table:

1. Select **Administration | Customization**, and select the new table, **InstallBase**, from the Secondary Entities drop-down.
2. Select **Tabs**. The message displayed indicates that there are no tab groups for the entity.
3. Click on the **New** button and add a name for the new Tab Group, for example **InstallBase**. The new tab group is displayed in the list of tab groups.
4. To add tabs to the tab group, click on the **Customize** button or the **Tab Group Name** hypertext link.
5. Add the new tabs to the tab group. For more information on creating tabs, please refer to Adding a Tab that Links to an ASP Page (page 4-25).
6. Choose the **Update** button, and Save the changes you made.
7. To view the tab group, you need to create a new main menu button and set it to link to an ASP page, which calls the tab group.

For step-by-step details on creating a main menu option to display a tab group, see below. For more detailed information please refer to Creating a Main Menu Button (page 4-20).

When you click the new menu button the tabs in the new tab group are displayed.

Editing the Main Menu Tab Group

The Main Menu is a special type of tab group customizable via Administration | Advanced Customization | System Menus.

Note: The User (My CRM), Main Menu, and Team CRM tab groups, are all editable via Administration | Advanced Customization | System Menus.

To edit the Main Menu tab group:

1. Select **Advanced Customization** | **System Menus**. The list of System Menus is displayed.
2. Click on the hypertext link or the **Customize** button of the tab group (System Menu) you want to edit, for example MainMenu.
3. Add the new button to the tab group and Save the changes.

For more information on customizing tabs refer to the System Administrator Guide.

Adding a Tab that Links to an ASP Page

Add a New Tab that Links a Company to an External Invoice List

Note: This example assumes that a connection has already been made to an external table called Invoices. It also assumes that a list called Invoices_List has already been created using the external Invoices table.

To add a new Tab that links to a list:

1. Click on **Administration** | **Customization** | **Company** | **Tabs**.
2. Click on the hypertext link of the Company tab group.The Customize Tabs For Company page is displayed.
3. From the Properties panel, enter a caption for the new tab, for example **Invoices**.
4. Select **customfile** from the Action list.
5. In the Custom File field, enter the name of the file you want to link to.
6. Click the **Add** button, and choose the **Save** button.The ASP page displayed below, Invoices.asp, sets the tab to display the invoice list for the current company. It is calling a previously created Invoice List. For more information on creating lists refer to How to Create a List (page 4-3).

```
<!-- #include file ="sagecrm.js" -->
<%
// Get the current company ID
var ThisCompany=CRM.GetContextInfo("Company","Comp_CompanyID");
// Call the list block that you previously created for invoices.

var Invoices=CRM.GetBlock("Invoice_List");
Invoices.Title="3rd Party Invoice History";

// Display the list for invoices for the company.
CRM.AddContent(Invoices.Execute("CustomerID="+ThisCompany));
Response.Write(CRM.GetPage());
%>
```

7. You can view the new tab by finding a company.
8. When you click the **Invoices** tab a list of invoices for the current company is displayed.

Restricting Access to the Tab

When creating or editing a tab, you can enter a SQL statement in the SQL field, which specifies that the current record must match in order for the tab to be displayed.

For example, if you are adding an invoices tab to the company tab group, you can restrict tab access to users in the Direct Sales team.

Note: Avoid using script in the following format if you intend on restricting several tabs: user_userid=4.This is because the database gets queried separately for each restricted tab.

Instead, use the following script to query CRM:

- U:4,5 - to limit the tab to users whose ID is 4 or 5, for example.
- C:4,5 - to restrict the tab to certain teams, for example teams with a Channel ID of 4 or 5.
- T: - to restrict the tab to territories.

The following table describes the fields on the Properties panel.

Field	Description
Caption	Name of the tab, for example Invoices.
Action	Selecting a tab action from the list enables you to display various CRM screens. Please refer to Tab Actions (page 4-27) for an explanation ofavailable actions.
Custom File / Url Name /Block Name / Tab GroupName / System Action	The field name displayed here depends on what Action type is selected. If the Action selected is customfile, enter the file name in the Custom File field. If the Action selected is customurl, enter the URL in the URL Name field. If the Action selected is runblock, type the block name in the Block Name field. If the Action selected is runtabgroup, type the tab group name in the Tab Group Name field. If the Action selected is other, select a system action from the System Act. field.
SQL	You enter SQL in this field to restrict use of the tab to specified users or groups of users.Also refer to Restricting Access to the Tab.
Bitmap	If you are creating a menu button that links to a custom page, choose a GIF from the list. If you have created your own graphic using a GIF editor, copy itto the ...\Img subdirectory of your install so that it is displayed as one of the choices.
New Window	If you want the new screen to display in a new window, set this field to Yes.

Field	Description
Available Online Only	If you want the tab to be available online only for Solo installations, select this check box. Please refer to the Solo Guide for more information.

Tab Actions

Various actions are available from the Action list on the Properties panel of the Customize Tabs For page when creating a new tab. These are:

- customfile
- customurl
- runblock
- runtabgroup
- other (system actions)

All of the actions enable you to display different CRM screens. Screen areas that can be displayed vary from typical search screens to custom screens created using ASP pages. Please refer to the System Administrator Guide for details of the various system actions that are available if you do not have Extensibility Module (EM). If you have EM, additional actions are available. These actions are described below.

Customfile and Customurl

The customfile and customurl options enable you to display custom ASP pages as well as URLs from a tab. Please refer to Adding a Tab that Links to an ASP Page (page 4-25), refer toCreating an External Link on the Main Menu (page 4-21), and Modifying System Menus (page 4-20) for examples of how to use both Actions.

Runblock

You use this action to display blocks of the following types:

- Any Screen name of a screen that is based on the current entity.
- Any List name of a list that is based on the current entity.
- Any EntryGroupBlock that is based on a screen of the current entity.
- Content blocks.
- Marquee blocks.
- Message blocks.
- Chart blocks.

Runtabgroup

You use this to display tab groups.

Other

Selecting this option enables you to select a System Action. These are described in detail in the System Administrator Guide.

Chapter 5: Database Customization

In this chapter you will learn how to:

- Get an overview of database customization.
- Create a new table.
- Create a new database connection.
- Create a new table connection.
- Create a Tab to Display a List of Invoices.
- Display an Individual Invoice from a List.
- Add new data entry and maintenance screens.
- Get an overview of table and entity scripts and functions.
- Work with the Advanced Customization Wizard.

Introduction to Database Customization

As well as creating new tables in the CRM system, you can create new database and external table connections within Administration | Advanced Customization | Tables And Databases. A database connection is the registration of another database that the CRM server can connect to directly or indirectly. You choose which database type you want to make a connection to from a selection of predefined database types. Once a database connection is set up, a new table connection can be created by referencing this connection or any other existing database connection that is set up within CRM.

To access Administration | Advanced Customization | Tables And Databases, you need:

- Extensibility Module.

To connect to external databases and tables:

- The database with which the connection is made needs to have a field that can be used to uniquely identify the table within CRM, and there must be a matching field with this value on the related table within CRM.

Creating a New Table

To create a new table in the CRM database:

1. From **Administration | Advanced Customization | Tables And Databases**, select the **Create Table** button. The Table Details screen is displayed.
2. Type a table name in the Table name field. Note that the name should be one word only. Spaces are not permitted in table names.
3. Click on the **Table Caption** field. A caption, which is the same as the Table Name, is entered automatically in the field.
4. Enter an ID Field name in the following format: <columnprefix>_<tablename>id, for example, newt_newtableid.
5. Enter a column prefix in the Column Prefix field, for example newt. Note that you must not include an underscore, this will be added automatically by CRM when the table is created.

Note: you must enter an ID Field (step 4) and a column prefix (step 5) in order to use the table like a normal CRM table with screens, lists etc. It is therefore recommended that you always enter an ID Field and a column prefix.

6. Complete the rest of the fields using the table at the end of this section.
7. Select **Save**.

The new table is created on the CRM database. The following table columns are automatically created when the new table is created.

Col Names	Additional Info
Newt_NewTableId	To record the unique ID for records.
Newt_CreatedBy	To record the User who creates a new record.
Newt_CreatedDate	To record the date when new records are created.
Newt_UpdatedBy	To record the user who updates a record.
Newt_UpdatedDate	To record the date when a record is created.
Newt_Timestamp	To record the time when a record is created.
Newt_Deleted	To record the date when a record is deleted.

If you want to be able to link the new table records to Companies, People, or Users (and, to this end, you entered a name in the Company ID Field, Person ID Field, or User ID field on the Table Details screen) an additional ID field is created.

The following table describes the fields on the Table Details screen:

Field	Description
Table Name	The name of the table you are about to create.
Table Caption	The text for the table caption. This caption is displayed in lists. **Note**: The caption should not contain spaces and it is highly recommended that the caption and the table name are the same.
ID Field Name	The name of the unique ID field (column) in the table that uniquely identifies the table.
Column Prefix	The prefix for the columns in the table. This is usually three to four characters. A column prefix typically reflects the table name, for example "comp_" is the prefix for the company table. Note you should not include the underscore or an error message will be generated.
Description Field	Fill in the description field if you want to use this table as a lookup. If this field is filled in then when you are configuring a selection entry type, the table will be available in the ‘Existing Lookups’ drop down.

Field	Description
	This field should only be filled in for tables that are going to have small amounts of rarely changing data in them as the records are loaded into memory. Also it is the users responsibility to ensure that metadata is refreshed whenever changes are made to the table to ensure that the changes are reflected in the drop down list. If the table contains a large number of records (an approximate limit is 1,000), CRM may time out when loading.
Top Level Entity	Select Yes if you want the table to be a Primary entity.
Allow Web Service Access	Allows the table to be accessed by Web Services.
Read-only SData	Allows the table to be accessed by SData Provider. Please see SData Read-only (page 10-1) for more information.
Workflow Id Field	The name of the field that will be used for workflow identification.
Company Id Field	Enter a field name that will be used to hold identity values to link the new table to the Company entity.
Person Id Field	Enter a field name that will be used to hold identity values to link the new table to the Person entity.
User Id Field	Enter a field name that will be used to hold identity values to link the new table to the User entity.

Creating a New Database Connection

To create a new database connection to, for example, connect to a third party database:

1. Select **Administration | Advanced Customization | Tables And Databases**.
2. Select the **New Database Connection** button. The Database Details page is displayed.
3. Enter the database details and choose the **Save** button.

The new database becomes available in the list of available databases when you are making a new table connection.

The table below describes the fields on the Database Details page.

Field	Description
Database Driver	The database type, for example ODBC, Oracle or SQL.
Server Name	The name of the database server if it is an SQL server.
Database Name	The name of the database that you want to connect to.

Field	Description
Database Description	Description of the database that you want to connect to.
Username	The database username.
Database Password	The database password.

Creating a New Table Connection

To connect to an external table:

1. Select **Administration | Advanced Customization | Tables And Databases**.
2. Select the **New Table Connection** button. The Table Details page is displayed.
3. Enter the table details and choose the **Save** button. Once a new table connection has been created:
    - The table becomes available from Administration | Customization | Secondary Entity drop-down list. Note that you cannot add views to an external table.
    - All the columns in the table are displayed as fields in Administration | Customization | <external table> | Fields. The fields can be added to various screen areas, such as Lists and Screens, in Administration | Customization | <external table>.

The table below describes the fields on the Table Details page.

Field	Description
Table Name	The actual name of the table that you want to connect to.
Table Caption	The text to be used to describe the table. This caption is displayed in lists.
Database	The database on which the table exists.

Note: It is not possible to create a connection to two or more external tables if both tables contain a field with the same name.

Example: Creating a Tab to Display a List of Invoices

The following example describes how to connect CRM to a table called Invoices, which resides on a third-party database called External. The example shows how to display data from the table through CRM. The Invoices table contains a field called Customerid that can be used to uniquely identify companies within CRM. The company table in CRM has a corresponding field with this value.

The External database and the Invoices table are not supplied with the sample data in CRM. The External database and the Invoices table are also used in Example: Displaying an Individual Invoice from a List (page 5-6) and in Example: Adding New Data Entry and Maintenance Screens (page 5-7) . You can try out the examples with your own sample third-party database, or create your own using a tool such as SQL Enterprise Manager.

The steps involved in displaying information from the Invoices table in list format are as follows.

- Step 1: Make a connection to the External third-party database.
- Step 2: Make a connection to the Invoices table within the database.
- Step 3: Create the List object for Invoices.

- Step 4: Create a custom page to display the list with all invoices for the current company.
- Step 5: Add a new tab in CRM that links to the custom page.

Step 1: Make a Connection to the Third Party Database

To make a connection to the third-party database:

1. Open **Administration | Advanced Customization | Tables And Databases**.
2. Select the **New Database Connection** button, and complete the fields on the Database Details page.
3. Choose the **Save** button. The connection between CRM and the external database is made.

Step 2: Make a Connection to the Invoices Table in the External Database

To make a connection to the Invoices table within the external database:

1. Open **Administration | Advanced Customization | Tables And Databases**, and select the **New Table Connection** button. The Table Details page is displayed.
2. Complete the fields on the Table Details page.
3. Choose the **Save** button. The table becomes available from the Secondary Entities drop-down list in **Administration | Customization**.

Step 3: Create the List Object for Invoices

To create the list object for Invoices:

1. Within **Administration | Customization | Invoices | Lists**.
2. Select the **New** button.
3. Enter a name for the list and select the table that it is to be based on.
   Note: You need make a note of the name of the new list you create. This is the object name you reference from the ASP page.
4. Click the **Save** button.
5. Click on the **new list name** hypertext or the **Customize** button, and add the columns from the Invoices table that you want to be displayed in the list.
6. Choose the **Update** button, and Save the new list.

Step 4: Create the Custom Page

To create the custom page to display the Invoice List for the current company:

1. Create a new ASP page and save it using the file name, **invoices.asp**.
2. The main block of the ASP page should contain statements to perform the following tasks:
   - Retrieve the identifying value for the current Company and assign it to a variable: ThisCompany=CRM.GetContextInfo("Company","Comp_CompanyId");
   - Create a block for the Invoice List and assign it to a variable. Note you need to reference the list created in the previous step. Invoices=CRM.GetBlock("Invoice_ List");
   - Write the list to the screen by executing the List Block, telling it to only show records for this company: CRM.AddContent(Invoices.Execute("Customerid="+ThisCompany)); Response.Write(CRM.GetPage());

The Invoices.asp file is displayed below.

```
<!-- #Include file = "sagecrm.js" -->
<%/*
```

```
 This ASP displays a list of invoices with the current context company.
 Pre-Requisit: 3rd Party invoice table must have a CustomerID equal to Comp_CompanyID
 */%>
<%
 // Get the current company ID
 var ThisCompany=CRM.GetContextInfo("Company","Comp_CompanyID");

 // Call the list block
 var Invoices=CRM.GetBlock("Invoice_List");
 Invoices.Title="3rd Party Invoice History"

 // Display the list for invoices with a customerID of ThisCompany
 CRM.AddContent(Invoices.Execute("CustomerID="+ThisCompany));
 Response.Write(CRM.GetPage());
 %>
```

Step 5: Create the Invoices Tab

To create a new tab in the company context that links to the Invoices ASP page:

1. Select **Administration | Customization | Company | Tabs**.
2. Select the **Company tab group** hyperlink. The Customize Tabs for Company page is displayed.
3. Enter **Invoices** in the Caption field.
4. Select **customfile** from the Action list.
5. Enter **Invoices.asp** in the Custom File field.
6. Select the **Add** button to include the new tab in the Company tab group, and choose the **Save** button.
7. The results can be illustrated for this example by opening a company within CRM, and clicking on the Invoices tab. A list of invoices for the company is displayed.

Example: Displaying an Individual Invoice from a List

The CRM Custom Jump action functionality can be used to display an individual invoice from a list. You can create a link from an individual entry in a list to a summary screen of the entry.

The steps involved in linking from the invoice list screen to an invoice detail screen are:

- Step 1: Edit the List object for the Invoices table.
- Step 2: Create a Screen object for Invoices.
- Step 3: Create a Custom Page to display the Screen for an individual Invoice.

Step 1: Edit the List Object for the Invoices Table

To add custom jumps to the Invoices List:

1. Select **Administration | Customization | Invoices | Lists**.
2. Click on the hyperlink of the Invoice List. The Maintain List Definition page is displayed.
3. For each field you want linked, set the **Hyperlink To** field to Custom Jump.
4. Set Custom File to be the name of the ASP page that displays the Invoices screen, in this case **invdetail.asp**.
5. In the Custom Id Field, type the name of the field on the Invoices table that uniquely identifies each record, for example **InvoiceID**.
6. Choose the **Save** button.

Step 2: Create a Screen Object for Invoices

To create a screen for displaying individual invoice details:

1. Open **Administration | Customization | Invoices | Screens**.
2. Select the **New** button to add a new screen, and edit it to add in the fields from the Invoices table that are to be shown. For more information refer to How to Create a Screen (page 4-5).
3. Choose the **Save** button.

Step 3: Create a Custom Page

Use the script below to create a Custom Page to retrieve the screen you just created for individual invoices. The name of this page must be the same as the entry in the Custom File field in Step 1, in this case, invdetail.asp:

```
<!-- #Include file = "sagecrm.js" -->
<%now = new Date();%>
<HTML> <BODY>
<% //Return the value of the field used as the hyperlink to this page.
ThisInvoice=Request.QueryString("InvoiceID");
// Create the block object from the screen block created in CRM
InvoiceDetailBlock=CRM.GetBlock('Invoice_Detail');
//Find the record using the QueryString returned above
Record=CRM.FindRecord("Invoices","InvoiceID="+ThisInvoice);
//Pass the record object to the Screen block for execution later.
InvoiceDetailBlock.ArgObj=Record;
Container = CRM.GetBlock("container");
//Add a block to the container
Container.AddBlock(InvoiceDetailBlock);
//Set the buttons to be displayed in the block.
Container.DisplayButton(Button_Default)=false;
//Write container to screen.
CRM.AddContent(Container.Execute());
Response.Write(CRM.GetPage());
%>
```

The results can be illustrated for this example by clicking on a company and selecting the Invoices tab. The list of invoices for that company displays. The fields that are set up as jumps display in hypertext link format.

When you click on an invoice number, the screen for that individual invoice is shown, with a Continue button to return to the list.

Example: Adding New Data Entry and Maintenance Screens

This section describes how to add a new table within CRM to store customer specific information that can be edited or deleted as required. The example used is a new table called InstallBase to store information needed after an opportunity has been closed, and before engineers start the implementation. The new table stores extra information relating to each company.

This example includes the following steps:

- Step 1: Creating the Installed Base table.
- Step 2: Adding the Installed Base tab.
- Step 3: Adding the Installed Base list.
- Step 4: Adding the Installed Base view/edit/delete screen.
- Step 5: Creating the ASP Page(s) needed to display the screens.
  - Scenario A: Where there is one record in the new table for each company
  - Scenario B: Where there are multiple records in the new table for each company.

Note: This example relates to a table holding information for a company, but it works equally well for extra information for a person, case, lead, or opportunity.

Step 1: Create the Installed Base Table

1. From **Administration | Advanced Customization | Tables And Databases**, select **Create Table** to create a new database table in CRM called InstallBase.
2. Complete the fields, making sure you type **inst_installbaseid** as the ID Field Name, **inst_** as the Column Prefix, and **inst_companyId** as the Company ID Field.
3. Select **Save** to save the new table.

Step 2: Add the Installed Base Tab

Once the fields for the Installed Base table have been created in CRM, you need to create a tab to display the Installed Base details.

To do this:

1. Open **Administration | Customization | Company | Tabs**.
2. Add a tab to the Company tab group to display the Installed Base details.
3. Set the tab Action field to **customfile** and enter the name to be used for your ASP page in the Custom File field, in this case **installbase.asp**.
4. Choose **Update** and then the **Save** button.

Step 3: Add the Installed Base List

If there are multiple records in the new table for each customer, a List object that is used to view all the Installed Base table details is required.

To do this:

1. Open **Administration | Customization | InstallBase | Lists**.
2. Select **New** to add a new list called InstallBase, and edit it to add the fields that are to be shown. You can enable links between individual list items and their summary screens by creating a Custom Jump on a field of the list item.

To set up a jump on a field:

1. Select the **inst_companyid** field, and set the **Hyperlink To Field** to Custom Jump.
2. Set the **Custom File** field to the name of the ASP file that displays an individual item.
3. Set the Custom Id Field to the name of the field in the new table that uniquely identifies each record.

Step 4: Add the Installed Base View/Edit/Delete Screen

To create a Screen object that can be used to edit the Installed Base table details:

1. Open **Administration | Customization | InstallBase | Screens**.
2. Select **New** to add a new screen.
3. Customize the new screen to add the fields that you want to display.
   For more information on Screen Customization refer to the System Administrator Guide.

Step 5: Create the ASP Page(s) Needed to Display the Screens

Scenario A. Single record in the new table for each company

Create the Custom Page to view/edit installed base records. The name of this page should be the name specified in the Company tab group.

Start with a sample Entry Group ASP page. The main body of the ASP page should contain statements to perform the following tasks:

1. Retrieve the identifier value for the current company and assign it to a variable.

   ```
   CompanyId=CRM.GetContextInfo("Company","Comp_CompanyId");
   ```

2. Create an instance of the Installed Base screen and assign it to a variable.

   ```
   InstallBase=CRM.GetBlock("Install Base Details");
   ```

3. Create a record for the Installed Base record and tell it which record to show. Check if the record already exists and if not create a new one. Set the title on the block depending on whether the user is adding or editing.

   ```
   record=CRM.FindRecord("InstallBase","companyid="+CompanyId);
   if (record.eof)
   {
     record=CRM.CreateRecord("InstallBase");
     record("CompanyId")=CompanyId;
     InstallBase.Title="New Install Base Details";
   }
   else
   {
     InstallBase.Title="Edit Install Base Details";
   }
   ```

4. Display the screen using the record as the argument.

   ```
   CRM.AddContent(InstallBase.Execute(record));
   Response.Write(CRM.GetPage());
   ```

   This allows the user to add or edit Installed Base details for each customer by clicking on the Installed Base tab for a company. The first time the tab is selected a record is added for that company. After that, when the tab is selected, the record is shown for editing.

The installbase.asp script is displayed below.

```
<!-- #include file ="sagecrm.js" -->
<%
// Get the Id of the current company
CompanyId=CRM.GetContextInfo("Company","Comp_CompanyId");
// create the Screen Block
InstallBase=CRM.GetBlock("Install Base Details");
//Find the record in the table for this company
record=CRM.FindRecord("InstallBase","Inst_CompanyId="+CompanyId);
if (record.eof)
{
// if the record does not exist then create one and set the company id
        record=CRM.CreateRecord("InstallBase");
        record("Inst_CompanyId")=CompanyId;
        InstallBase.Title="New Install Base Details";
}
```

```
else
{
  InstallBase.Title="Edit Install Base Details";
}
if (CRM.Mode <= 1)
{
  CRM.Mode=1;
}
// Display the record
CRM.AddContent(InstallBase.Execute(record));
Response.Write(CRM.GetPage());
%>
```

Scenario B. Multiple records in the new table for each customer

This requires two custom pages, one to view a list of all the records and the other to jump to individual records.

Create a first Custom Page to view a list of all the records, the name of this page should be the name specified in the tab group for Company.

Start with a sample Entry Group ASP page(see example above).The main block of the ASP page should contain statements to perform the following tasks:

1. Retrieve the identifying value for the current Company and assign it to a variable:

   ```
   ThisCompany=CRM.GetContextInfo("Company","Comp_CompanyId");
   ```

2. Create a block for the Installed Base List and assign it to a variable. Note the name of the list is the name of the list created in the step above.

   ```
   InstallBase=CRM.GetBlock("installbaselist");
   ```

3. Write the list to the screen by executing the List Block, telling it to only show records for this company:

   ```
   CRM.AddContent(InstallBase.Execute("Inst_CompanyId="+ThisCompany));
   ```

4. Finally, add a New button to the screen to allow new records to be added. This calls another ASP page.

   ```
   CRM.AddContent(CRM.Button("New","new.gif",CRM.Url("InstallBaseEdit.asp")));
   Response.Write(CRM.GetPage());
   ```

Create a second Custom Page to jump to view/edit/delete individual records. The name of this page should be the name specified in the 'Custom Action File' section of the Installed Base list.

Start with a sample Entry Group ASP page. The main block of the ASP page should contain statements to perform the following tasks:

1. Retrieve the Id value of the record that is to be viewed from the Query string.

   ```
   ThisInstallBase=Request.QueryString("Inst_InstallBaseId");
   ```

2. Create a block for the Installed Base Screen and assign it to a variable.

   ```
   InstallBaseItem=CRM.GetBlock("InstallBaseDetailsBox");
   ```

3. Turn on the Delete button on the block to allow existing records to be deleted:

   ```
   DisplayButton(Button_Delete)=true;
   ```

4. Turn on the Continue button to allow users to return to the list when they are finished:

```
DisplayButton(Button_Continue)=true;
```

5. Check if this is an existing record or if a new record has to be created. Create the Record object if required. Note that if it is a new record, the exact Company Id must be set on it and it must go straight into Edit mode:

```
if (!Defined(ThisInstallBase))
{
       CompanyId=CRM.GetContextInfo("Company","Comp_CompanyId");
       InstallBaseRecord=CRM.CreateRecord("InstallBase");
       InstallBaseRecord("Inst_CompanyId")=CompanyId;
}
if (CRM.Mode <= Edit)
{
       CRM.Mode=Edit;
}
else
{
       InstallBaseRecord=CRM.FindRecord("InstallBase","Inst_
InstallBaseId="+ThisInstallBase);
}
```

6. Display the block, passing in the Record object. Note that the edit/delete/add functionality is handled by the block internally.
   To view the results, within CRM, go to a company and select the Installed Base tab. The list of Installed Base records for that company are shown. The fields that are set up as jumps are underlined. When you click on a jump, the screen for that individual installed base record is shown and can be edited or deleted. New records may be added from the list screen by clicking on the New button.

The Installbaselist.asp script is displayed below

```
<!-- #include file ="sagecrm.js" -->
<% // Get the value of the current Company ID
ThisCompany=CRM.GetContextInfo("Company","Comp_CompanyId");
// create the List Block
InstallBase=CRM.GetBlock("installbaselist");
// display the List
CRM.AddContent(InstallBase.Execute("Inst_CompanyId="+ThisCompany));
// add 'New' button to allow user to add new records for this company
CRM.AddContent(CRM.Button("New","new.gif",CRM.Url("InstallBaseEdit.asp")));
Response.Write(CRM.GetPage());
%>
```

Reporting on Data in the Installed Base Records

Follow these steps to create a report on the Installed Base table.

To report on the Installed Base table:

1. Create a view for the external table from **Administration | Customization | Installed Base | Views**.
2. Create a new report category for Installed Base from **Main Menu | Reports | New Report Category**.
3. Create a report in the new category, ensuring that you base it on the Installed Base view. For more information on creating reports, please refer to the System Administrator Guide.

The company ID field on the Installed Base table that just holds the Id (that is, an integer value) can be set to show the company name in the report by setting its Entry Type to be a Search Select, using the Company entity.

Note: You can create a view that links the Installed Base table with, for example, the Company table, so that more fields are available on the report.

Table and Entity Scripts and Functions

Introduction to Table and Entity Level Scripts

Table and Entity Level Scripts are an alternative method of creating SQL triggers that can be performed within the system. Table Level Scripts are executed when a record is inserted, updated, or deleted on a specified table in CRM. Entity Level Scripts are executed when an entity is updated, inserted, or deleted.

The Extensibility Module is required to create and customize Table and Entity Level Scripts in Administration | Customization | <Entity> | TableScripts.

The benefits of using Table and Entity Level Scripts instead of SQL triggers are:

- Improved database concurrency: The scripts are decoupled from the tables on which they are acting.
- Easier debugging: The ErrorStr statement can be included to display diagnostic and handled error messages. If an unhandled script error occurs, the system displays the script name, line number, and the error message.

You access the Table Level Scripting and the Entity Level Scripting functionality from within Administration | Customization | <Entity> | TableScripts.

Both types of script can be executed on system tables or on any tables that are connected to the system. Each script must have the four functions InsertRecord(), PostInsertRecord(), UpdateRecord(), and DeleteRecord() defined. These functions are automatically included in a template when you create a new script.

The following table describes each of the fields on the Table Script page. Refer to this table when creating any of the four script types: Table Level, Detached Table Level, Entity Level, or Entity Level with Rollback.

Field	Description
Name	A name to identify the script.
Windows User as Domain/ User (Optional)	If you want to run the script as a different user, enter the user name here.
User Password	Enter the above user's password here.
Script Type	Select the type of script you want to create. The options are Table Level, Detached Table Level, Entity Level, and Entity Level with Rollback.
View	This field is applicable when creating Entity Level Scripts only. Enter the view from which fields are available in the script. This must be relevant to the current entity. If no view is entered, the default Entity

Field	Description
	View for that entity is used: Company: vEntityCompany Person: vEntityPerson E-mail: vEntityPhoneEmailPerson, vEntityPhoneEmailCompany Address: vEntityAddress
Order	If there is more than one script for a table, this field can be used to specify the order of the scripts.
On error, only display the default error message	Select this check box to hide the error details from the user. You can enter an alternative error message to be displayed in the Default Error Message field. If you check this check box and do not enter a default error message, no errors are reported to the user.
On error, retry the script after delay	Select this check box to set a script to retry after a defined amount of time when an error occurs. The script is then scheduled to re-run in a few minutes and continues to be rescheduled until it runs successfully. This option is useful for situations where an external resource isn't available but is expected to become available in the future. Every time the script is rescheduled the amount of time until CRM retries is increased.
On disconnected clients, delay execution until the script can be run on the server	Select this check box if you are running a script that requires an external resource that might not be available on a disconnected client. The script is scheduled for execution on the server when the user next performs a synchronization. **Note**: Selecting this option transforms the script into a Detached Table Level script when it is run as a result of Solo client changes. This means that it is not run immediately on the client synch, but is subject to the normal detached table level script delay (usually just a few minutes). Also, it is not suitable where the script type is Entity Level With Rollback as it cannot handle the rollback feature when run on the server.
Default Error Message	Enter the text of the message that you wish to be displayed to the user when an error occurs. This is only displayed if any of the On Error.... check boxes are selected.
Logging Level	Select from the list the level of diagnostic information that is displayed for a script when you click the Show Log button after creating a new script. Off: Logging switched off. No entry in Log table. Low: Low level diagnostic information in Log table. Medium: Medium level diagnostic information in Log table. High: High level diagnostic information in Log table. An example of setting a logging level and using the Show Log button is described later in this section.
Table Level Script	This is where you enter the actual Table or Entity Level script.

Creating a Table Level Script

Table Level scripts are executed when a record is inserted, updated, or deleted on the specified table. Table Level scripts additionally enable you to reference CRM objects and allow you to access external applications, such as Microsoft Excel. For example, you could use a Table Level Script to write transaction logs of sensitive information to specific columns of a text file.

To create a Table Level script:

1. Open **Administration | Customization | <Entity>**, and select the **TableScripts** tab.
2. Complete the fields on the Table Script page using the table provided in the previous section, making sure you select Table Level from the Script Type list.
3. Enter the script in the Table Level Script field in the InsertRecord(), PostInsertRecord(), UpdateRecord(), or DeleteRecord() section of the field, depending on the type of script you want to run.
4. Choose the **Save** button.
5. The script is run when you create a record, after you create a record, when you update a record, or when you delete a record, depending on what type of script you entered in the Table Script field. Once the script is run, you can use the **Show Log** button to view a log of diagnostic information about the script. This is especially useful if you run a detached script, or if you selected the **On Disconnected Clients, Delay Execution Until the Script Can Be Run On The Server** check box and want to check that the script was successful. You can also use it to view more precise diagnostic information as to why a script didn't work.
6. To view the logging information, select the script you created from the list of existing scripts.
7. Select the **Show Log** button. The log is displayed in a new browser window.
8. Diagnostic information is displayed according to the logging level you set. This can be changed each time the script is run. You use the **Close Window** button to return to the main screen and the **Clear Log** button to clear the log of entries.

Detached Table Level Scripts

Detached Table Level Scripts are created in the same way as Table Level scripts, except you select Detached Table Level from the Script Type list when completing the Table Script details. Unlike Table Level scripts, Detached Table Level scripts are not run immediately but within a predefined amount of time. This enables the system to store a queue of scripts that need to run, and it means that a user does not need to wait for a script to complete.

If there are no other scripts queued at the server the script is run in a matter of minutes. This type of script is useful when the execution of the script is likely to be time consuming and you don't want the user to have to wait. For example, users do not see errors as they happen. The Log file must be viewed for any diagnostic errors.

Creating an Entity Level Script

Entity Level scripts and Entity Level with Rollback scripts are executed when an entity is inserted, updated, or deleted. Entity Level Scripts should be used:

- When the action is dependent on the whole entity-for example, you want to do something when a whole Company is inserted, not just when a record is added to the Company table.
- Entity Level with Rollback scripts can be used when you want to stop the action happening if there is a validation error or other error with the script.

The following entities can have Entity Level Scripts: Company, Person, E-mail, and Address. The scripts are invoked from only the following standard CRM screens when the final Save is clicked.

Screen	EntityScript	Function
New Company	Company	InsertRecord()
Change Company	Company	UpdateRecord()
Delete Company	Company	DeleteRecord()
New Person	Person	InsertRecord()
Change Person	Person	UpdateRecord()
Delete Person	Person	DeleteRecord()
Edit Phone/E-mail	Email	UpdateRecord()
New Address	Address	InsertRecord()
Edit Address	Address	UpdateRecord()
Delete Address	Address	DeleteRecord()

Entity Level Scripts are created in the same way as Table Level Scripts, except you select Entity Level or Entity Level with Rollback from the Script Type list when completing the Table Script details.

For example, with an InsertRecord Entity Level script attached to the company entity, each time you create a new company, the InsertRecord() function is executed after all the normal company updates have been done (this includes inserts into many tables, for example: address, address_link, phone, email, person, person_link) but before the final commit has been done.

If there is an error while executing the script and the script type is Entity Level with Rollback, all the changes are undone and the company is not inserted. The error is shown on the insert screen.

You enter entity level scripts in the same template as table level scripts.

Example: UpdateRecord in an Entity Level Script

The following example describes the use of the UpdateRecord() function in an Entity Level script. The script is designed to enter a record in an external table called Invoices, which resides on a third-party database called External, when the company type is changed from Prospect to Customer. The External database and the Invoices table are not supplied with the sample data in CRM.

To create the script:

1. Select **Administration | Customization | Company | TableScripts**.
2. Select the **New** button.
3. Complete the fields on the Table Script page, ensuring that you select **Entity Level** from the Script Type list.
4. Enter the following script in the Table Script field underneath the function UpdateRecord() entry.

   ```
   Please refer to Developer Help files for code sample
   ```

5. Choose the **Save** button.

This script is triggered when the company entity is updated. When you change a company type from Prospect to Customer a record is created in the external Invoices table.

Example: InsertRecord

The following example describes the use of the InsertRecord() function in a Table Level script. The script is designed to assign any new cases to the account manager of the selected company when the case is created. Workflow for Cases needs to be disabled before you try this example.

To create the script:

1. Select **Administration | Customization | Cases | TableScripts**.
2. Select the **New** button.
3. Complete the fields on the Table Script page, ensuring that you select **Table Level** from the Script Type list.
4. Enter the following script in the Table Script field underneath the function InsertRecord() entry.

   ```
   Please refer to Developer Help files for code sample
   ```

5. Choose the **Save** button.
6. Find a company and create a new case for it, but do not assign the case to a user.
7. The company account manager, in this example Brian Little, is automatically assigned to the case when you save the case.

Example: PostInsertRecord

You can include the record identifier generated in the InsertRecord() function in the PostInsertRecord() function of the same script. The following example describes the use of the PostInsertRecord() function in a Table Level script to send a communication suggesting a follow-up call to the Company account manager when a new case is created.

Note: PostInsertRecord is mainly used when you want to insert or update other records using the new identity ID that has been created for the record that was just inserted. You cannot update the current record in the PostInsertRecord method as it has already been saved at that point. You can read values from the Values collection, however any changes to the Values collection will not take effect.

To create the script:

1. Open the Table Level Script you created above.
2. Enter the following script in the Table Script field underneath the function PostInsertRecord() entry.

   ```
   Please refer to Developer Help files for code sample
   ```

3. Choose the **Save** button.
   To view the results, create a new case and note the user who it is assigned to. When you do this, log on at that user and open the Calendar/Tasks list for that user. The new communication is listed.

Example: UpdateRecord

The following example describes the use of the UpdateRecord() function in a Table Level Script. The script is designed to set all opportunities associated with a company to Stage Sale Closed, when the user changes the Status to Archive.

To create the script:

1. Select **Administration | Customization | Company | TableScripts**.
2. Select the **New** button.

3. Complete the fields on the Table Script page, ensuring that you select **Table Level** from the Script Type list.
4. Enter the following script in the Table Script field underneath the function UpdateRecord() entry.

   ```
   Please refer to Developer Help files for code sample
   ```

5. Choose the **Save** button.

   When you open a Company Summary, change the Status to Archive, and click the **Opportunities** tab, the stage of every opportunity related to the company is set to Sale Closed.

Example: DeleteRecord

The following example describes the use of the DeleteRecord() function in a Table Level Script. The script is designed to check whether there are any outstanding leads associated with a person, if the person record is deleted. If there are any outstanding leads, a message is displayed.

To create the script:

1. Select **Administration | Customization | Person | TableScripts**.
2. Select the **New** button.
3. Complete the fields on the Table Script page, ensuring that you select **Table Level** from the Script Type list.
4. Enter the following script in the Table Script field underneath the function DeleteRecord() entry.

   ```
   Please refer to Developer Help files for code sample
   ```

5. Choose the **Save** button.

   To view the results, find a person who has an associated lead and delete the person record. The warning message is displayed.

Advanced Customization Wizard

Creating a New Main Entity

To create a new main entity:

1. Select **Administration | Customization | Component Manager**. The Components tab is displayed. If there are any components available to install, they are displayed in the Available Components list. If not, a message is displayed to inform you that there are no components in the INF directory—this is the directory on the server where components are stored. In addition, a panel called Add Component is displayed.
2. Select the **Browse** button and navigate to the location where you saved the Advanced Customization Wizard ZIP file.
3. Select the file and select **Open** from the Windows dialog box. The path to the file is displayed on the Add Component panel.
4. Select the **Upload New Component** button. When you do this, the component becomes available on the Available Components list. In the background, the Advanced Customization Wizard ZIP file is unzipped and uploaded to the INF directory on the server. If an INF directory doesn't already exist, it is created at this point.

5. Select the component and click on the **Install Component** button. The Parameter Info screen is displayed, with a number of fields that you complete or select in order to name the new entity and create associations between it and the system.
6. Complete the fields on the **Parameter Info** screen. See Advanced Customization Wizard Parameters (page 5-18) for a detailed description of each field.
7. Select the **Install Component** button. Progress messages are displayed, and the Continue button becomes available when the new entity is fully created. The following are created as standard:
   - Name and status fields.
   - Search, entry, summary and top content screens.
   - A grid for the new entity.
   - A tab group with a tab that runs a custom summary screen.

   Other screen elements created depend on the parameters you specified in Step 6. Refer to the Advanced Customization Wizard Parameters (page 5-18) for more details.
8. Select the **Continue** button. You are returned to the Components tab. The Advanced Customization Wizard component is listed in the Installed Components panel on this tab.

Note: The Advanced Customization Wizard component also remains listed on the Available Components list at the top of the screen. This enables you to create as many new entities as you require by selecting the component and following the installation steps above.

You can now view the new entity you created, and you can customize it. Please refer to Customizing a New Main Entity (page 5-25) for more information.

Advanced Customization Wizard Parameters

The following Table describes the fields on the Parameter Info screen.

Field	**Description**
Entity Name	Enter the name of the new entity. This is the name of the database table for the entity, as well as the caption for the entity that will be used throughout CRM to identify it. The name must be less than 27 characters in length and must not be the same name as an existing table in the CRM database.
Entity Column Prefix	Enter the letter string to be used as a prefix to the names of the columns in the new entity's database table. The column prefix must be four characters in length. There is no need to include an underscore. The prefix must follow the particular server's rules for identifiers. For example, when working on an SQL Server, the prefix must follow SQL Server's identifier rules.
Tag With Component Name	This allows you to script out and further customize new entities that you create. Once you type an entity name in the Entity Name field, the value in the Tag With Component Name field is set to <EntityName>_ Component and a new component (with the same name) is added to your list of Existing Components.

Field	Description
	Having created a new entity, you can select the new component from the Existing Components list and: Click on the Preview Script button to view all of the changes involved in creating the new entity. Set <EntityName>_Component as the currently recording component and then make further customizations to the entity you created. Script out the entire customization (entity creation and further customizations). Refer to Scripting Customizations (page 6-4) for more information on scripting out components.
Add To My CRM	Select this check box to create a custom list and a custom tab for the My CRM work area. This enables you to view a list of all the new entity records that are associated with a particular user.
Add To Find	Select this check box to create a custom search entry screen and a corresponding search results list. These allow the new entity's records to be searched for using the Find functionality in CRM.
Add To Team CRM	Select this check box to create a custom list, an asp page that displays the list, and a custom tab for the Team CRM work area. This enables you to view a list of all new entity records associated with a particular Team.
Has Companies	Select this check box to create a company tab and to add a corresponding custom company list to the tab group. This enables you to view a list of associated companies for all new entity records. It also enables you to link existing companies to the new entity via a Link button. If you want to set up deduplication for companies in this scenario, refer to Enabling Company and Person Deduplication (page 5-21).
Has Accounts	Displayed only if Integration is set up. Select this check box to create an account tab and to add a corresponding custom account list to the tab group. This enables you to view a list of associated accounts for all new entity records. It also enables you to link existing accounts to the new entity via a Link button.

Field	Description
Has People	Select this check box to create a people tab and to add a corresponding custom people list to the tab group. This enables you to view a list of all associated people for all new entity records. It also enables you to link existing people to the new entity via a Link button. If you want to set up deduplication for people in this scenario, refer to Enabling Company and Person Deduplication (page 5-21).
Has Opportunities	Select this check box to create an opportunities tab and to add a corresponding custom opportunities list to the tab group. This allows you to see all the associated opportunities for all new entity records.
Has Leads	Select this check box to create a lead tab and to add a corresponding custom leads list to the tab group. This allows you to view all the associated leads for all new entity records.
Has Cases	Select this check box to create a cases tab and to add a corresponding custom cases list to the tab group. This allows you to view associated cases for all new entity records.
Has Communications	Select this check box to create a communications tab and to add a corresponding custom communications list to the tab group. This allows you to view associated communications for all entity records.
Workflow	Select this check box to create a workflow for the custom entity. Selecting the check box also enables default workflow rules for the new entity.
Deduplication	Selecting this check box creates a deduplication screen for the new entity. Deduplication rules can then be set up in CRM. If the new entity "Has People" or "Has Companies" and you want to set up deduplication screens for them, you need to refer to Enabling Company and Person Deduplication (page 5-21).
For Dot Net	Selecting this check box creates an entity for which you can write a .Net module instead of using ASP pages. The entity is created with metadata in the usual way but as ASP pages are not created, you need to use the .NET DLL to customize the entity.
Has Library	Selecting this check box creates a library tab and adds a corresponding custom library list to the tab group. This enables you to view all associated library entries for all new entity records.
Has Workflow Progress	Selecting this check box creates a progress table for the custom entity table. It also provides the ability to add progress notes for custom entity records.
Owned By Companies	Selecting this check box adds a custom tab to the Company tab group that displays a list of all the associated new entity records for a particular company.

Field	Description
Owned By Accounts	Displayed only if Integration is set up. Selecting this check box adds a custom tab to the Account tab group that displays a list of all the associated new entity records for a particular account.
Owned By Orders	Displayed only if Integration is set up. Selecting this check box adds a custom tab to the Orders tab group that displays a list of all the associated new entity records for a particular order.
Owned By Quotes	Displayed only if Integration is set up. Selecting this check box adds a custom tab to the Quotes tab group that displays a list of all the associated new entity records for a particular quote.
Owned By People	Selecting this check box adds a custom tab to the People tab group that displays a list of all the associated new entity records for a person.
Owned By Opportunities	Selecting this check box adds a custom tab to the Opportunities tab group that displays a list of all the associated new entity records for an opportunity.
Owned By Leads	Selecting this tab adds a custom tab to the Leads tab group that displays a list of all the associated new entity records for a lead.
Owned By Cases	Selecting this check box adds a custom tab to the Cases tab group that displays a list of associated new entity records for a case.
Allow Web Service Access	Selecting this check box ensures that the new entity is enabled for web services. Please refer to Web Services (page 9-1) for more information.
Read-only SData	Selecting this check box ensures that the new entity is enabled for SData. Please refer to SData Read-only (page 10-1) for more information.

Enabling Company and Person Deduplication

If the entity you created "has companies" or "has people", and you want a Deduplication page to be displayed when you create a Company or Person from within the context of the new entity, you need to edit the <entityName>Company.asp page or the <EntityName>Person.asp page.

To enable deduplication if the entity "has" Companies, open the <entityName>Company.asp and change the action from 140 to 1200.

Alternatively, to enable deduplication if the entity "has" People, open the <EntityName>Person.asp and change the action from 141 to 1201.

For example, let's say you created a new company that "has" People. You want the Person deduplication page to be displayed when you create a new Person from within the Project context.

To do enable deduplication:

1. Change the script in the <ENTITYNAME>Person.asp from

```
CRM.URL(141)+"&Key-1="+iKey_
CustomEntity+"&PrevCustomURL="+List.prevURL+"&E=Accounts", 'Person', 'insert'));
```

to

```
CRM.URL(1201)+"&Key-1="+iKey_
CustomEntity+"&PrevCustomURL="+List.prevURL+"&E=Accounts",  'Person', 'insert'));
```

1. Save the file.
   When you select the New action button to create a new Person within a Project record, the Person Deduplication page is displayed.

Custom Files and Metadata

Certain specific custom files and metadata changes are created by the Advanced Customization Wizard. These are detailed below.

Custom Files

A number of custom files are generated for the new entity depending on the selections you made on the Parameter info screen when you were creating it. The custom pages are stored automatically in:

…Program Files\Sage\CRM\<installname>\WWWRoot\CustomPages\<Entity Name>

The table below lists all possible custom files that can be created and gives a description of each.

File Name	Description
<Company><EntityName>.asp	This page displays the list of all new entity records owned by a particular entity e.g. company (if Owned By Companies was selected on the Parameter Info screen). A similar file can be generated for: *People* *Leads* *Opportunities* *Cases* *Accounts* *Quotes* *Orders*
<EntityName><Person>.asp	This page lists all of the new entity's people (if Has People was checked). Depending on your selections, a similar file can be created for: *Communications* *Case* *Lead* *Opportunity* *Company* *Library* *Accounts*
<EntityName>Channel.asp	This page displays the list of all of the new entity records associated with a Team. The list is displayed on the Team CRM area (if Add To Team CRM was selected on the Parameter Info screen.
<EntityName>Summary.asp	The summary page for new entity records.

File Name	Description
<EntityName>Find.asp	This page allows you to search for the new entity records (if Add To Find was selected on the Parameter Info screen).
<EntityName>ToDo.asp	This page displays the list of all the new entity records associated with a user. The list is displayed on the My CRM area (if Add To My CRM was selected on the Parameter Info Screen).
<EntityName>Dedupe.asp	This page displays the custom dedupe screen if Deduplication was selected on the Parameter Info screen. If not, it redirects you to the <EntityName>New.asp
<EntityName>Conflict.asp	This page lists all of the conflicts that your dedupe entrygroup found.
<EntityName>Library.asp	Enables library items to be linked to the new entity.
<EntityName><Company>Link.asp	Enables you to create links between the entity records and other companies or people.
<EntityName>New.asp	This page allows you to create new entity records.
<EntityName>WF.asp	This page allows you to create a Workflow for the new entity.
<EntityName>ProgressList.asp	Enables you to progress the new entity record.

Metadata

The following metadata may be created in CRM, depending on the selections you made on the Parameter Info screen.
You can view these in Enterprise Manager, for example, in the Custom_Tables table, and you can view them in CRM via Administration | Customization. The table below lists the metadata that is created.

Metadata	Description
<entityname>SearchBox	The entry screen used for search selects and finds on new entities.
<entityname>NewEntry	The entry screen used to create new entity records.
<entityname>BoxDedupe	The deduplication screen for the custom entity (if Deduplication is selected on the Parameter Info screen).
<entityname>TopContent	The context area for the new entity records.

Metadata	Description
<entityname>SummaryScreen	The summary screen for new entity records
<entityname>SearchBox	The search screen for finding new entity records.
<entityname>Grid	The grid used for search selects and finds on new entity records.
<entityname>UsersGrid	The grid used to list new entity records for a particular User.
<entityname>ChannelGrid	The grid used to list new entity records for a particular Team.
MainEntity<entityname>Grid	The grid used to list new entity records for a particular main entity (if Owned By <MainEntity>, is selected on the Parameter Info screen).
<entityname>	The tab group for the new entity.

Making Custom Entities Available for Reassignment

Administrators and info managers can reassign entity records associated with one user to another user or team of users.

To access the screen for reassigning user records:

1. Select **Administration | Users | Users**.
2. Use the Find screen to locate a user and click on that user’s name. The User Details page is displayed. There are three buttons available for disabling users and reassigning their records: **Reassign**; **Reassign and Disable**; and **Disable**.
3. Click on the **Reassign** button. The Reassign User Records page is displayed. You can see that it is also possible to reassign records from the new custom entity, Project. In addition, you can filter which records to reassign by selecting them according to their status. In this example, only projects that have their status flagged to "Not Started" will be reassigned.

After you create a custom entity, it will appear automatically in the Reassign User Records page. In addition, the field used as a filter, Status ("proj_status"), is added by default to the new table. The only task facing the administrator is to add the necessary options to the Status selection field. In this example, they are "Not Started," "In Progress," and "Complete".

To specify options for the custom entity’s Status field:

1. Select **Administration | Customization | <EntityName>**. In this case, click on the name of the Project entity.
2. Select the **Fields** tab if it is not already open.
3. In the row for Status, click on the hyperlink indicating the field’s type, **Selection**. The Maintain Lookup Selection for Status page is displayed.
4. Enter the appropriate values in the Add Translation and Code fields and click on the Add button. In this instance, you would specify the third option to appear in the down-down list (and in the list appearing on Reassign record pages) by typing **Complete** into the Add Translation field and entering **Status 3** as a code. For more information, refer to the chapter "Field Customization" in the *System Administrator Guide*.

Customizing a New Main Entity

This section explains how to:

- Customize a new entity that has been created using the Advanced Customization Wizard.
- Change the new entity logo.
- Add a report view to an entity.

Customizing a New Entity

You can customize screens, fields, lists, and tabs created by the Advanced Customization Wizard in the same way as you customize screens, fields, and lists for a default CRM entity, such as a Company, or a Person.

To customize a new entity:

1. Click on Administration | Customization.
2. Select the new entity, for example Project, from the Customization home page.

   Note: Depending on the options you chose on the Parameter Info screen an entity progress table may be available too, for example ProjectProgress. This can be customized in the same way as a typical Progress table.

   The Customize Fields tab for Project is displayed. The screen displays all the standard CRM fields that have automatically been added to the entity database table.

   You can customize the new entity's Fields, Screens, Lists, Tabs, and Views in the normal way. Please refer to the *System Administrator Guide* for more information.

Changing the Entity Logos

Two images (a small one and a large one) are automatically used as the logo for all new entities you create. They are named according to the entity name you provide—<EntityName>.gif and small_<EntityName.gif>— when creating the entity, and they are copied to the following location:

…Program Files\Sage\CRM\<installname>\WWWRoot\Img\Icons

You can change the logos by overwriting the default ones if you wish.

Adding a Report View to an Entity

You can create a view to specify tables and columns from which a report can be drawn. A view can be created for any entity.

For example, you might want to create a new report view for the Project entity. This view will show cases associated with projects, who the cases were logged by, case status, case priority, and case description.

To create a new report view:

1. Select **Administration** | **Customization** | **Project**. The entity context that you select should correspond to the main database table you reference in the view. The **Fields** tab provides you with the names of the columns that can be added from the table of the entity you have selected.
   In addition to the fields shown in the interface, each table has a hidden unique identifier that is used for the SQL joins.

Table	Unique ID
Project	proj_projectid
Cases	case_caseid

2. Select the **Views** tab.
3. Click on the **New** button. The New Views page is displayed.
4. Type in the View Name

   Note: Name the view starting with a single "v", with a single word, and with no spacing, for example, vProjectCaseView. The View Script field is automatically populated with the start of the script.

5. Select the **Reports View** check box. This makes the view available when creating a new report.
6. Type a short description of the view in the Description field.
7. Type a translation for the view in the Translation field. This is what the user will see on the screen when the view is selected from the drop-down list.
8. Type in the SQL for the new view.

```
CREATE VIEW vProjectCaseView
AS
SELECT proj_name, case_caseid,  case_openedby, case_priority, case_status, case_
description
FROM PROJECT
INNER JOIN cases
ON proj_projectid = case_projectid
SQL script for the new report view
```

   The columns in the SELECT statement will be the columns available in the report.

9. Select the **Save** button. The new view will now be available in the Source View drop-down list on the Report Options, Step 1 of 2 page when you are creating a new report.

Advanced Customization Wizard Example

Creating a New Entity called Project

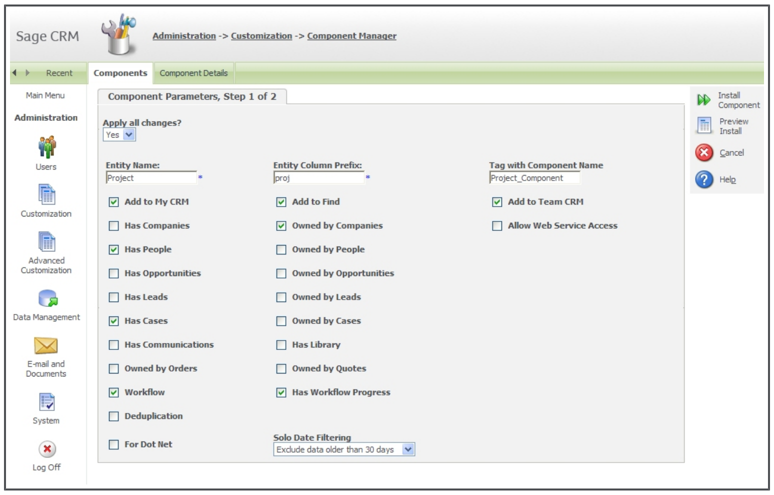
This example shows how to create a new entity with the following features:

- The entity name is Project.
- The column prefix for the new entity is "proj".
- The new Project entity is owned by the Company entity.
- It can have People and Cases associated with it.
- It is available as a tab in the My CRM and Team CRM work areas.
- Individual Projects can be found by selecting the Find menu button.
- Workflow and Workflow Progress screens are available for Projects.

Before starting, ensure that the Advanced Customization Wizard is installed in Component Manager as described in Creating a New Main Entity (page 5-17).

To create the new entity:

1. Select the **Components** tab.
2. Highlight the **Main Entity Wizard** component from the Available Components list, and select the **Install Component** button.
3. When the Parameter Info screen is displayed, complete the fields as follows:



Component Parameters, Step 1 of 2

Apply all changes? Yes

Entity Name: Project *	Entity Column Prefix: proj *	Tag with Component Name: Project_Component
✓ Add to My CRM	✓ Add to Find	✓ Add to Team CRM
☐ Has Companies	✓ Owned by Companies	☐ Allow Web Service Access
✓ Has People	☐ Owned by People	
☐ Has Opportunities	☐ Owned by Opportunities	
☐ Has Leads	☐ Owned by Leads	
✓ Has Cases	☐ Owned by Cases	
☐ Has Communications	☐ Has Library	
☐ Owned by Orders	☐ Owned by Quotes	
✓ Workflow	✓ Has Workflow Progress	
☐ Deduplication		
☐ For Dot Net	Solo Date Filtering: Exclude data older than 30 days	

Parameter Info screen

In the Entity Name field, you add the name of the custom entity, in this case **Project**. The **Tag With Component Name** field automatically displays the entity name with the "_Component" suffix appended. In this example, the field displays "Project_Component." In the Entity Column, enter the name of the prefix that will be used to identify fields in CRM. For example, the Project entity will probably feature information about a project manager, the name of whom could be stored in a field called "proj_manager." Note: as previously explained, the system adds the required underscore ("_") character by default, so the admin should enter only the prefix name. In this instance, type **proj**.

4. Select the **Install Component** button.
5. When the component is installed and the new entity is created, you can see how your selections on the Parameter Info screen translate to CRM by taking a look at the following areas in CRM and the CRM database.
   - The entity name is Project—a new database table called Project is created.
   - The new entity (Project) becomes available from the Customization home page.
   - The entity name is Project—a new database table called Project is created.
   - The column prefix for the new entity is "proj". The standard fields created for the new entity are prefixed with "proj_".
   - The new Project entity is owned by the Company entity. Each new Project you create must be associated with a company.
   - A Project record can have People and Cases associated with it.
   - The new entity is available as a tab on the My CRM and Team CRM work areas.
   - Individual Project records can be searched for using the Find main menu option.
   - You can create a Saved Search for projects.
   - Workflow is available for Projects.
   - A Project Progress secondary entity is created.

Custom files and meta data created are discussed in Custom Files and Metadata (page 5-22).

Chapter 6: Component Manager

In this chapter you will learn how to:

- Get an overview of the component manager.
- Record customizations.
- Change the current component.
- Script customizations.
- Script multi-stage customizations.
- Script workflows.
- Save a component.
- Install a component.
- Understand advanced component options.
- Modify component manager scripts.

Introduction to Component Manager

Component Manager allows customizations made on one CRM system to be saved and transferred to another CRM system. It enables CRM developers to package and reuse implementation-specific customizations in future implementations.

Recording

Recording Customizations (page 6-2)
Advanced Component Options (page 6-9)

Scripting

"Scripting" is the term used to describe the process of generating a set of script files from a particular component. See Scripting Customizations (page 6-4) for full details. Advanced Component Options offers an alternative way to script components (see Advanced Component Options (page 6-9)).

Installing

When a scripted component is installed to a second CRM system all of the customizations recorded in the script are copied to the second system. See Installing a Component (page 6-7) for more information.

What Types of Customizations can be Recorded?

All changes made in the Administration | Customization section of CRM can be recorded and scripted using the Component Manager. Specifically, you can use the Component Manager to record and script:

- Field Customizations
- Field Security where the update applies to "Everyone"
- Screen Customizations-including Field Level Scripting and Custom Content
- View Customizations
- List Customizations
- Tab Customizations - including System Menus and Menu Buttons

- Block Customizations - including Dashboard blocks
- TableScript Customizations
- Translations-including inline translation mode, field customization method and translations list method
- Reports (creation of new reports and modification of existing ones)
- Most Workflow Customizations. See Scripting Workflows (page 6-6).
- Button Groups
- Table And Database connections
- Interactive Dashboards. **Note**: Only "unassigned" templates should be shared between systems, as the users and teams will differ in different systems.

Component Manager only records customizations. It does not record configuration information, or data, and it does not record some other specific customizations. For example the following are **not** recorded:

- Field Security other than where the update applies to "Everyone"
- Field deletions
- Products
- Currencies
- Configuration settings
- User data
- Workflow Escalation Rules
- Territory changes
- Related entities
- Security profiles

Customization script files, such as ASP pages, will be included automatically if they are directly referred to (for example by a newly created tab). However when an ASP page is updated, or when a file that is indirectly referred-to is added (for example an "include file" in an ASP page), then these files must be manually copied to the component folder. See Scripting Customizations (page 6-4) for more information.

Recording Customizations

Customizations made to your CRM system can be recorded as a Component and then transferred to another CRM System. When Component Manager is started, customizations are recorded to the Current Component (see below). When Component Manager is stopped customizations are not recorded.

The Component Details Screen

The Component Details screen contains the options for selecting and recording components. To open the Component Details screen:

1. Open **Administration | Customization | Component Manager**.
2. Select the **Component Details** tab.

The Component Details tab has two panels:

- **Current Component.** This panel displays details of the current component. All the customization changes you make are saved to the current component unless you create a new component and set it as the current component. The default component name for the current component is "Changed".

- **Existing Components.** This panel lists all existing components on the server. These include components that have been installed, components available for installation, as well as components you created.

Starting Component Manager

1. On the Component Details tab, click the **New** button.

   If you are working on an implementation where you are likely to create more than one component, you should always make sure that you script the current component before you start creating a new one. Otherwise, you risk overwriting customization changes you recorded previously. Please refer to the Scripting Customizations (page 6-4) page.

2. Type the component name in the **Component Name** field, and type a description in the **Description** text box. The "Set To Be Current Component" check box is select by default. This means that the newly created component will start recording immediately.
3. Click the **Save** button. The component details are displayed in the Currently Recording Component panel, as the component is now the current component. In addition, the component is displayed in the list of Existing Components.

Once the Current Component is set:

- The Component Manager automatically records changes you make and it associates the changes with the current component. Please refer Introduction to Component Manager (page 6-1) for more details.
- On all customization screens (for example when adding a new field or modifying a list) you will see a blue banner with the text:

  This customization will be labeled with Component Name "[Name]"

  This allows you to verify that the changes you are making are being recorded to the correct component.
- The Component continues to record changes against the current component until you do one of the following:
  - Stop Component Manager.
  - Set a different component to be the current component.

  You can make customization changes over a number of days, and you can log in and out of CRM during this period. You can also perform multi-stage customization. Please refer to the Scripting Multi-Stage Customizations (page 6-5) page.

Stopping Component Manager

The Existing Components panel is the only panel displayed on the Component Details screen. It remains active until you start recording again or decide to script out an existing component.

Adding Customizations to an Existing Component

You can add customizations to any of the components in the Existing Components List. To do this:

- If currently recording a different component, then select another component in the Existing Components List and click "Set to be the Current Component". If the current component has never been scripted, a dialog box is displayed to remind you.
- If you are not recording any components, then select a component from the Existing Components List and click "Start Component Manager"
- You can also use Advanced Component Options to add customizations to an existing component. See Advanced Component Options (page 6-9).

Changing the Current Component

If a number of components are listed in the Existing Components List, you can make any of them the Current Component. Once you do this, all customizations you make are recorded against the new current component.

To change the current component:

1. From the **Existing Components** list, highlight the component you want to set as the current component.
2. Click the **Set To Be Current Component** button. If the current component has not yet been scripted, a dialog box is displayed to remind you.
3. You need to select the **Cancel** button and script the existing component at this stage before you create a new one, or select the **OK** button to continue without scripting the existing component.
4. When the component is set as the current component, its details are displayed on the Currently Recording Component panel, and a blue banner with the text *This customization will be labeled with Component Name '[Name]'* is displayed on all customization pages.

Scripting Customizations

"Scripting" is the term used to describe the process of generating a set of script files from a particular component. These files can then be installed to a second CRM system.

Before final scripting of a component, you can preview the script that will be generated.

Previewing Changes

To view the customization script in your browser window:

1. Select the **Component Details** tab and select the component you want to script from the Existing Components list.
2. Click on the **Preview Script** button. When you do this, the customization script is displayed in the Script Preview Of display area.
3. Select the **Continue** button to close the Script Preview Of display area.

Scripting Changes

To script recorded changes:

1. From the **Component Details** tab, ensure that component you want to script is selected from the Existing Components list.
2. Select the **Script Component** button.
3. You may accept the default filename, however you may wish to choose a new one if you have already scripted the current component. See Scripting Multi-Stage Customizations (page 6-5).

4. Add a brief description of the changes you've made in the **Description** field. The Description field is not a required field. Information you type here will be saved in the ECF file (see below) when the component is scripted. This information may be useful at a later date. For example, you can write a brief description that will remind you of the customization changes you made in a particular implementation before you reproduce the changes in a new installation.
5. Ensure that **Script As XML** is not checked. If you check this box then an XML file will be produced instead of an ES file. This XML file cannot be used by Component Manager on the target system. Script As XML is only used in specific scenarios involving integration to other applications.
6. Select the Script Component button. The component is scripted and the generated file names and locations are displayed.

Note: The Component Manager automatically copies files from the system's Custom Pages folder only if they are included in a tab group with the customfile action. If your customization files include other files such as ASP, .NET Dlls, TXT, JS, or INC files, then copy these extra files into the component's Custom Pages directory when the component is finished scripting. The Custom Pages folder will be at *<installpath>\<installname>\inf\<component name>\CustomPages*

Note:The Component Manager can be used to script Interactive Dashboard changes. This can only be done if the data sources, users and channels are identical in both Sage CRM systems. For example, if you script out a dashboard gadget based on a Saved Search, the saved search must exist on the new system for the gadget to work on the dashboard.

When you script a component, all the files created are written to the *<CRM install path>\<installName>\inf* directory. In a typical implementation, the INF directory contains the ECF file (for example MyComponent.ECF) and the component directory. The component directory contains the ES script file (for example MyComponent.ES) and any another files you scripted, for example ASP pages.

- The ECF file lists the component name and details.
- All of the customization changes you record are scripted out in JavaScript to the ES script file.

See Saving a Component (page 6-7).

Scripting Multi-Stage Customizations

When you are recording customization changes, you may want to script all the changes you have made up to a certain point in order to replicate just those changes in a new implementation. You may then want to continue making changes and replicate the new changes as well as the previous ones in another implementation. Component Manager allows you to do this.

For alternative solutions, see Advanced Component Options (page 6-9).

To script a multi-stage customization:

1. Create a new component called **123 Comms**, for example, and set it as the current component.
2. Save it and begin making customization changes.
3. When you reach the point where you want to script the changes you have made, select the **Script Component** Button. The Component Scripting page is displayed, with the component name you added in Step 1 in the File Name field.

4. In the **File Name** field, type the name you want assigned to the component files when you script this stage of the customization.
5. Type a description in the **Description** field.
6. Select the **Script Component** button.
7. Select the **Continue** button.The component name you added in Step 1 is still the current component on the Component Details page.
8. Make some additional customization changes. These changes are recorded as part of the current component, along with the changes you already recorded and scripted.
9. Specify a new file name and select the **Script Component** button. The component files generated contain all of the changes you made in the first part of the customization as well as the recent changes you made. You can continue to script changes in this way as many times as you need to, but you must make sure that you use a different, meaningful file name each time you script.
10. When you have fully completed the customization, it is advisable to carry out a final scripting, keeping the current component name in the **File Name** field. It is always recommended to script the current component before you move on to a new customization.
11. When you are installing the script to the second system, ensure that you select **Apply All Changes** if you want all of the changes that exist in the final version of the component to be installed. See Installing a Component (page 6-7).

Scripting Workflows

In Sage CRM version 6.1 and later, it is possible to use Component Manager to script workflows.

Changes made to workflows are recorded to the current component along with any other customizations.

When installing the component on the target system the following rules apply:

- If there is no workflow on the target system that has the same name, the workflow in the component will be created.
- Any workflow on the target system having a different name will not be affected by the installation of the component.
- If there is a workflow on the target system that has the same name, this workflow will be amended to incorporate the workflow from the component. The workflow tree will be replaced with the tree from the source system.
- If any workflow rules or states are deleted on the source system, the corresponding items will not be deleted on the target system. Deletions of workflow rules and states are not transferred by component manager. The items will still appear on the side bars of the workflow in the target system, although they are not used in the workflow tree anymore.

Note that Escalation rules are not recorded. See Introduction to Component Manager (page 6-1) for more information on what is and is not recorded by Component Manager.

Note: Components that were generated prior to 6.2 with workflow rules and loaded into 6.2 will have all workflow rules set to disabled. The user will be required to be enable rules individually after component is installed. Components generated with 6.2 will not be affected.

Saving a Component

Script files generated by Component Manager (see Scripting Customizations (page 6-4)) should be compressed to a ZIP format before installing to a second system.

When a component is scripted, the files are saved to the following location:

<CRM install path>\<installname>\inf

For example C:\Program Files\Sage\CRM\CRM\inf\

The following files and subfolder are created (where "MyComponent" is the filename chosen when scripting the component):

- MyComponent.ecf - this file contains the component name and description
- MyComponent\ - this subfolder contains MyComponent.es and also any other files that were part of the customization (for example ASP, .NET Dlls, TXT, JS or INC files)
- MyComponent.es - this is the actual script file containing the instructions to install the component items on the second system

All of the above files must be saved into a single ZIP file (e.g. MyComponent.ZIP). To create a ZIP file:

1. Open Windows Explorer and browse to the **inf** folder.
2. Select the **MyComponent** folder and **MyComponent.ecf**, right-click, and select **Send To | Compressed (zipped) Folder**. Windows will create the ZIP file for you.

This ZIP file will be used when you are installing the component (see Installing a Component).

Installing a Component

This section deals with the process of installing an existing component, created elsewhere, to your CRM system.

For information on creating component files see Recording Customizations (page 6-2) and for details on saving the components to a Component ZIP file see Saving a Component (page 6-7).

A Component ZIP file is a file containing the component script files that has been compressed into ZIP format. Component ZIP files may contain more than one component.

To install the component from a Component ZIP file:

1. Go to **Administration | Customization | Component Manager**, and select the **Components** tab.
2. In the **Add Components** panel, click **Browse** to select the ZIP file that contains the component files.
3. Click **Upload New Component**. The component name should now appear in the **Available Components** list.
4. Click on the component name in the **Available Components** list. The **View Details** button allows you to view more information about the component before it is uploaded, such as the version it was created in, and a detailed description of the component. Click **Install Component**.

   Note: you may see a message "No parameter information found in [component name]". This can be ignored unless you have modified the script to include

> parameter information. Please see Modifying Component Manager Scripts (page 6-11). If parameters were specified, the "Parameters, Step 1 of 2" page is displayed with a number of fields. Complete the fields and select the Install Component button to continue installing the Component. The "Parameters, Step 1 of 2" page may also be displayed without fields, but with information about the component you are installing.

5. On the next screen, you can click **Preview Install** again to see the actual script that will be executed when the component is installed along with a prediction of whether each step will be successful.
6. Select **Yes** or **No** in the **Apply All Changes** drop-down. The Apply All Changes drop-down is only relevant to changes made by previous components to the same objects (screens, fields, lists etc) that are being changed by the current component. If you choose **Yes** then everything in the component will be installed, overwriting any changes from made by previous components. Choose **No** to preserve changes from previous components. See Scripting Multi-Stage Customizations (page 6-5).

> In most circumstances you should select **Yes**. You should only select **No** when you have a specific reason for doing so as it may result your component being only partially installed.

7. Click **Install** to proceed with the actual installation. Component Manager starts to install the scripts. This involves:
   - Loading the new information.
   - Recreating the views.
   - Reloading the metadata.

> You will be warned if you install a component created in a newer version of CRM than the version that the component was created in. For example, a component created in version 6.3 will generate a warning if you try and install it in a 6.2 system.

8. Wait for the **Component Installed** message.

At this point you may wish to view the log file that has been generated. See **Component Manager Log File** for more information on the log file. Otherwise click **Continue** to return to the Components tab.

Component Manager Log File

A log file is generated automatically when a component is installed. Please see Installing a Component (page 6-7).

This log file can be viewed either:

- At the end of the installation process by clicking "View Log File".
- By browsing to Administration | System | Logging and selecting "Component Install Logs" from the drop-down.

The log file is in CSV format and so can be opened by Microsoft Excel or other spreadsheet programs. You will have to click to confirm that you wish to open this file type from your browser

The Log file contains two columns, and a row for each action attempted during the component manager installation.

The first column contains one of these values:

- **OK** - to indicate success of the action
- **Overwrite** - to indicate that the action will overwrite a previous change.
- **Fail** - to indicate failure

The second column contains the script for the attempted action.

Advanced Component Options

Advanced Component Options allows you to create or script new components that are based on customizations that were made within a certain date range and/or by a certain user. For example, Advanced Scripting would allow you to create a component that contained all customizations made by the System Administrator; or all customizations made within the last 14 days; or all customizations made by the System Administrator between May 1st and May 31st.

Components that are generated by Advanced Scripting will contain all changes that meet the selected criteria **regardless of whether Component Manager was turned on or not while the changes were being made**. This makes Advanced Scripting a powerful tool, however it is important to understand that **Advanced Scripting can change previously recorded components**, so please read and understand this section carefully before using the tool.

Advanced Component Options can be used for the same types of customizations that are recorded when Component Manager is turned on. See Introduction to Component Manager (page 6-1) for the list of customizations that can be recorded.

How to Generate a Component Script using Advanced Component Options

Review Scripting Customizations (page 6-4) for an overview of component scripting.

1. In **Administration | Customization | Component Manager**, select the **Component Details** tab
2. Click the **Advanced Scripting** button
3. On the Generate Advanced Component Script page, enter a **New Component Name**. This will be the name that is given to the generated script files.
4. Select **And** or **Or** to join the criteria fields. If you select **And** then the criteria (below) are restrictive; all must be true. If you select **Or** then any must be true for the customization to be included.
5. Optionally, you can select a component from the **Select To Include Existing Component** list. If you select a component, then all of the changes that have been saved to that component will be included in your generated script. Note that the **And** or **Or** selection includes this component as a criterion, so you are able to select, for example: **all customizations that are in MyComponent or that were changed in the previous month**.
6. Select the criteria that you want to use to specify the customizations that you want to include in the script. The possible criteria are:

Criterion	Details
Created Date	Select an absolute or relative date range. Includes all customizations that were created within the range.
Updated Date	Select an absolute or relative date range. Includes all customizations that were updated within the range.
Created By	Select a user. Includes all customizations that were created by the selected user.
Updated By	Select a user. Includes all customizations that were updated by the selected user.

7. Click the **Script Component** button. You are taken to the Component Scripting page.
8. Accept the default filename (unless you already have scripts with the same name that you wish to preserve) and enter a **description** for the component.
9. Click the **Script Component** button. The component is scripted and the generated file names and locations are displayed.
10. Return to the Components tab. You will see your component listed under **Available Components**.

You can now save this component for installation on another system. See Saving a Component (page 6-7).

How to Create a New Component Using Advanced Component Options

This procedure should be used with care as it may modify other components on your CRM system.

1. In **Administration | Customization | Component Manager**, select the **Component Details** tab.
2. Click the **Advanced Scripting** button.
3. On the Generate Advanced Component Script page, enter a **New Component Name**. This will be the name that is given to the new component.
4. Select **And** or **Or** to join the criteria fields. If you select **And** then the criteria (below) are restrictive; all must be true. If you select **Or** then any must be true for the customization to be included.
5. Optionally, you can select a component from the **Select To Include Existing Component** list. If you select a component, then all of the changes that have been saved to that component will be included in your generated script. Note that the **And** or **Or** selection includes this component as a criterion, so you are able to select, for example: **all customizations that are in MyComponent or that were changed in the previous month**.
6. Select the criteria that you want to use to specify the customizations that you want to include in the new component. The possible criteria are:

Criterion	Details
Created Date	Select an absolute or relative date range. Includes all customizations that were created within the range.

Criterion	Details
Updated Date	Select an absolute or relative date range. Includes all customizations that were updated within the range.
Created By	Select a user. Includes all customizations that were created by the selected user.
Updated By	Select a user. Includes all customizations that were updated by the selected user.

7. Click the **Mark Component** button. At this point the system will check whether any customizations that meet your criteria have already been included in other components. If there are any such customizations you will see a blue banner warning: **This will move customizations belonging to other components. Do you wish to Save Anyway?**.

   If you proceed, by clicking the Mark Component button again then the customizations that were in the other component(s) will be transferred to your new component. They will no longer be part of the old component(s), and if you script out the old components the customizations will not be included in the script.

8. You are taken to the Component Details tab. You will see your component listed under **Exiting Components**.

You can now record additional customizations to this component, or you can script it out for installation on another system. See Scripting Customizations (page 6-4).

Modifying Component Manager Scripts

Before you install components, you can modify the scripts that are generated in the ES and ECF files by Component Manager.

You can specify parameters in the ECF file (see Script Parameters (page 6-13)), and you can manipulate the JavaScript generated during the scripting stage in the corresponding ES script file. See Scripting Customizations (page 6-4) for more information on the ECF and ES file locations.

Modifications may be made in order to:

- Make a component usable in multiple installations
- Change screen names
- Add messages
- Copy ASP pages and other files from one location to another
- Create new tables
- Add new columns
- Search for and replace words in ASP pages.
- Modify reports
- Add views
- lDatabase, which returns the current installed database.
- lLocale, which indicates the installed locale. There are two constants that can be returned which are lWestern and lJapanese.
- sViewText, which can be used as a temporary storage buffer when scripting views.

Variable and constant names are case sensitive and can be used for any component APIs.

It is also possible to specify parameters to be passed to the script files. See Script Parameters (page 6-13).

Error Handling

There is an **errorhandling** option that allows you to control the way any errors are handled. By default any errors that occur during the running of the component are listed in the messages on the screen, but the component will keep running and all other changes specified in the ES file will be saved.

To change this behavior, add the following line to the ECF file:

```
errorhandling=strict
```

When this line is included:

- Any error will cause the component to stop running.
- All changes already made will be rolled-back.

Referential Integrity

From Sage CRM 7.0 there is new architecture being used to facilitate the interactive dashboard. To allow the persistence of the Meta Data Model within this architecture strict referential integrity needed to be enforced within Sage CRM meta data tables.

This means there is a requirement to order the code in components correctly within the ES script file. For example, you cannot add a view if the table the view is based on has not already been added.

The following is a list of custom table dependencies:

Custom_Edits - (Custom_Tables)

Custom_Views - (Custom_Tables)

Custom_ScreenObjects - (Custom_Tables, Custom_Views[optional])

Custom_Lists - (Custom_ScreenObjects, Custom_Edits)

Custom_ContainerItems - (Custom_ScreenObjects x2)

Custom_Tabs - (Custom_ScreenObjects)

Custom_Screens - (Custom_ScreenObjects, Custom_Edits)

FieldSecurity - (Custom_Edits)

UserSettings - (Users)

TerritoryPermissions - (Custom_Tables ,Users, TerritoryProfiles, Territories)

Channel_Link - (Users, Channel)

Users - (Channel, TerritoryProfiles, Territories)

Note: Using AddCustom_Data or RunSQLto update the above tables is problematic as the foreign keys will not be set automatically. Using the table appropriate method is the recommended practice. For example, to update Custom_Edits use AddCustom_Edits.

Script Parameters

When modifying Component Manager scripts, you may wish to pass parameters to your ES script file. This is accomplished by adding a Parameters section to the ECF file as explained below.

See Scripting Customizations (page 6-4) for information on the generation of the ES and ECF files, and see Modifying Component Manager Scripts (page 6-11) for an overview of modifying scripts.

Parameters must be one of the following types:

- TEXT
- CHECKBOX
- MULTITEXT
- PASSWORD
- INTEGER
- SELECT
- DATETIME
- DATE

To specify parameters, create a parameters section in the ECF file with the keyword *Params:* and put the

parameters on separate lines beneath the keyword in the following format:

```
<Parameter Type>
<Attribute=Value>,<Attribute=Value>,<Attribute=Value>
```

For example

```
Params:
TEXT Name=ServertName,Caption=Enter server name,Required=True
CHECKBOX Name=IncludeThis,Default=On,Caption=Include extras
PASSWORD Name=Password
INTEGER Name=NumUnits,OnChange=alert('You have entered'+NumUnits.value+' units.');
```

When you are installing the component, the fields are displayed with the attributes you

specified. To use the value(s) entered you call the Param() method in the ES script file (see Param). For example, to retreive the value entered to the 'Enter Server Name' text box, you need to call Param(ServerName) in the ES script file.

The following table describes the attributes that you can specify for parameters. All of the attributes are optional, except Name.

Attribute	Description
Name	Required. Name of the field. You can use this attribute with the Param function to get back the value entered by the user on the Parameter Info screen.
Default	The default value for the parameter.
NewLine	This is true by default. You set it to false if you want to keep the parameter on the same line as the previous one.
Rows	The number of rows that the parameter should take up. Default is 1.
Cols	The number of columns that the parameter will take up. Default is 1.

Attribute	Description
Required	Set to true to make sure the user enters a value for this parameter.Validation is done when the user clicks on the Install Component button.
ReadOnly	Set to true to show a read only value to the user.
Size	The number of characters that can be entered in a Text type parameter. Default is 20.
MaxLength	The number of characters that the parameter takes up on the screen. Default is 40.
Caption	This is the text that will appear beside the parameter on the screen.
CaptionPos	This defines the position of the caption relative to the value.
OnChange	JavaScript to be applied when the user changes the value in the field.
Attribute=Family	For parameters of type SELECT, this specifies what caption family to use to get the drop-down list.

Phone and E-mail Changes

A number of fields relating to phone and e-mail data have been dropped in 7.1 and two new link tables (for phone and e-mail) have been created. The views vPhone and vEmail have been dropped.

Components that reference these fields and views need to be reviewed.

CRM provides an error message if any inconsistencies are detected:

```
ChildCases70 jscript error: The view vListCases contained in
the view script is different to the view existing in the
database. The script view has been saved as vListCases_new.
The following fields are in your view but not in ours: Comp_
EmailAddress, Comp_FaxAreaCode, Comp_FaxCountryCode, Comp_
FaxNumber, Comp_PhoneAreaCode, Comp_PhoneCountryCode, Comp_
PhoneNumber, Pers_EmailAddress, Pers_FaxAreaCode, Pers_
FaxCountryCode, Pers_FaxNumber, Pers_PhoneAreaCode, Pers_
PhoneCountryCode, Pers_PhoneNumber
```

Other areas which may be impacted by the phone and e-mail changes are:

- Search SQL properties of fields
- Search Select Advanced
- SOAP Web Services code (wsdl is different)
- .NET code
- Block Usage
    - If you use fields directly in blocks (personboxshort) that are referenced
    - Self Service Block References

Please refer to the *Installation and Upgrade Guide* for more information on the Phone and E-mail table changes.

Component Manager Methods

The following methods and parameters are covered in this section:

AddCoachingCaptions (page 6-15)

AddColumn (page 6-16)
AddCustom_Captions (page 6-16)
AddCustom_ContainerItems (page 6-16)
AddCustom_Data (page 6-17)
AddCustom_Databases (page 6-18)
AddCustom_Edits (page 6-18)
AddCustom_Lists (page 6-19)
AddCustom_Report (page 6-20)
AddCustom_ReportBand (page 6-21)
AddCustom_ReportChart (page 6-21)
AddCustom_ReportField (page 6-22)
AddCustom_ReportGroup (page 6-22)
AddCustom_ScreenObjects (page 6-23)
AddCustom_Screens (page 6-24)
AddCustom_Scripts (page 6-25)
AddCustom_Tables (page 6-26)
AddCustom_Tabs (page 6-27)
AddLPCategory (page 6-27)
AddLPGadget (page 6-28)
AddLPLayout (page 6-28)
AddLPUserLayout (page 6-28)
FinishLandingPage (page 6-29)
AddMessage (page 6-29)
AddProduct (page 6-29)
AddView (page 6-29)
AddCustom_Relationship (page 6-20)
CopyAndDropColumn (page 6-30)
CopyAspTo (page 6-30)
CopyFile (page 6-30)
CreateNewDir (page 6-30)
CreateTable (page 6-31)
DeleteColumn (page 6-31)
DeleteCustom_Caption (page 6-31)
DeleteCustom_Captions (page 6-32)
DeleteCustom_Field (page 6-32)
DeleteCustom_Screen (page 6-32)
DeleteCustom_ScreenObjects (page 6-32)
DropTable (page 6-33)
DropView (page 6-33)
DropConstraint (page 6-32)
FileOpen (page 6-33)
GetDLLDir (page 6-34)
GetInstallDir (page 6-34)
Param (page 6-34)
QueryResultsToFile (page 6-34)
RunSQL (page 6-35)
ProgressScriptTransaction (page 6-34)
SearchAndReplaceCustomFile (page 6-35)
SearchAndReplaceInDir (page 6-35)
SearchAndReplaceInFile (page 6-35)
TableExists (page 6-35)

AddCoachingCaptions

Description	Adds a new coaching caption, and returns its id.
Parameters	Coch_ActionID: The action number of this caption. Coch_CaptCode: The caption code for this caption.

AddColumn

Description	Use this to physically add a column to an existing table.
Parameters	Col_TableName: The actual name of the table to which the column is to be added. Col_ColumnName: The new column name. Col_Type: The CRM entry type that this column is to have. Col_Size: For certain types of field, for example, text fields, specify how many characters it is to be. Col_AllowNulls: If this column allows null values. True for allow nulls, False otherwise. Col_IsUnique: If this column must have unique values. True for unique values, False otherwise. Col_IsIdentity(Boolean): indicates whether the column should be created as an auto-incrementing field.

AddCustom_Captions

Description	Use this to add/change translations in the system. Also returns the Id value of the record added.
Parameters	Capt_FamilyType: The Family type that this caption belongs to. For example, Tags. Capt_Family: The family for this caption. Capt_Code: The Code for this caption. Captions are identified by their Family and Code. Capt_Order: The order that this should appear in. Capt_US: The US English translation. Capt_UK: The UK English translation. Capt_FR: The French translation. Capt_DE: The German translation. Capt_ES: The Spanish translation. Capt_DU: The Dutch translation. Capt_JP: The Japanese translation. Capt_IntegrationID :(Optional) The identifier of the integration that this caption belongs too. Not used outside of the integration module

AddCustom_ContainerItems

Description	Use this to add blocks to a container. Also returns the Id value of the record added.
Parameters	Cont_Container: Name of the container. Cont_BlockName: Name of block to be added. Cont_Order: SYSINT Cont_NewLine: Set to 1 for new line and 0 for the same line. Cont_Width: Sets width. Cont_Height: Sets height. Cont_Deleted: Flag to indicate if record is deleted.

AddCustom_Data

Description	Adds or updates data in any table in CRMTableName: Name of the table to update. Using AddCustom_Data to update some custom tables(Custom_Edits, Custom_Views, Custom_ScreenObjects, Custom_List, Custom_ContainerItems, Custom_Tabs, Custom_Screens) is problematic as the updates to the foreign key will not be set automatically. Using the table appropriate method is the recommended practice e.g. to update Custom_Edits use AddCustom_Edits
Parameters	**TablePrefix**: The prefix of the table. **IdColumn**: The name of the column on the table that holds the Id value. **Fields**: A comma separated list of fields to update. **Values**: A comma separated list of values to match up with the fields. The special values ISBLANK, ISNULL, ISNOW can be used in place of field values. ISBLANK is equal to ""; ISNULL marks column as NULL; ISNOW will write the current date and time, to the nearest second. If the value is targeted for a string field it should be enclosed in double quotes. For example: *Please refer to Developer Help files for code sample*. **KeyFields**: A comma separated list with the indices of the fields from the fields list that are to be used to identify if a record already exists-and therefore whether to update it or insert it. For example, this code updates the Custom_Tables table and sets the field Bord_SoloOptions to be 5 where Bord_DatabaseId is null and Bord_Name = 'UserContacts'. *Please refer to Developer Help files for code sample*. **ExtraSQL**: String. Default value is blank. If this is passed in, it is used as an extra condition that must be met in order to update the record. If the record exists, it is updatedonly if the ExtraSql condition is true. If the record does not exist, it is inserted as usual. If the record exists but does not satisfy the ExtraSql condition, the function returns 0 to indicate that no record was updated. For example: *Please refer to Developer Help files for code sample*. This updates the Capt_DU column of custom captions, only where the existing value is NULL.

AddCustom_Databases

Description	Use this to add links to external databases. Also returns the Id value of the record added.
Parameters	Cdbo_Description: Description of the other database. Cdbo_AliasName: Alias for the database. Cdbo_UserName: Username to use to connect to database. Cdbo_Password: Password to go with above username. Cdbo_Deleted: Flag to indicate if record is deleted. Leave as null. Cdbo_DriverName: Type of driver to use to connect to database. Cdbo_ServerName: Name of the server where the database resides. Cdbo_DatabaseName: Actual database name.

AddCustom_Edits

Description	Use this to add/change the properties of a field in CRM. Also returns the Id value of the record added.
Parameters	ColP_Entity: The name of the entity. ColP_ColName: The name of the field. ColP_EntryType: The entry type. Please refer to the CRMEntryBlock Object for a list of entry types. ColP_DefaultType: The default type of the column. ColP_DefaultValue: If the default type of the column is to use a default value, then enter the default value here. ColP_EntrySize: How many characters will be seen at a time when editing the column. ColP_LookUpFamily: If applicable for the entry type then this is a string with the lookup family name. ColP_LookUpWidth: The width of the column. ColP_Required: Set to Y if this column must be filled in, null or N otherwise. ColP_AllowEdit: Set to N if this column is readonly, and set to Y or null otherwise. ColP_SearchDefaultValue: If applicable to the entry type, this is the search default value. ColP_System: Set to Y if this is a system column and is not to appear on customization screens, set to N or null otherwise.

AddCustom_Lists

Description	Used to add columns to a customized List group. Also returns the Id value of the record added.
Parameters	GriP_GridName: The name of the grid in which this column appears. GriP_Order: The order in which this column appears in the grid. GriP_ColName: The name of the column. GriP_AllowRemove: Flag to indicate if this column can be removed or not. 'Y' to allow removal, 'N' for not. GriP_AllowOrderBy: Flag to indicate if the list can be ordered by this column. Y to allow order by. N or null otherwise. GriP_OrderByDesc: Flag to indicate if the order by should start with descending order. GriP_Alignment: Code to indicate the column alignment. CENTER (or NULL) for center, LEFT for left, RIGHT for right. GriP_Jump: Code to say if this column can hyperlink to another page. GriP_ShowHeading: Flag to indicate if the heading should show for this column in the list or not. Y for show heading. N or null otherwise. GriP_ShowSelectAsGif: Flag to indicate if this column should show as a gif. Y for yes, N or null otherwise. GriP_CustomAction: String with name of page to hyperlink to if the Jump is set to Custom. GriP_CustomIdField: String with name of field to use as the id field if the Jump is set to Custom. GriP_DeviceID: The device id of this list, look up from Devices table. Grip_CreateScript: String. Lets you set a create script on a column. The default value is blank.

AddCustom_Report

Description	Used to create a report
Parameters	Repo_Category: String Repo_Name: String Repo_Title: String Repo_Description: String Repo_Bands: Integer Repo_ExportAsXML: String Repo_FooterCentrePageData: String Repo_FooterLeftPageData: String Repo_FooterRightPageData: String Repo_HeaderCentrePageData: String Repo_HeaderLeftPageData: String Repo_HeaderRightPageData: String Repo_FooterCentrePageDataImage: String Repo_FooterLeftPageDataImage: String Repo_FooterRightPageDataImage: String Repo_HeaderCentrePageDataImage: String Repo_HeaderLeftPageDataImage: String; Repo_HeaderRightPageDataImage: String Repo_PrintOptions: Integer Repo_UserFilterField: String Repo_PrivateUserID: Integer Repo_ReportStyle: String

AddCustom_Relationship

Description	Used to add entity relationships for SData provider
Parameters	TableName: String ColumnName: String TableNameRelated: String ColumnNameRelated: String RelationshipType: Integer IsCollection: Boolean LinkTableName: String LinkColumnName: String LinkColumnNameRelated: String

AddCustom_ReportBand

Description	Used to add a report band.
Parameters	ReBa_ReportID: Integer ReBa_DetailLevel: Integer ReBa_CrossTabField: String ReBa_DisplayOptions: Integer ReBa_ViewName: String ReBa_WhereClause: String ReBa_DetailLinkField: String ReBa_MasterLinkField: String

AddCustom_ReportChart

Description	Used to add a chart to a report.
Parameters	ReCh_ReportID: Integer ReCh_Options: Integer ReCh_BackImageName: String ReCh_BarStyle: String ReCh_BottomCaption: String ReCh_BottomDateFunction: String ReCh_BottomFieldName: String ReCh_Elevation: Integer ReCh_GradientEndColour: String ReCh_GradientStartColour: String ReCh_Height: Integer ReCh_HorizontalOffset: Integer ReCh_LeftCaption: String ReCh_LeftFieldName: String ReCh_LeftFunction: String ReCh_LegendAlignment: String ReCh_MarksStyle: String ReCh_Perspective: Integer ReCh_PieRotation: Integer ReCh_Rotation: Integer ReCh_Style: String ReCh_Tilt: Integer ReCh_VerticalOffset: Integer ReCh_Width: Integer ReCh_Zoom: Integer

AddCustom_ReportField

Description	Used to add a report field.
Parameters	ReFi_ReportBandID: Integer ReFi_Alignment: String ReFi_DataField: String ReFi_UsageType: String ReFi_DisplayOrder: Integer ReFi_JumpDestination: String ReFi_JumpFileName: String ReFi_JumpIdentifier: String ReFi_Mask: String ReFi_TotalsType: String ReFi_SortOrder: Integer

AddCustom_ReportGroup

Description	Used to add a report group.
Parameters	ReGr_ReportBandID: Integer ReGr_GroupByField: String ReGr_GroupOrder: Integer ReGr_HasFooter: String ReGr_JumpDestination: String ReGr_JumpFileName: String ReGr_JumpIdentifier: String

AddCustom_ScreenObjects

Description	Use this to add a new list or screen to the system. Also returns the Id value of the record added.
Parameters	CObj_Name: The name of the custom_screenobject.This must be unique. CObj_Type: The type of the object. This can be List, Screen, SearchScreen, TabGroup, filterbox, or Block. CObj_EntityName: The name of the entity on which this object is based. CObj_AllowDelete: If a user is allowed to delete this object. Set to Y if allowed to delete. CObj_Deleted: If this object is deleted. Set to 1 if it is deleted. CObj_TargetTable: The table from which fields can be added to this object. CObj_Properties: Comma separated list of properties. For internal use only. CObj_CustomContent: Custom script that can be added to the object. CObj_UseEntity: The table from which fields can be added to this object. CObj_TargetList; Internal use only. CObj_Ftable: Internal use only. CObj_FtableFCol: Internal use only. CObj_CheckNameOnly: Optional. If set to true the screen object will be updated checking name only (instead of checking against name AND entity).

AddCustom_Screens

Description	Use this to add or amend fields on a screen. Also returns the Id value of the record added.

Parameters	SeaP_SearchBoxName: The name of the screen in which this field appear. This should match Cobj_Name from Custom_ScreenObjects table.
	SeaP_Order: The order in which this field appears within the screen.
	SeaP_ColName: The column name of this field.
	SeaP_Newline: Number to indicate if this field should show on a new line. Set to 1 for new line and 0 for the same line.
	SeaP_RowSpan: Number of rows that this field should span on the screen.
	SeaP_ColSpan: Number of columns that this field should span on the screen.
	SeaP_Required: Set to Y if the field is required, N if it is not.
	SeaP_DeviceID: Number of the device that this screen is for. Look up from the Devices table.
	SeaP_OnChangeScript: Client-side javascript to apply to the onchange event of the field.
	SeaP_ValidateScript: Server side script to be run to validate the field.
	SeaP_CreatedScript: Server side script to be run when field is created.
	SeaP_Jump: Where this field will jump to if clicked.
	
	Note that instead of using AddCustom_Screens, you can set some prefined variables and then call AddEntryScreenField. For example this use of AddCustom_Screens:
	`AddCustom_Screens('GlobalLibraryFilterBox',1,'libr_filename',0,1,1,'N',0,'','','','');`
	can also be written like this:
	`EntryScreenName='GlobalLibraryFilterBox';`
	`FieldOrder=1;`
	`FieldName='libr_filename';`
	`NewLine=false;`
	`RowSpan=1;`
	`ColSpan=1;`
	`Required=false;`
	`AddEntryScreenField();`

AddCustom_Scripts

Description	Use this to add table and entity level scripts. Also returns the Id value of the record added.
Parameters	CScr_TableName: The name of the table to which this script applies. CScr_ScriptName: The name of the script. CScr_Order: The execution order in which this script should be run, if there are more than one script on a table. CScr_Script: The actual script. CScr_ScriptUser: The name of the user that the script should run under, if applicable. CScr_ScriptType: This can be one of 4 values that correspond to the options available for Script Type. The values are ‘entity’ (Entity Level Script), ‘entitywrb’ (Entity Level with Rollback), ‘tls’ (Table Level) and ‘tlsdetached’ (Detached Table Level). Note that the parameter in this position used to be for CScr_IsEntityScript which had values of ‘Y’ or ‘N’. Every effort has been made to ensure backward compatibility for existing scripts still using this old parameter however it is advised to update any scripts using this to use the ScriptType instead. CScr_UseRollBack: Y/N flag to indicate if rollback is available on this script. CScr_ViewName: The name of the view that this script should use to get information from.

AddCustom_Tables

Description	This is used to add a table. Also returns the Id value of the record added.
Parameters	Bord_Caption: The caption given to this table. Bord_System: Flag to indicate if this is a system table or not. Set to Y for system tables. System tables cannot be seen via Administration | Customization. Bord_Hidden: Flag to indicate if this table is to be hidden. Set to Y for hidden tables. Hidden tables cannot be seen in Administration | Customization. Bord_Name: The actual physical table name of the table. Bord_Prefix: The prefix that is attached to all the fields in this table. Bord_IdField: The name of the field in the table that holds the unique id of each row. Bord_PrimaryTable: Flag to indicate if this table is a primary table, (and thus if it has territory security on it). Set to Y for primary tables, N or Null otherwise. Bord_ProgressTableName: the name of the table used for workflow progress. Bord_ProgressNoteField: the name of the field used for workflow tracking notes Bord_WorkflowIdField (Boolean): whether the table has a <prefix>_ WorkflowId field Bord_DatabaseId: The Id of the database that this table is in. Lookup from Custom_Databases, leave blank for tables in the main CRM database.

AddCustom_Tabs

Description	Used to add or edit tabs in tab groups. Also returns the Id value of the record added.
Parameters	Tabs_Permission: Used for internal CRM tabs only. Should be set to 0. Tabs_PerLevel: Used for internal CRM tabs only. Should be set to 0. Tabs_Order: The order in which this tab appears in the tabgroup. Tabs_Entity: The name of the tab group. Tabs_Caption: The caption for this tab. Tabs_Action: String name of action for this tab. Tabs_Customfilename: Name of ASP page to use for this tab, if the action is set to CustomFile. Tabs_WhereSQL: SQL clause that restricts the appearance of the tab. Tabs_Bitmap: Bitmap used for the tab. Tabs_DeviceId: The device id of this list, look up from Devices table. Tabs_SecurityEntity: the entity used to determine whether user has permission to see/use that tab Tabs_OnlineOnly (boolean): used to indicate whether the tab is visible on Solo clients. Tabs_Deleted(Integer):used to indicate if the tab is deleted or not. Tabs_InButtonGroup: used to implement button groups.Set to 1 if implementing a button group.

AddLPCategory

Description	Adds a dashboard category. Result is category id that was added
Parameters	Name: the name of dashboard category ParentId: id of parent category Deleted: should be 0 if not deleted

AddLPLayout

Description	Adds a dashboard. Result is dashboard id that was added
Parameters	Name – name of dashboard
	Description – description of dashboard
	CategoryId – category that dashboard belong to (0 if none)
	LayoutXml – dashboard layout Xml data (all gadget Ids have to be replaced with markers for FinishLandingPage() method)
	LayoutType – type of layout (currently one only)
	Deleted – should be 0 if not deleted
	IsTemplate – if dashboard is template
	TemplChannels – CRM teams that dashboard is assigned to
	TemplUsers – CRM users that dashboard is assigned to
	SourceId – id of source dashboard (for FinishLandingPage() method)

AddLPGadget

Description	Adds a gadget.
Parameters	Name – name of gadget
	Description – description of gadget
	GadgetType – type of gadget
	LayoutXml – gadget layout Xml data (all gadget Ids have to be replaced with markers for FinishLandingPage() method)
	DataBinding – xml data about CRM data source to be used with gadget (CRM report id, CRM entity id etc.)
	LayoutId – dashboard id that gadget belongs to
	CategoryId – category Id that gadget belongs to (not used)
	Deleted – should be 0 if not deleted
	SourceId – id of source gadget (for FinishLandingPage() method)

AddLPUserLayout

Description	Assigns template layout id to user
Parameters	LayoutId – dashboard id
	TemplateLayoutId– template dashboard id
	UserID – user id
	Deleted – should be 0 if not deleted

FinishLandingPage

Replaces all the markers in layoutXml with new ids collected during creation of dashboards and gadgets.

This must be run after the other interactive dashboard methods. The interactive dashboard methods should be run in this order:

AddLPCategory

AddLPLayout

AddLPGadget

AddLPUserLayout

FinishLandingPage

AddMessage

Description	Use this to show information to the user while the component is being installed.
Parameters	Ms_Message: String with message to be shown on-screen.

AddProduct

Description	Used to add new products.
Parameters	Prod_Name: String. Prod_Description: String. Prod_ListPrice: String. Prod_Image: String. Prod_APR: String.

AddView

Description	Used to add views.
Parameters	AViewName: The name of the view. AEntity: The entity the view relates too. For system and hidden entities, use the System entity. ADescription: A free text description of the view. AViewScript: The sql script of the view. ACanEdit: If true, then the view can be edited. ACanDelete: If true, then the view can be deleted. AReportsView: If true, then the view can be used in reports. ATargetsView: If true, then the view can be used in groups. ForceOverwrite: If true, and there are differences between the view in the database and the view being installed because the customer has customized their install, then overwrite the customer's changes with ours. ASearchView: If true, then the view can be used by keyword searches

CopyAndDropColumn

Description	Used to alter the properties of a column by creating a temporary column, copying over the data, dropping the old column, then renaming the temporary column to the old column name.
Parameters	Col_TableName: The name of the table. Col_ColumnName: The name of the column. Col_Type: The new data type of the column. Col_Size: The new size of the column. Col_Allow_Nuls: Boolean value to indicate whether the column allows nulls.
Return Value	None

CopyAspTo

Description	User to copy an ASP file from one location to another.

Parameters	CTo_FileName: From file path /name CTo_NewFileName: To file path /name. For example:
	`Copy-AspTo('custompages\\subfolder\\edit.asp','custompages\\subfolder\\edit.asp');`
	It is also permissible to use relative paths, as in this example:
	`CopyAspTo('custompages\\Edit.asp', '..\\custompages\\system\\Edit.asp');`

CopyFile

Description	Copies the source file to the target file.
Parameters	SourceFile: The name of the file to be copied TargetFile: The new filename
Return Value	None

CreateNewDir

Description	This function creates the specified directory.
Parameters	DirName: The name of the directory to be created.
Return Value	None

CreateTable

Description	Use this to physically create a table in the database. The table will be created with the CRM standard fields on it. That is, CreatedBy, CreatedDate, UpdatedBy, UpdatedDate, TimeStamp, and Deleted.
Parameters	Cr_Tablename: The actual table name. Cr_Prefix: The prefix that will be added to every column in the table. Cr_ Identity: The name of the identity column in the table (this must start with the prefix). Cr_PrimaryTable: If this is to be a primary table in CRM. Primary tables will have the security territory column added to them. Set to True for primary tables, false otherwise. SystemTable: If this is a system table, set to false. HiddenTable: If this is to be a hidden table, set to True-that is, if the table is not to be seen within Administration | Customization. Cr_WorkflowIdField: indicates whether this table should have a <prefix>_WorkflowId field. Cr_ProgressTableName: the name of the table used for progressing, i.e. like CaseProgress. Cr_ProgressNoteField: the name of the field used for notes progress. Cr_NoIDCol (Boolean):indicates whether the table has an auto-incrementing field.

DeleteColumn

Description	Drops the specified column from the specified table.
Parameters	TableName ColumnName
Return Value	None

It is recommended practice to use the Delete_CustomField method for removing columns.

DeleteCustom_Caption

Description	Deletes the specified caption.
Parameters	Capt_FamilyType Capt_Family Capt_Code
Return Value	None

DeleteCustom_Captions

Description	Deletes the specified caption family
Parameters	Capt_FamilyType: Capt_Family:
Return Value	None

DeleteCustom_Field

Description	Deletes the specified field from all screens, lists, reports, saved searches, notifications, etc.
Parameters	ATableName AColumnName
Return Value	None

DeleteCustom_Screen

Description	Used to delete a screen.
Parameters	SeaP_SearchBoxName: Deletes all the items for the specified screen name regardless of device.

DeleteCustom_ScreenObjects

Description	Used to delete screen objects.
Parameters	CObj_Name: The name of the custom_screenobject from which deletes are based. DeleteHeader: This flags whether the record of this object from the "Custom_ScreenObjects" table is deleted. If this is false, the record is not deleted but the sub items are always deleted from: • Custom_Lists • Custom_Screens • Custom_ContainerItems CObj_DeviceID: If this is specified (not '0' or '1'...see below) then only the specific items for this device are deleted. If this is '0' then all records for every device are deleted. If this is '1' then only desktop information is deleted.

DropConstraint

Description	Used to drop any constraint from a table in the database - e.g. foreign key
Parameters	AConstraintName: String. The name of the constraint to drop. This procedure will delete the constraint from the database. ATableName: String. The name of the table the constraint is on.

DropTable

Description	Used to drop a table from the database.
Parameters	ATableName: String. The name of the table to drop. This procedure will delete the table from the database.

DropView

Description	Used to delete/drop views.
Parameters	AViewName: String. The name of the view to drop. This deletes the view from the database.

FileOpen

Description	Use this to open a CSV or Excel file and read in the values. This method will open the file and return a "DataFile" object which can then be used to process the values in the file. See info on the "DataFile" object for more details.
Parameters	AFileName. WideString. The full name of the file, including the drive and directory.

Return Value	DatFile Object The "DataFile" Object has the following two properties: EOF. Boolean value, returns if the end of the file has been reached. FieldCount. Integer value, returns the number of columns in the file (based on the current row). The "DataFile" Object has the following two methods:
	
	NextRow()
	
	GetField()

NextRow()

Description	Skips the file pointer on to the next row in the file
Parameters	None

GetField()

Description	Returns the value in the given field for the current row.
Parameters	AIndex-an integer value indicating the column number to return the value for. 0 returns the value in the first column for the current row, and so on.

GetDLLDir

Description	This function returns the full path to the eware.dll.
Parameters	None
Return Value	String

GetInstallDir

Description	This function returns the name of the folder on the server where the install is.
Parameters	None
Return Value	String For example: *Please refer to Developer Help files for code sample*.

Param

Description	Use this to get back a parameter value.
Parameters	Pr_SearchNam: Parameter value name.

ProgressScriptTransaction

Description	This function will commit (to the database) what has already been processed and start a new transaction.
Parameters	CommitOnError (optional) – Commit db transactions even if there was an error.
Return Value	None

QueryResultsToFile

Description	This function runs a given SQL select statement and writes the results to a CSV file on the server.
Parameters	FileName: String. This should be the full path and file name of the file to write to. QueryString: String. This should be a SQL select statement.
Return Value	String With either an error message or a message saying how may rows were exported. For example: *Please refer to Developer Help files for code sample*.

RunSQL

Description	RunSQL allows you to execute any sql statement, from simple things to inserting or updating a record, to creating and dropping tables and views. It can be used to handle scripting that can't be accomplished by one of the other methods. Any SQL Script will execute here, so great care should be taken when using this method. In particular, extensive testing is recommended before using RunSQL in a production environment.
Parameters	Sql: The SQL Script
Return Value	None

SearchAndReplaceCustomFile

Description	Used to search for and replace text in a specified file.

Parameters	Sr_FileName: The filename including the full path;
	Sr_StringToSearch: String;
	Sr_ReplaceString: String;

SearchAndReplaceInDir

Description	Used to search for and replace text in all files in a specified directory.
Parameters	ADirPath: String AStringToSearch: String AReplaceString: String

SearchAndReplaceInFile

Description	Used to search for and replace text in a specified file. File must be located in <installdir>\CustomPages\ directory

Parameters	Sr_FileName: String;
	Sr_StringToSearch: String;
	Sr_ReplaceString: String;

TableExists

Description	Returns true if the specified table exists in the custom_table
Parameters	TableName
Return Value	Boolean

Component Manager Scripting Examples

This section takes you through the following examples:

Example: Multiple Installs (page 6-36)
Example: Changing A Screen Name (page 6-36)
Example: Adding A Message (page 6-36)
Example: Copying An ASP Page (page 6-37)
Example: Replacing Text In An ASP Page (page 6-37)
Example: Creating A New Table (page 6-37)
Example: Adding A New Column (page 6-37)
Example: Using the DataFile Object (page 6-38)
Example: Adding a New View (page 6-38)

Example: Multiple Installs

You can specify that a component can be used for multiple installs. You might find this useful if you install a component, make further customization changes, and then want to undo them. Rather than undoing the changes manually, you can simply reinstall the component on a clean CRM install.

To specify that a component can be used for multiple installs:

1. Open the ECF file, and type **multipleinstalls=y**.
2. Save the change you made.

Example: Changing A Screen Name

To change a screen name, for example from DemoScreen1 to Demo:

1. Open the **ECF** file, and type:
   ```
   Params:
   Text Name=ScreenName,Caption=Type new screen name here,Required=True
   ```
2. Open the **ES** script file and change the following script:
   ```
   AddCustom_ScreenObjects('DemoScreen1','Screen','Opportunity', 'Y','0','','','');
   ```
   To
   ```
   AddCustom_ScreenObjects('Param(ScreenName'),'Screen','Opportunity',
   'Y','0','','','');
   ```
3. Save both files and proceed to install the component.
4. When the Parameter Info input screen displays, type **Demo** in the Screen Name field, and click **Continue**. When you install the component, DemoScreen1 is renamed Demo and the record is added to the custom ScreenObjects Table.

Example: Adding A Message

To add a message to the end of an installation:

1. Open the **ECF** file, and type:
   ```
   Params:
   Text Name=EntityName,Caption=Enter new name
   ```
2. Open the **ES** script file and add the following script:
   ```
   AddMessage('A new screen called '+Param('EntityName')+' was installed into CRM.');
   ```
3. Save both files and proceed to install the component.

4. When the Parameter Info screen displays, type Demo in the Entity Name field, and select the **Continue** button.
5. When you install the component, the following message will display at the end of the installation: "A new screen called Demo was installed into CRM".

Example: Copying An ASP Page

To copy an ASP page from one location to another:

1. Open the **ES** script file and add the following script:

```
CopyAspTo('custompages\\subfolder\\edit.asp','custompages\\subfolder\\edit.asp');
```

2. When you install the component, EDIT.ASP will be copied from the Phase1 component directory to the following location in the current installation: \custompages\subfolder\edit.asp

Example: Replacing Text In An ASP Page

To search for and replace text in an ASP page:

1. Open the **ES** script file and add the following script.

```
SearchAndReplaceInFile('Edit.asp','Find','Search');
```

2. When you install the component, any instances of the word Find in EDIT.ASP are replaced with the word **Search**.

Example: Creating A New Table

To create a new table:

1. Open the **ES** script file and add the following script:

```
CreateTable('DemoTable','dem','demo','false','false','false');
```

2. When you install the component, a table called **DemoTable** will be added to CRM. The following columns will be automatically created for the table:

   Dem_TableId
   Dem_ System
   Dem_ CreatedBy
   Dem_ CreatedDate
   Dem_ UpdateBy
   Dem_ UpdateDate
   Dem_ TimeStamp
   Dem_ Deleted

Example: Adding A New Column

To create a new column:

1. Open the **ES** script file and add the following script:

```
AddColumn('DemoTable','Dem_Description',10,'(25)','True','False');
```

2. When you install the component, a column called **Dem_Description** will be added to the DemoTable created in Example: Creating A New Table (page 6-37).

Example: Using the DataFile Object

The following is an example of using the **DataFile** object to loop through the rows in a spreadsheet and perform actions with the values found.

- Open the **ES** script file and add the following script:

```
var MyFile = FileOpen('c:\\data\\mydata.xls');
var i = 0;
while (!MyFile.EOF)
{
       i = 0;
       while (i < MyFile.FieldCount)
       {
               sValue = MyFile.GetField(i);
               //do something with value
               i++;
       }
       MyFile.NextRow();
}
```

Example: Adding a New View

The following is an example of a component script for adding a view. This example also shows the use of the **iDatabase** variable.

To add a view:

- Open the **ES** script file and add the following script:

```
Please refer to Developer Help files for code sample
```

Chapter 7: Graphics and Charts

In this chapter you will learn how to:

- Get an overview of charts.
- Get an overview of effects for pie charts only.
- Get an overview of special effects for charts.
- Use external data to create charts.
- Work with chart examples.
- Get an overview of graphics.
- Get an overview of graphics formats.
- Discuss performance tips for using graphics.
- Use external images.
- Work with graphics examples.
- Get an overview of graphics effects.
- Get an overview of animation.

Introduction

This section provides an introduction to how to implement Sage CRM Graphics and Charts in an ASP page:

- Graphics Overview (page 7-8)- Graphics can be generated dynamically within CRM. They can also be data aware and therefore representative of data stored at the time the graphic is generated.
- Charts Overview (page 7-1) - Charts directly inherit from Graphics and are an extension of the Graphic Block that allow the production of different types of charts. As such they can be generated dynamically within CRM. They can also be data aware and therefore representative of data stored at the time the chart is generated. Much of what can be applied for graphics applies to charts.

Charts Overview

Charts directly inherit from Graphics and are an extension of the Graphic Block that allow the production of different types of charts. As such they can be generated dynamically within CRM. They can also be data aware and therefore representative of data stored at the time the chart is generated. Much of what can be applied for graphics applies to charts.Charts, like graphics, can be generated with just a few commands, or customized at length.

From v7.1 animated and interactive charts based on Fusion Charts v3.2 (www.fusioncharts.com) are available.

Fusion Charts are rendered as JPGs in ASP and .NET.

Pre-v7.1 charts map to Fusion Charts as follows:

Pre v7.1 Chart Type	Fusion Chart Type
Line	Line

Pre v7.1 Chart Type	Fusion Chart Type
Bar	Column2D
Area	Area2D
Point	Line
HBar	Bar2D
Pie	Pie2D
FastLine	Line

Fusions Charts System Parameters

The following system parameters can be used to influence the appearance of Fusion Charts.

System Parameter	Description
ChartTimeoutSeconds	Timeout in seconds for the fusion charts being converted to images for display in Sage CRM
ChartUseFlash	Master switch for toggling between Flash and images
ChartOverruledWidthID	Overrule Width for Interactive Dashboard chart width
ChartOverruledHeightID	Overrule Height for Interactive Dashboard chart height
ChartOverruledWidthCD	Overrule Width for Classic Dashboard chart width
ChartOverruledHeightCD	Overrule Height for Classic Dashboard chart width
ReportsShowTextErrorInsteadOfImage	Works in collaboration with ChartUseFlash - allows errors (no data and no flash to be displayed as customized images in accordance with the filenames below)

Pie Chart Only Effects

These are effects that can only be applied to a populated pie chart.

- RotatePie. Performing the same as rotation but specific to pie charts to produce a smoother effect. For example:

```
Effect('RotatePie','45');
```

- PiePatterns. Specific to Pie Charts, this effect allows for patterns to be used in pie charts. This could prove more effective for printing. For example:

```
Effect('PiePatterns','true');
```

- ExplodePie. This can extract a slice of a pie chart by a varying amount. The first parameter is the index of the slice and the second is the amount. For example:

  ```
  Effect('Explode','6,20');
  ```

Special Effects For Charts

When using a Chart Block, the effect command provides a host of new effects that can be applied to a chart, in addition to the effects described previously. Note that these effects can be carried out in an ASP scripting loop and reproduced as part of an animation.

- **Elevation**, This describes front plane rotation and can take a value from -90 to 90 degrees of elevation. For example:

  ```
  Effect ('Elevation','45');
  ```

- **Perspective**. This sets the view of the chart with perspective effect. The value passed to it is a percentage from 0 to 100. For example:

  ```
  Effect('Perspective','45');
  ```

- **Tilt**. This carries out a rotation on the chart within the Chart Panel. Values passed to it from 0 to 360 rotate it anti-clockwise, and negative values rotate it clockwise. For example:

  ```
  Effect('Tilt','10');
  ```

- **Rotation**. Rotation describes front plane rotation by the specified degrees. Increasing the value positively brings the right of the Chart towards the viewer and the left of the Chart away, moving around a vertical axis at the central horizontal point of the Chart. For example:

  ```
  Effect('Rotate','45');
  ```

- **HorizOffset**. This can displace the position of a chart horizontally by the number of pixels specified. For example:

  ```
  Effect('HorizOffset','45');
  ```

- **VertOffset**. This can displace the position of a chart vertically by the number of pixels specified. For example:

  ```
  Effect('VertOffset','45');
  ```

Using External Data for Charts

Bubble Charts

For Bubble Charts, the manual chart entry takes on this form:

```
ManualChartEntry("Xpos,Ypos,Radius [,Label] [,Color] );
```

For example:

```
// Adds a bubble of radius 5 at 100,100
ManualChartEntry('100,100,5',false);
// Given the label "Jan"
ManualChartEntry('100,100,5,Jan',false);
// Drawn in Yellow
ManualChartEntry('100,100,5,Jan,Yellow',false);
```

Chart Examples

Example: Adding a New Chart

Add a Bar Chart that Shows Opportunity Forecast

For this example, CRM connects to the opportunity table and obtains the information for opportunity forecasts. The chart must be aware of the user's context. It needs a field to uniquely identify the current opportunity. From v7.1, Fusion Charts v3.2 are used, this means charts may need to be resized using mychart.Resize(600,600).To create a chart for opportunity forecast, you need to:

1. Add a new tab that links to a custom page.
2. Create the custom page to display the chart of the opportunity forecast for the current opportunity.To create the custom page to display the chart of the forecast for the current opportunity:
3. Retrieve the identifying value for the current Opportunity and assign it to a variable:

```
Var

OppId=CRM.GetContextInfo('opportunity','oppo_opportunityid');
```

4. Create a block for the chart and assign it to a variable:

```
var chart;

chart=CRM.GetBlock('chart');
```

5. Issue commands to the chart to define its data and style:

```
with (chart)
{
Stylename('Area');
Title='Forecast over time';
Description='Forecast of Opportunity over time';
Resize(600,600);
ShowLegend(false);
MinY=0;
LabelX='Date';
LabelY='Forecast';
XProp="Oppo_Forecast";
XLProp="Oppo_CreatedDate";
YProp="Oppo_Forecast";
SQLText='Select * from OpportunityProgress Where
'+'(Oppo_OpportunityId='+OppId+') and '
+'(Oppo_Forecast is not null) and '
+'(Oppo_Forecast>0)';
}
```

6. Write the chart to the screen by executing the Chart Block:

```
CRM.AddContent(chart.Execute());
Response.Write(CRM.GetPage());
```

To view the results, select an opportunity, click the chart tab and a chart of the forecast for that opportunity is displayed.

The Chart.asp file is displayed below:

```
<!-- #include file ="sagecrm.js" -->
<BODY>
<%
/* Find the current opportunity id in CRM and store it as
text to be included in sql queries for the
charts */
var
OppId=CRM.GetContextInfo('opportunity','oppo_opportunity
id');
//The chart variable contains the block to be retrieved.
var chart = CRM.GetBlock('chart');
//The chart commands establish the style of the chart
with (chart)
{
Stylename('Area');
Title='Forecast of Opportunity Over Time';
Description='Forecast of Opportunity over time';
Resize(600,600);
ShowLegend(false);
MinY=0;
LabelX='Date';
LabelY='Forecast';
XProp="Oppo_Forecast";
XLProp="Oppo_CreatedDate";
YProp="Oppo_Forecast";
/* Data represented in the chart is created through SQL.
The SQLText command specifies the records to be used.
The example states that we wish to look at all opportunity
progress tracking records for the current chart.*/
SQLText='Select * from OpportunityProgress Where '
+'(Oppo_OpportunityId='+OppId+') and '
+'(Oppo_Forecast is not null) and '+'(Oppo_Forecast>0)';
}
CRM.AddContent(chart.Execute());
Response.Write(CRM.GetPage());%>
</BODY>
</HTML>
```

Adding a Fusion Chart in OnCreate script

For example:

```
strChart = "<script type='text/javascript'
src='../FusionCharts/FusionCharts.js'></script>";
strChart += "<div id='chartContainer'>FusionCharts will load here!</div>";
strChart += "<script type='text/javascript'>";
strChart += "var myChart = new FusionCharts( '../FusionCharts/Column3D.swf',
'myChartId', '400', '300', '0', '1' );";
strChart += "myChart.setDataURL('../CustomPages/Data.asp');";
strChart += "myChart.render('chartContainer');</script>";
CRM.AddContent(strChart);
```

Adding a Fusion Chart in Custom Content

For example:

```
<script type="text/javascript" src="../FusionCharts/FusionCharts.js">
</script>
```

```
<div id="chartContainer">FusionCharts will load here!</div>
<script type="text/javascript">
var myChart = new FusionCharts( "../FusionCharts/Column3D.swf", "myChartId",
"400", "300", "0", "1" );
myChart.setDataURL("../CustomPages/Data.asp");
myChart.render("chartContainer");
</script>
```

Example: Organization Chart

The Organizational Charting Block, or 'Org Chart', can be used to create a relationship diagram through ASP. This block is a descendant of the Graphic Block and inherits and builds upon all the properties and methods available. Organizational charts can be customized in terms of appearance and the data they represent. They can also be hyperlinked-URLs can be specified through ASP to determine what should happen if a user clicks on a particular entry. This allows greater interaction with the user and provides useful links to whatever the various entries are representing.

You can also use the Related Entities feature to set up and view complex relationship types between primary entities. See the System Administrator Guide for more information on Related Entities.

Adding a Diagram to Show Relationships Between Companies

1. First you need to call an instruction to assign an org chart to a variable:

   ```
   var org;
   org=CRM.GetBlock('orgchart');
   ```

2. To demonstrate how to use the Org Chart Block effectively, assume that a relationship exists between a number of companies. Lets say that a corporation known as "ABC International" has three subsidiaries-ABC Health, ABC Travel, and ABC Entertainment. In an organizational diagram, "ABC International" is at the top. To add the first entry, use the OrgTree command in the form of:

   ```
   OrgTree(
   'Add',
   '[ParentName],[Name],[Child=true/false],[Url],[Description],[Relationship]'
   );
   ```

3. ABC International is the first entry. It is not a child of anything else.

   ```
   org.OrgTree('Add',',ABC International,false');
   ```

4. This gives a basic entry. You may want to add more to it in terms of functionality. For example, if the user clicks on it, they are directed to the company Web page. To do this add the entry using:

   ```
   org.OrgTree('Add',',ABC International,false,www.abc.com');
   ```

5. You can now add the subsidiaries of this company. Use the same structure as before, however, tell the OrgChart Block that the next entries you are adding have a parent called 'ABC International'. You also need to say that these are children, as opposed to siblings, so that the block can then place the new entries beneath 'ABC International'. The URL for these entries can be left blank, but you need to add the relationship each one has with the parent.

```
org.OrgTree('Add','ABC International,ABC Health,
true,www.abc.com/health,,Subsidiary');

org.OrgTree('Add','ABC International,ABC Travel, true,
www.abc.com/travel,,Subsidiary ');
org.OrgTree('Add','ABC International,ABC Entertainment,
true,www.abc.com/entertainment,Subsidiary');
```

6. Now call the block:

```
Response.Write(org.Execute());
```

7. The default styles used for the org chart are rounded boxes and arrows that show the connection between different entries. The arrows do not effectively show the relationship between the various entries, so you need to change the line style to a ray. Different relationships produces different ray colors and hovering over the ray gives a relationship description. You can also change the style of boxes to be square at this point. You do this by adding the following line before the Execute statement is called:

```
OrgTree('LineStyle','Ray');
OrgTree('BoxStyle','Square');
```

The OrgChart.asp file is displayed below:

```
<HTML>
<!-- #include file ="sagecrm.js" -->
<BODY>
<% var org;
org=CRM.GetBlock('orgchart');
org.OrgTree('Add',',ABC International,false,www.abc.com');
org.OrgTree('Add',
'ABC International,ABC Health, true,www.abc.com/
health,,Subsidiary');
org.OrgTree('Add',
'ABC International,ABC Travel, true,www.abc.com/
travel,,Subsidiary ');
org.OrgTree('Add',
'ABC International,ABC Entertainment, true, ' +
'www.abc.com/entertainment,,Subsidiary ');
org.OrgTree('LineStyle','Ray');
org.OrgTree('BoxStyle','Square');
CRM.AddContent(org.Execute());
Response.Write(CRM.GetPage());%>
<br>
</BODY>
</HTML>
```

Setting Parameters for the Organizational Graphic

Backgrounds and Icons

Icons and different backgrounds for the entities can be employed. Icons are referred to by the parameter 'EntityIcon' and backgrounds can be changed with the parameter 'EntityImage'. These can be in any of the formats that can be handled by the Graphic Block. This can be done by pointing to the image you wish to use in the following ways:

```
OrgTree('EntityIcon','c:\\entityimage.bmp');
OrgTree('EntityImage','c:\\back.jpg')
```

Title

By default, no title is added. A title can be added using the following:

```
org.OrgTree('Title','Structure of ABC International');
```

Dimensions

Various dimensions for the elements within the Organizational Block can help determine its size and appearance. The Full Box Height and Width describes the number of pixels, including outside the box, that each entity occupies. The standard box width and height properties describe only the dimensions of the box itself without considering margins or the outside area. Changing these values can determine the entire overall screen estate that the organizational chart occupies. These parameters can be set as follows:

```
OrgTree('FullBoxHeight','50'); //Default 100
OrgTree('FullBoxWidth','88'); //Default 200
OrgTree('BoxWidth','40'); //Default 75
OrgTree('BoxHeight','25'); //Default 40
```

Line and Box Styles

By default, the boxes are rounded and an arrow is used as the line style. However, this can be easily changed. The box style can either be 'Square' or 'Rounded'. The line style can be 'Ray', 'Line' or 'Arrow'. This can be done using the following commands:

```
OrgTree('LineStyle','Arrow');
OrgTree('LineStyle','Line');
OrgTree('LineStyle','Ray');
OrgTree('BoxStyle','Square');
OrgTree('BoxStyle','Round');
```

Animation

By default, the organizational chart uses animation at a delay of 30 ms and draws out each of the nodes in turn. This can be switched off so that all the nodes appear at the same time. Note also that the organizational chart as a graphic has access to the Animation command to cater for delay and loop parameters. Animation can be toggled in the following way:

```
OrgTree('Animation','false');
```

Legend

With the new ray line style indicating the relationship between two entries, a legend may appear to show what the colors represent. This can be switched off using:

```
OrgTree('ShowLegend','false');
```

Graphics Overview

Graphics can be generated dynamically within CRM. They can also be data aware and therefore representative of data stored at the time the graphic is generated.

A graphic can be customized to vary depending on the user that displays it, with little or no changes made by the author of the ASP.

Graphics can be generated with just a few commands, or customized at length. See Example: Adding a New Graphic (page 7-10) for an example of an ASP page for dynamically generating a graphic.

Graphics Formats

The CRM Graphic Block currently supports the following different graphics formats:

Graphic	Format Description	Characteristics
JPEG	Work of the Joint Photographic Experts Group	16 million colors (24 bit). High level of compression (small file size). Default format used for graphics and charts.
GIF	Graphics Interchange Format	256 colors (8 bit). High level of compression (small file size). Animation and Transparency supported.
BMP	Bitmap as commonly used in Microsoft Windows	Various color depths. No compression (large file size).

The characteristics of these formats are particularly important as they can affect the image quality of your graphic or chart.

By default, the JPEG image format is used.

JPEG vs GIF Images

```
var graphic;
graphic=CRM.GetBlock('graphic');
graphic.SaveAsGifs=true;
```

Where animation and transparency are not required, JPEG images are recommended instead of GIFs as they allow your image to contain a much greater color depth. Fusion charts are rendered as jpeg in ASP and .NET, they are rendered as Flash in the classic and interactive dashboard.

Graphics Performance Tips

Avoid Large Images

Windows has limitations in handling very large bitmaps and the "breaking point" differs considerably among various PCs. Windows and the video device driver cause the limitation.

Use GIF Images Where Possible

Windows has limitations in handling very large bitmaps and the "breaking point" differs considerably among various PCs. Windows and the video device driver cause the limitation.

```
<SCRIPT language=Javascript>
if (screen.colorDepth<=8) {graphic.SaveAsGifs=true}
</SCRIPT>
```

Be Conservative with Animations

On a client machine, large animations can consume a lot of processor time. Imagine a 50-frame animated GIF that is 250 x 250 pixels using 24-bit color. When decompressed, all 50 frames requires roughly 9MB of data if stored in memory simultaneously (250 * 250 * (24/8) * 50). If the GIF is animated over three seconds, for example, that is 3M/second of data, which requires a lot of processor power. When an animated GIF is looping, the browser does not simply repeatedly display image frames that were decoded in earlier iterations. This requires 9MB of data to be kept in memory. Instead, the image is continually re-decoded from a copy of the animation that is stored on disk. Therefore processor power is used instead of consuming large amounts of memory.

External Images

Images can be saved and loaded from the server. For Graphic Blocks, they may be used to generate part of the image.

Note that the Graphic Block always converts any loaded image to a JPEG or GIF image, as these are the standard types supported by Web browsers.

Graphics Examples

Example: Adding a New Graphic

To demonstrate how to use the CRM Graphic Block effectively, follow this example on how to create a progress bar. The progress bar illustrates Opportunity Certainty for the current opportunity.

To create a graphic for opportunity certainty, you need to:

- Add a new tab that links to a custom page.
- Create the custom page to display the graphic of the opportunity certainty for the current opportunity.

To create the custom page:

1. Retrieve the identifying value for the current Opportunity Certainty and assign it to a variable:
   ```
   var Progress=CRM.GetContextInfo('opportunity','oppo_certainty');
   ```
2. Create a block for the graphic and assign it to a variable:
   ```
   var progressbar; progressbar=CRM.GetBlock('graphic');
   ```
3. Issue commands to the progress bar to define the dimensions and style of the graphic (see the Progress.asp example below).
4. Write the graphic to the screen by executing the Graphic Block.
   ```
   CRM.AddContent(progressbar.Execute());
   Response.Write(CRM.GetPage());
   ```
   To view the results, go to an opportunity, click the new tab you created.

The following script is the complete Progress.asp file:

```
<!-- #include file ="sagecrm.js" -->
<HTML>
<BODY>
<%var progress=CRM.GetContextInfo('opportunity','oppo_certainty'); var progressbar;
progressbar=CRM.GetBlock('graphic');
with (progressbar)
{
ImageWidth=100;
ImageHeight=20;
Description='Opportunity Certainty';
GradientFill('Blue','White','L',256);
MoveTo(0,0);
LineTo(99,0);
LineTo(99,19);
LineTo(0,19);
LineTo(0,0);
Rectangle(0,0,100,20);
```

```
TextOut(40,1,progress+'%',true);
}
CRM.AddContent(progressbar.Execute());
Response.Write(CRM.GetPage());
%>
</BODY>
</HTML>
```

Example: Pipeline

The Pipeline Block is an extension of the Graphic Block. The pipeline graphic can be used to represent data over a chosen cross section. This block is a descendant of the Graphic Block and inherits and builds upon all the properties and methods available.

The Pipeline graphic block automatically displays in the Opportunities, Cases and Leads list screens in the context of My CRM, Company, People, Opportunity, Case, and Lead. When you click on a section of the pipeline, the relevant list is filtered to show only the section that you clicked on.

Adding a Pipeline to Show the Value of Opportunities

The following example displays the forecasted value of all opportunities for a company for all of the opportunities in their various stages. It acts as an instantly recognizable barometer of the data that it represents.

Within the ASP you must inform the pipeline of the user's context. It also needs a field to uniquely identify the current company.

To create a pipeline for opportunity forecast, you need to:

- Add a new tab to the company context that links to a custom page.
- Create the custom page to display the pipeline with the opportunity forecast for the current company.

To add a new tab to the company context that links to a custom page:

1. Click the **Administration** button in the main menu.
2. Select **Customization** from the context area of the screen
3. Select the **Company** context.
4. Select **Customize Tabs** and click the hyperlink for the Company tabs.
5. Add a name for the new tab in the Captions field.
6. In the Action field select **customfile** from the list.
7. In the Custom File field enter the name of the ASP file.
8. Choose the **Update** button and then the **Save** button

To add a new tab to the company context that links to a custom page:

1. Click the **Administration** button in the main menu.
2. Select **Customization** from the context area of the screen
3. Select the **Company** context.
4. Select **Customize Tabs** and click the hyperlink for the Company tabs.
5. Add a name for the new tab in the Captions field.
6. In the Action field select **customfile** from the list.
7. In the Custom File field enter the name of the ASP file.
8. Choose the **Update** button and then the **Save** button.

To create the custom page to display the chart with all the forecasts for the current opportunity:

1. Retrieve the identifying value for the current Opportunity and assign it to a variable:

```
var
CompId=CRM.GetContextInfo('company','comp_companyid')
;
```

2. Create a block for the pipeline and assign it to a variable:

```
var pipe=CRM.GetBlock('pipeline');
```

3. Using the Record object, retrieve details about the opportunities and add these to the pipe using the Pipeline. Entries are added to the pipe using the AddPipeEntry command using the following format:

```
AddPipeEntry([Name],[Value],[Description],[Url]);
```

4. Issue commands to the pipeline to define its data:

```
var SQLPipe='select sum(Oppo_Forecast) as a,'
+'Oppo_Stage from vOpportunity '
+'where (Oppo_PrimaryCompanyid='+CompId+') '
+'group by Oppo_Stage order by Oppo_Stage';
var Querypipe=CRM.CreateQueryObj(SQLPipe);
Querypipe.SelectSQL();
var pipe=CRM.GetBlock('pipeline');
var SQLPipe='select sum(Oppo_Forecast) as a,'
+'Oppo_Stage from vOpportunity '
+'where (Oppo_PrimaryCompanyid='+CompId+') '
+'group by Oppo_Stage order by Oppo_Stage';
var Querypipe=CRM.CreateQueryObj(SQLPipe);
Querypipe.SelectSQL();
var pipe=CRM.GetBlock('pipeline');
pipe.Pipe_Summary='<TABLE>'<TD
Class=TableHead>Qualified(70)'</TD>'</Table>';
Response.Write(pipe.Execute());
else {Response.Write(ErrMsg);}
```

5. Customize the look of the pipeline graphic.

```
with(SQLPipe)
{PipelineStyle('SelectedWidth','10');
PipelineStyle('SelectedHeight','10');
PipelineStyle('PipeWidth','10');
PipelineStyle('PipeHeight','60');
PipelineStyle('Margin','100');
PipelineStyle('Shape','Circle');
PipelineStyle('UseGradient','True');
PipelineStyle('Animated','False');
PipelineStyle('Selected','');
PipelineStyle('SelectedWidth','10');
PipelineStyle('SelectedHeigth','10');
PipelineStyle('ShowLegend','True');
PipelineStyle('ChooseBackGround', '5');
}
```

6. Save the ASP file with the same name as you used when creating the tab, to the Custom Pages folder of your CRM directory.
   SamplePipeline.asp is displayed below

```
<!-- #include file ="sagecrm.js" -->
<%=Body%>
<%var
CompId=CRM.GetContextInfo('company','comp_companyid');
var SQLPipe='select sum(Oppo_Forecast) as a,'
+'Oppo_Stage from vOpportunity '
+'where (Oppo_PrimaryCompanyid='+CompId+') '
+'group by Oppo_Stage order by Oppo_Stage';
var Querypipe=CRM.CreateQueryObj(SQLPipe);
Querypipe.SelectSQL();
var pipe=CRM.GetBlock('pipeline');
with(pipe)
{
PipelineStyle('SelectedWidth','10');
PipelineStyle('SelectedHeight','10');
PipelineStyle('PipeWidth','10');
PipelineStyle('PipeHeight','60');
PipelineStyle('Margin','100');
PipelineStyle('Shape','circle');
PipelineStyle('UseGradient','True');
PipelineStyle('Animated','False');
PipelineStyle('Selected','');
PipelineStyle('SelectedWidth','10');
PipelineStyle('SelectedHeigth','10');
PipelineStyle('ShowLegend','True');
PipelineStyle('ChooseBackGround', '5');
}
while (!Querypipe.EOF)
{
Label=Querypipe('Oppo_Stage');
Value=Querypipe('a');
pipe.AddPipeEntry(Label,parseFloat(Value),Value+"");
Querypipe.Next();
}
pipe.ChooseBackGround(1);
// Setting the active section of the pipeline. This
can be altered to be variable controlled
pipe.Selected=2;
// The summary allows the addition of any desired
text in html format, allowing summary
// tables etc for the selected pipe section. This
example shows a simple hard coded value.
pipe.Pipe_Summary='<TABLE><TD
Class=TableHead>Qualified(70)</TD></Table>';
CRM.AddContent(pipe.Execute());
Response.Write(CRM.GetPage());
//else {Response.Write(ErrMsg);}
%>
<br>
</BODY>
</HTML>
```

Setting Parameters for the Pipeline Graphic

Dimensions and Sizes

```
'SelectedWidth','10');
PipelineStyle('SelectedHeight','10');
PipelineStyle('PipeWidth','40');
PipelineStyle('PipeHeight','60');
PipelineStyle('Margin','100');
```

Shape

```
PipelineStyle('Shape','Circle');
PipelineStyle('Shape','Rounded');
PipelineStyle('Shape','Rectangle');
```

Other Appearance Changes

```
PipelineStyle('UseGradient','True');
PipelineStyle('Animated','True');
PipelineStyle('Selected','Sold');
PipelineStyle('Selected','');
PipelineStyle('ShowLegend','False');
```

Graphic Effects Basics

Change Image Color

There is also the facility to change one particular color to another in an image. For instance, all instances of the color blue could be changed to red using the following command:

```
Effect('ChangeColor','Blue,Red');
```

Clear An Image

To clear an image completely, and for example wash it with a particular color, the Clear effect is available. If no color is specified, then the canvas is cleared as white.

```
Effect('Clear','Blue');
```

Display Errors

When CRM catches an error, it produces a graphic by default that lists that error and other errors. This can be switched off using:

```
Effect('DisplayErrors','false');
```

Drawing Functions

There is a wide variety of commands available to the Graphic Block.

Pen

The pen function details the style behind which the drawings are carried out. Commands such as LineTo and Arc use the current pen style to determine their output.

The functions available through the Pen command include:

Color

The color of the pen can be specified using the following syntax:

```
Pen('Color','Blue');
PenColor('Blue');
```

Width

The pen width specifies the thickness of the pen in drawing actions and takes a number as its value parameter.

```
Pen('Width','3');
PenWidth('3')
```

Style

The Style command can be used to draw a dotted or dashed line, or to omit the line that appears as a frame around shapes. There are various values that Style can take:

Value	Line style
Solid	Solid (Default)
Dash	Dashes
Dot	Dots
DashDot	Alternating dashes and dots.
DashDotDot	Series of dash-dot-dot combinations.
Clear	No line is drawn (can be used to omit the line around shapes that draw an outline using the current pen).

Examples include:

```
Pen('Style','Dot');
Pen('Style','Solid');
Pen('Style','Clear');
```

Brush

The brush function is used to fill solid shapes, such as rectangles and ellipses, with a color or pattern. The pattern may be a predefined image, which is loaded into the brush. The functions available through the 'Pen' command include:

- **Color**. The color of the brush can be specified using the following syntax:

  ```
  Brush('Color','Blue');
  ```

- **Load**. An image can be loaded into the brush and used in all painting effects. The image format that can be loaded can be any one of those supported for import by the Graphic Block-namely .ico, .emf/.wmf, .bmp, .gif and .jpg. This is done using the following syntax:

  ```
  Brush('Load','c:\\winnt\\winnt.bmp');
  ```

- **Fill**. Carries out a fill command in the graphic using the current brush. It takes four parameters in the form of a rectangle to specify the area to be filled in. These are Left, Top, Right, and Bottom respectively. For example:

  ```
  Brush('Fill','0,0,100,100');
  ```

- **Style**. Allows the use of a number of predefined brushes for filling in drawing functions. These are:

- Bdiagonal
- Clear
- Cross
- DiagCross
- Fdiagonal
- Horizontal
- Solid
- Vertical

For example:

```
Brush('Style','DiagCross');
```

Font

The font function determines the way TextOut and TextOutCenter commands are performed. Modes available include:

- **Name** Changes the current typeface. Be sure that the font is installed on the server. Note that Truetype fonts prove more effective when performing the rotate font function.

  ```
  Font('Name','Times New Roman');
  ```

- **Size** Determines the size of font to be used.

  ```
  Font('Size','24');
  FontSize('24');
  ```

- **Color** Specifies the color of writing to use.

  ```
  Font('Color','Blue');
  FontColor('Blue');
  ```

- **Styles available** The following styles can be toggled in the following ways:

  ```
  Font('Underline','True');
  Font('Italic','False');
  Font('Strikeout','False');
  ```

- **Rotation** A rotation effect can be applied to all text output. To ensure success, use True type fonts. Simply specify Rotate with the desired angle and all TextOut and TextOutCenter commands now show the font in a rotated form.

  ```
  Font('Rotate','45');
  ```

Merging

An external image can be merged onto a graphic. A color is also passed as the transparent color for the external image. The position of this external image can be specified with X and Y parameters. Otherwise it simply appears starting from 0,0 in the top left hand corner of your image.

```
Effect('Merge','c:\\Person.ico');
Effect('Merge','c:\\Person.ico,50,50');
```

Special Effects

Through the use of the graphics command, various effects can be applied to a graphic object. These include dithering, zooming, and transparency.

- **Dithering** There are six different dithering functions that can be applied to an image. They can help to improve its appearance, especially where color is limited. These established modes are
  - Burkes
  - FloydSteinberg
  - JaJuNi
  - Sierra
  - SteveArche
  - Stucki

  A command such as the following is be used to apply one of these modes to an image:

  ```
  Effect('Dither','FloydSteinberg');
  ```

- **Zooming** An image can be magnified using the zoom parameter together with a percentage of zoom required. The area to be zoomed is, by default, the center of the image.

  ```
  Effect('Zoom','200');
  ```

- **Transparency** Available only in GIF images, enabling transparency causes any whiteness contained within an image to become transparent. On a Web browser, this shows any applied image to that area. Transparency can be toggled in the following manner:

  ```
  Effect('Transparency','true');
  ```

Animation

Adding Frames

This allows the state of your current graphic to be the next frame in your animation. If no frames have been added previously, this is the first frame in the animation. The second parameter specifies the time period for which this frame is shown.

```
Animation('Add','50');
```

Leaving the delay parameter blank results in the default delay being used. For example:

```
Animation('Add','');
```

Delay

To specify the default delay for frames, use the delay procedure with the desired value.

```
Animation('Delay','50');
```

Loops

An animation can be looped a specific number of times or indefinitely. By default, an animation is shown once (1). To have an animation repeat indefinitely, specify the value 0.

```
Animation('Loop','0');
```

ASP Example

This is a sample animation.asp file:

```
<!-- #include file ="sagecrm.js"-->

<% var anim;
var progress=70;

anim=CRM.GetBlock('graphic');

with (anim) {

ImageWidth=130;

ImageHeight=20;

Pen('Color','Black');

MoveTo(0,0);

LineTo(99,0);

LineTo(99,19);

LineTo(0,19);

LineTo(0,0);

Rectangle(0,0,100,20);

Pen('Color','Blue');

for (y=1;y<=progress;y++)

{MoveTo(y,1);

LineTo(y,19);

TextOutCenter(101,0,129,19,y+'%',false,false);

Animation('add','10')}

}

container=CRM.GetBlock('container');

with (container)

{AddBlock(anim);

displaybutton(Button_Default)=false;}

CRM.AddContent(container.Execute());

Response.Write(CRM.GetPage());

%>
```

Chapter 8: ASP Object Reference

In this chapter you will learn how to work with:

- AddressList Object
- Attachment Object
- AttachmentList Object
- CRM Object
- CRMBase Object
- CRMBlock Object
- CRMChartGraphicBlock Object
- CRMContainerBlock Object
- CRMContentBlock
- CRMEntryBlock Object
- CRMEntryGroupBlock Object
- CRMFileBlock Object
- CRMGraphicBlock Object
- GridColBlockObject
- CRMListBlock Object
- CRMMarqueeBlock Object
- CRMMessageBlock Object
- CRMOrgGraphicBlock Object
- CRMPipelineGraphicBlock Object
- CRMQuery Object
- CRMRecord Object
- CRMSelfService Object
- CRMTargetListField Object
- CRMTargetListFields Object
- CRMTargetLists Object
- Email Object
- MailAddress Object
- MsgHandler Object

Introduction to the ASP Object Reference

This section details the properties and methods of the CRM objects and blocks.

There is a quick reference table here.

Note that in examples throughout this Developer Guide, the CRM object is referred to as 'CRM'. In some older versions of the include file, the CRM object may be referred to as "eWare".

For example, if this returns an error:

```
CRM.Mode=Save;
```

then you can replace it with

```
eWare.Mode=Save;
```

Quick Reference Table

Object/Block	Property	Method

CRM Object	Mode	AddContent(Content)
		CreateQueryObj(SQL, Database)
		CreateRecord(TableName)
		FindRecord(TableName, QueryString)
		GetBlock(BlockName)
		GetCustomEntityTopFrame(EntityName)
		GetPage()
		GetTrans(Family, Caption)
		RefreshMetaData(Family)
		SetContext(EntityName, EntityID)

CRMBase Object	FastLogon
	TargetLists
		Button
		GetContextInfo(Context, FieldName)
		GetTabs(TabGroup)
		Logon(LogonId, Password)
		Url(Action)
		ConvertValue(Avalue, AfromCurr, AToCurr)

CRMTargetLists Object	TargetListID
	Category
	Name
	Description
	ViewName
	Fields
	OrderByFields
	WhereClause
		Save()
		Include(ATargetID)
		Exclude(ATargetID)
		Retrieve()

CRMTargetListFields Object	Parent
	Count
	Item
		New(CRMTargetListField)
		Delete(Index)

CRMTargetListField Object	DataField	

CRMSelfService Object	Authenticated
	AuthenticationError
	VisitorInfo
		EndSSSession(QueryString, ContentString, Cookie)
		Init(QueryString, ContentString, Cookie)

Email Object	Body
	IsHTML
	Subject
	Priority
	Recipients
	SenderName
	SenderAddress
	DeliveryTime
	Attachments
	BCC
	CC
		Send()
		AddFile(Value)
		Clear()
		Header(Value)

AddressList Object	Items Count	AddAddress(Address, Name)
MailAddress Object	Name Address	
AttachmentList Object	Count Items LibraryPath	

Object/Block	Property	Method

Attachment Object	Name
	Extension
		Save(Name, Path)
		SaveAs(Name, Path)

CRMRecord Object	DeleteRecord
	Eof
	IdField
	Item
	ItemAsString
	OrderBy
	RecordCount
	RecordID
		FirstRecord()
		NextRecord()
		RecordLock
		SaveChanges()
		SaveChangesNoTLS()
		SetWorkflowInfo(vWorkflowName, vWorkflowState)

MsgHandler Object	Msg
	Debug
	EmailAddress
		Log(value)
		MailAdmin(Subject, Body)
		GetUniqueFileName(Path, FileName)

CRMQuery Object	Bof
	DatabaseName
	Eof
	FieldValue
	RecordCount
	SQL
		ExecSql()
		Next()
		NextRecord()
		Previous()
		SelectSql()

CRMBlock Object	ArgObj
	CheckLocks
	DisplayForm
	FormAction
	Height
	Name
	NewLine
	ShowValidationErrors
	Title
	ViewName
	Width
	Mode
		Execute(Arg)
		Validate()

CRMContainerBlock Object	ButtonAlignment
	ButtonImage
	ButtonLocation
	ButtonTitle
	DisplayButton
	ShowNewWorkflowButtons
	ShowWorkflowButtons
	WorkflowTable
		AddBlock(Block)
		AddButton(ButtonString)
		DeleteBlock(BlockName)
		GetBlock(BlockName)

CRMContentBlock	Contents	

CRMEntryGroupBlock Object	ShowSavedSearch	AddEntry(EntryName, Position, Newline)
		DeleteEntry(EntryName)
		GetEntry

Object/Block	Property	Method
CRMEntryBlock Object	AllowBlank Caption CaptionPos CreateScript DefaultType DefaultValue EntryType FAM FieldName Hidden JumpEntity LookUpFamily MaxLength MultipleSelect OnChangeScript ReadOnly Size ValidateScript AllowUnassigned Restrictor CopyErrorsToPageErrorContent	RemoveLookup

CRMListBlock Object	CaptionFamily
	PadBottom
	prevURL
	RowsPerScreen
	SelectSql
		AddGridCol(ColName, Position, AllowOrderBy)
		DeleteGridCol(ColName)
		Execute(Arg)
		GetGridCol

GridColBlockObject	Alignment AllowOrderBy CustomActionFile CustomIdField JumpEntity ShowHeading ShowSelectAsGif	
CRMMarqueeBlock Object	HorizontalMaximum HorizontalMinimum ScrollSpeed StyleSheet VerticalMaximum VerticalMinimum	
CRMFileBlock Object	DirectoryPath FileName ProperCase Translate	
CRMMessageBlock Object	DisplayForm mAddressFrom/mNameFrom mBody mErrorMessage mSentOK mShowCC/mShowBCC mSubject	

Object/Block	Property	Method

CRMGraphicBlock Object	Border
	Description
	hSpace
	ImageHeight
	ImageWidth
	SaveAsGIF(text)
	vSpace
		Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4)
		Animation(Mode, Value)
		Brush(Mode, Value)
		Chord(X1,Y1,X2,Y2,X3,Y3,X4,Y4)
		Effect(Mode, Value)
		Ellipse(X1,Y1,X2,Y2)
		FlipHoriz()
		FlipVert()
		Font(Mode, Value)
		FontColor(Color)
		FontSize(Size)
		GradientFill(StartColor, EndColor, Direction, Colors)
		GrayScale()
		LoadBMP(Filename)
		LoadImage(text)
		LoadJPG(Filename)
		LineTo(X,Y)
		Monochrome()
		MoveTo(X,Y)
		Pen(Mode, Value)
		PenColor(Color)
		PenWidth(Width)
		PieShape(X1,Y1,X2,Y2,X3,Y3,X4,Y4)
		Rectangle(X1,Y1,X2,Y2)
		Resize(Width, Height)
		Rotate(Number)
		RoundRect(X1,Y1,X2,Y2,X3,Y3)
		SaveAsGIF(text)
		SaveAsJPG(text)
		TextOut(X, Y, Text, transparent=True/False)
		TextOutCenter(Left, Top, Right, Bottom, Text, Transparent, Ellipse)

CRMChartGraphicBlock Object	LabelX
	LabelY
	SQLText=Text
	XLProp=text
	Xprop=text
	Yprop=text
		BackGradient(Visible, color1, color2)
		ChartTitle(text)
		ManualChartEntry(Value, MakeNull=true/false)
		ShowLegend(true/false)
		Stylename(Style)

CRMOrgGraphicBlock Object		OrgTree(Mode, Value)

CRMPipelineGraphicBlock Object	Pipe_Summary
	Selected
		AddPipeEntry(Name, Value, Description)
		ChooseBackGround(Value)
		PipelineStyle(Mode, Value)

Examples

The following sample ASP pages are covered in this section:

Sample CRMGridColBlock ASP page (page 8-7)
Sample CRMListBlock ASP page (page 8-7)
Using the New Workflow Properties in an ASP page (page 8-8)

Sample CRMGridColBlock ASP page

The following example illustrates how to use the GridColBlock object to add and remove columns in a list and to edit the properties of columns in a list. The page should be used from the Company tab group to show all the People for the current Company.

```
<!-- #include file ="sagecrm.js" -->
<%// Start with the Person List
PersonList=CRM.GetBlock("persongrid");
// Add the Person fax number as the 3rd column in the list,
with no heading
GridCol=PersonList.AddGridCol("pers_faxnumber",2);
GridCol.ShowHeading=false;
// Get the GridCol Block for the FirstName column
GridCol=PersonList.GetGridCol("pers_firstname");
GridCol.AllowOrderBy=true;
GridCol.ShowHeading=true;
// Set FirstName column to jump to another asp page
GridCol.JumpEntity="custom";
GridCol.CustomActionFile="myP.asp";
GridCol.CustomIdField="pers_personId";
// Remove the Company Name column from the List
PersonList.DeleteGridCol('comp_name');
PersonList.Title="Standard Person Grid, with some changes";
CompanyId=CRM.GetContextInfo('company','comp_companyid');
CRM.AddContent(PersonList.Execute('pers_companyid='+CompanyId));
Response.Write(CRM.GetPage());
%>
```

Sample CRMListBlock ASP page

This example creates a case list from the company context, where the two columns status and stage are removed if the user is not from the Sales Team. An extra column, 'FoundIn', is added if the user is from the Operations Team.

```
<!-- #include file ="sagecrm.js"-->
<%// get the current Company Id
ThisCompanyId =
CRM.GetContextInfo('Company','Comp_CompanyId');
// get a reference to the CaseListBlock
CaseListBlock = CRM.GetBlock('CaseListBlock');
// build the SQL WHERE clause
SearchSql = 'Case_PrimaryCompanyId='+ThisCompanyId +
" and Case_Status='In Progress' "
// check the users team Id
UserChannel =
CRM.GetContextInfo('User','User_PrimaryChannelId');
// remove fields if not Sales team
if (UserChannel != 1) { // where 1 is Channel Id of Sales
Team
CaseListBlock.DeleteGridCol('Case_Status');
CaseListBlock.DeleteGridCol('Case_Stage');}
// add field for Development Team
```

```
if (UserChannel == 5) {
// where 5 is Channel Id of Operations Team
FoundIn = CaseListBlock.AddGridCol('Case_FoundVer');
FoundIn.AllowOrderBy = true;
// allow sort by Found In column
}
// execute the Block, passing in the sql clause
CRM.AddContent(CaseListBlock.Execute(SearchSql));
Response.Write(CRM.GetPage());%>
<%=EndBody%:>
```

Using the New Workflow Properties in an ASP page

The ASP page specified in the Custom File Name property, must do the following:

- Use a Container Block (container, list, or entry group).
- Set the Container Block WorkflowTable property to the table name.
- Set the Container Block ShowWorkflowButtons property to 'true'.
- Pass in the Record object as the argument to the execute method of the container.

Example1- ShowWorkflowButtons

This file can be referred to as:

- The custom file name in the List to jump to. In this case it shows the edit screen for one record in the table and the workflow buttons for the current record.
- The custom file name in the Primary rule. In this case the page creates a new record in the workflow.

```
<!-- #include file ="sagecrm.js" -->;
<%
Response.Write(CRM.GetTabs());
ThisId=Request.QueryString("Tab_TableId");
TableDetailBox=CRM.GetBlock("MyTableEntryBox");
Holder = CRM.GetBlock('container');
Holder.AddBlock(TableDetailBox);
with (TableDetailBox) {
// turn on the delete button
DisplayButton(Button_Delete)=true;
// turn on continue button - goes back to list
DisplayButton(Button_Continue)=true;
Title = "My Table Details";}
// if the id was NOT passed in then this is NEW mode
if (!Defined(ThisId)) {
MyRecord=CRM.CreateRecord("MyTable");
if (CRM.Mode &lt;= Edit)
CRM.Mode=Edit;}
else {
MyRecord=CRM.FindRecord("MyTable","Tab_TableId="+This
Id);
}
Holder.ShowWorkflowButtons = true;
Holder.WorkflowTable = 'MyTable';
CRM.AddContent(Holder.Execute(MyRecord));
Response.Write(CRM.GetPage());%>
```

Example 2- WorkflowTable and ShowNewWorkflowButtons

The following sample ASP page demonstrates the new Container Block properties- WorkflowTable and ShowNewWorkflowButtons. This page displays a list of records and buttons for any Primary rules on the workflow for MyTable if the primary rules are configured to use a custom file.

```
<!-- #include file ="sagecrm.js" -->
<%
Response.Write(CRM.GetTabs());
MyList=CRM.GetBlock("MyTableList");
// to show the button on the right, put list into a
container and add button to the container
Holder = CRM.GetBlock("container");
// add the list to the container
Holder.AddBlock(MyList);
// dont show the default edit/save button on the
container
Holder.DisplayButton(Button_Default) = false;
// tell the list which records to show
MyList.ArgObj = ''; //show all records!
// Make sure the Primary rules workflow 'new' buttons
show
Holder.WorkflowTable = 'MyTable';
Holder.ShowNewWorkflowButtons = true;
// display the container
CRM.AddContent(Holder.Execute(''));
Response.Write(CRM.GetPage());%>
```

AddressList Object

The AddressList Object is used to customize the scripts deployed by E-mail Management. Part of the Email Object, this object provides access to the To, CC and BCC lists of addresses. You can access this object as follows:

```
myaddresslist = eMail.CC.
```

Please refer to the *System Administrator Guide* for more information on E-mail Management.

Methods

AddAddress(Address, Name)

Description	This adds an address to the list. You generally use this function when preparing to send a new e-mail.
Parameters	Address: This is a valid e-mail address. Name: This is the user friendly name associated with the e-mail address.
Example	`email.Recipients.AddAddress("test@domain.com", "test user");`

Properties

Items

Description	Returns a Mail Address Object, which gives access to e-mail addresses and names.
Parameters	Integer. The index of the address.
Example	`emailAddress= email.Recipients.Items(1).Address;`

Count

Description	Returns a count of the number of e-mail addresses in the list.
Values	Integer (read only)

Example	`if (email.Recipients.Count==1) {`
	`    //do something`
	` }`

Attachment Object

The Attachment Object is used to customize the scripts deployed by E-mail Management. This object provides access to an individual attachment. You use the AttachmentList Object's "items" property to access this object: myAttachment = eMail.Attachments.Items(1);

Please refer to the *System Administrator Guide* for more information on E-mail Management.

Methods

Save(Name, Path)

Description	This saves the attachment to a specified folder. The return value is Boolean. True if the save succeeds and False if not.
Parameters	Name: This is the name that the attachment is to be saved as. The parameter is passed by reference and may be returned with a different value than the one sent in. Path: The full physical name of the folder where the attachment should be saved.
Example	`AttItem.Save("my file.txt", "D:\Program Files\Sage\CRM\<myinstallname>\Library");`

SaveAs(Name, Path)

Description	This saves the attachment to the specified folder as a file with the specified name. The return value is Boolean. True if the save succeeds and False if not.
Values	Name: This is the name that the attachment is to be saved as. Path: The full physical name of the folder where the attachment should be saved.
Example	`AttItem.SaveAs("my file.txt", "D:\Program Files\Sage\CRM\<myinstallname>\Library")`

Properties

Name

Description	This is the Name of the attachment Item. Usually this is the name of the actual file that was attached.
Values	String (read/write)
Example	`NewName = AttItem.Name + "test" + AttItem.Extension;`

Extension

Description	This is the extension of the file attachment.
Values	String (read only)
Example	`NewName = AttItem.Name + "test" + AttItem.Extension;`

AttachmentList Object

The AttachmentList Object is used to customize the scripts deployed by E-mail Management. This object provides access to the e-mail attachments. You can access this object as follows myAttachmentList = eMail.Attachments;

Please refer to the *System Administrator Guide* for more information on E-mail Management.

Properties

Count

Description	This is a count of the number of attachments in the list.
Values	Integer (read only)

Example	`for (i = 0; i < Attachments.Count; i++) {`
	`//do something`
	`}`

Items

Description	This returns an Attachment Object.
Values	Integer. The index of the attachment.

Example	`for (i = 0; i < Attachments.Count; i++)`
	`{`
	`AttItem = Attachments.Items(i);`
	`//do something with the attachment item`
	`}`

LibraryPath

Description	This is the path to the CRM library. It assumes that the library exists in a subdirectory of the CRM install directory called "library". This value can be overwritten if required.
Values	String
Example	`var libdir = Attachments.LibraryPath + "\\" + CompanyQuery("comp_name");`

Email Object

The Email Object is used to customize the scripts deployed by E-mail Management. It provides access to the e-mail itself through its properties and methods. This object is passed into the script by default as the Email Object but can also be accessed from the MsgHandler Object as follows: myemail = MsgHandler.msg.

Please refer to the *System Administrator Guide* for more information on E-mail Management.

Methods

Send()

Description	This sends an e-mail using the contents of the Email Object.
Parameters	None
Example	`email.Send();`

AddFile(Value)

Description	This adds a file attachment to the Email Object.
Parameters	Value: The full physical path to the file that is to be attached. It returns True or False depending on whether the file exists or not
Example	`email.AddFile('C:\Temp\report.doc')`

Clear()

Description	This clears the contents of the Email Object. You typically use this before you want to send a new e-mail.
Parameters	None
Example	`email.Clear();`

Header(Value)

Description	Returns any named header value from the e-mail. If the header value does not exist a blank string is returned
Parameters	Value: The name of the header value that you wish to retrieve
Example	`comm("comm_replyto") = email.Header("Reply_To");`

Properties

Body

Description	This is the body text of the e-mail.
Values	String (read/write)
Example	`comm("Comm_Email") = eMail.Body`

IsHTML

Description	This is used when sending an e-mail only. If set to true the e-mail is sent as an html e-mail.
Values	Boolean (read/write)
Example	`eMail.IsHTML = true;`

Subject

Description	This is the subject text of the e-mail.
Values	String; (read/write)
Example	`comm("Comm_Note") = eMail.Subject;`

Priority

Description	This is the priority of the e-mail. Its values are Low (0), Normal (1), and High (2)
Values	Integer (read/write)

Example	`var prHigh=2;`
	`eMail.Priority=prHigh;`

Recipients

Description	This returns an AddressList Object, which holds all of the e-mail addresses that were in the To list on the e-mail.
Values	Returns an AddressList Object.

Example	`var MyAddressList;`
	`MyAddressList = email.Recipients;`
	`//now we get the first email address from //the list`
	`var singleaddress =`
	`MyAddressList.Items(0).Address;`

SenderName

Description	This is the Name of the person who sent the e-mail.
Values	String (read/write)

Example	`comm("comm_from") = "\""+eMail.SenderName + "\" " + "<" +`
	`eMail.SenderAddress + "> ";`

SenderAddress

Description	This is the e-mail address of the person who sent the e-mail.
Values	String

Example	`comm("comm_from") = "\""+eMail.SenderName + "\" " + "<" +`
	`eMail.SenderAddress + "> ";`

DeliveryTime

Description	The DeliveryTime requires the datetime that the e-mail was delivered to the inbox.
Values	None

Example	`commdate = new Date(eMail.DeliveryTime);`
	`comm("Comm_datetime") = commdate.getVarDate();`

Attachments

Description	This returns an AttachmentList Object, which holds a list of the attachments on the e-mail.
Value	Returns an AttachmentList Object.
Example	`var Attachments = email.Attachments;`

BCC

Description	This returns an AddressList Object, which holds all of the e-mail addresses that were in the BCC list on the e-mail.
Values	Returns an AddressList Object.
Example	`var singleaddress = email.BCC.Items(0).Address;`

CC

Description	This returns an AddressList Object, which holds all of the e-mail addresses that were in the CC list in the e-mail.
Values	Returns an AddressList Object.
Example	`var singleaddress = email.CC.Items(0).Address;`

MailAddress Object

The MailAddress Object is used to customize the scripts deployed by E-mail Management. This object provides access to an individual address from the AddressList Object. You can return an individual MailAddress object as follows:

```
myaddress = eMail.CC.Items(1);
```

Properties

Name

Description	This is the user friendly name associated with the e-mail address.
Values	String (read/write)
Example	`FromName=eMail.CC.Items(1).Name;`

Address

Description	This is the e-mail address for this item.
Values	String (read/write)
Example	`FromAddress=eMail.CC.Items(1).Address;`

MsgHandler Object

The MsgHandler Object is used to customize the scripts deployed by E-mail Management. It provides basic access to the Email Object and functionality for the system. It is the top level object within E-mail Management and scripting. It is passed into the script at run time. Please refer to the System Administrator Guide for more information on E-mail Management.

Methods

Log(value)

Description	If the debug option is turned on, messages are written to a log file in the CRM install directory, for example CRM.log.
Parameters	String: Log message.

Example	`MsgHandler.Debug = true;`
	`MsgHandler.Log("testing application");`

MailAdmin(Subject, Body)

Description	Sends an e-mail to the e-mail address specified in the EmSe_ AdminAddress in the custom_emailaddress table.
Parameters	Subject: The string value that appears in the subject of the e-mail. Body: The string value that appears in the body of the e-mail.

Example	`var sSubject = "Unknown customer";`
	`var sBody = "An unknown customer attempted to mail the service";`
	`MsgHandler.MailAdmin(sSubject,sBody);`

GetUniqueFileName(Path, FileName)

Description	Takes a file and pathname and checks if the file exists in the path already. If not, the file name is returned. If the file does exist, it returns the next valid name for the file.
Parameters	Path: This is the path where the location of the file is checked for. Filename: This is the file name that is used to check if it is used already.

Example	`NewName = MsgHandler.GetUniqueFileName(libdir, AttItem.Name);`
	`AttItem.SaveAs(NewName, libdir);`

Properties

Msg

Description	This returns the Email Object. Note that this should not be accessed from the script as the object has already been passed in as the Email Object.
Values	Returns the Email Object.
Example	`myemail=MsgHandler.msg`

Debug

Description	Use this to flag whether messages sent via the Log method are actually written to the log file.
Values	Boolean (read/write)
Example	`MsgHandler.Debug = true;`

EmailAddress

Description	This is the e-mail address of the service.
Values	String (read only)
Example	`var serviceaddress = MsgHandler.EmailAddress;`

CRM Object

The CRM Object provides basic access to CRM objects and functionality. You use the methods of this object to create new objects, get existing objects, and execute objects.

The CRM Object is the parent of the CRMBase Object and the CRMSelfService Object. The CRMBase Object contains all the CRM custom metadata, and the CRMSelfService Object enables authenticated and anonymous visitors to access varying levels of CRM data in View Only mode.

Methods

AddContent(Content)

Description	The Value in the Content parameter is added to the page in memory and is returned when you call GetPage(). This can also be used in a CreateScript to pass something that is only available server side, for example the current user's email address, see example 2 below.
Parameters	Content: String. The value in Content should be the result of a Block.Execute() method

Example	`MyList = CRM.GetBlock('PersonGrid');`
	`CRM.AddContent(MyList.Execute`
	`("pers_lastname like 'B%'"));`
	`Response.Write(CRM.GetPage());`

Example 2	*Please refer to Developer Help files for code sample*.

CreateQueryObj(SQL, Database)

Description	Creates a new query object from the system database or an external database connected to CRM.
Parameters	SQL: A valid SQL string. Database: The database to open. If this is left blank, the system default database is used.

Example	The following example creates a query object from the company view in the CRM (default) database.
	
	`Query= CRM.CreateQueryObj("Select * from vcompany");`
	`Query.SelectSql();`
	`CRM.AddContent (Query.FieldValue("comp_name"));`
	`while (!Query.eof)`
	`{`
	`CRM.AddContent (Query.FieldValue("comp_name") + '`
	`');`
	`Query.NextRecord();`
	`}`
	`Response.Write(CRM.GetPage());`

CreateRecord(TableName)

Description	Creates a new record object in a specified table. You must call the SaveChanges method to persist the record to the database. Note that SaveChanges is automatically called by the Execute method of most blocks when the mode is set to Save.
Parameters	TableName: The table that this record is created in. This table can be either a CRM table or one that has been added by creating an external table connection within CRM.

Example	The following example displays a record summary for the current person.
	
	Note that GetContextInfo is a method of the CRMBase object.
	
	`ThisPersonId = CRM.GetContextInfo('Person','Pers_PersonId');`
	`ThisPersonRecord =`
	`CRM.FindRecord('Person','Pers_Personid='+ThisPersonId);`
	`PersonBlock = CRM.GetBlock('PersonBoxLong');`
	`CRM.AddContent(PersonBlock.Execute(ThisPersonRecord));`
	`Response.Write(CRM.GetPage());`

FindRecord(TableName, QueryString)

Description	Finds an existing row on the given table and returns a record object.
Parameters	TableName: The table that this record is extracted from. This table can be a CRM table or an external table that has been added by creating an external table connection within CRM. QueryString: The SQL WHERE clause that identifies the record you are interested in.

Example	The following example displays a record summary for the current person.
	
	Note that GetContextInfo is a method of the CRMBase object.
	
	`ThisPersonId = CRM.GetContextInfo('Person','Pers_PersonId');`
	`ThisPersonRecord =`
	`CRM.FindRecord('Person','Pers_Personid='+ThisPersonId);`
	`PersonBlock = CRM.GetBlock('PersonBoxLong');`
	`CRM.AddContent(PersonBlock.Execute(ThisPersonRecord));`
	`Response.Write(CRM.GetPage());`

GetBlock(BlockName)

Description	Returns a CRM block. You use this method to call child blocks of the CRM Object. This is one of the most commonly used methods in CRM ASP generation. You retrieve a block, customize it, and display it.
Parameters	BlockName: Any valid block type, or any existing screen or list in CRM. For more information on block naming refer to Creating a New Block (page 4-18).

Example	The following example creates a marquee block:
	
	Marquee = CRM.GetBlock("marquee");
	The following example gets a container block, a search block, and a list block. It then adds the search screen and list block to the container and displays the container.
	
	Note that you can also set the execute statement to use the search results as the argument for the list.
	
	`Search=CRM.GetBlock("CompanySearchBox");`
	`List=CRM.GetBlock("CompanyGrid");`
	`Holder = CRM.GetBlock("Container");`
	`Holder.AddBlock(Search);`
	`Holder.AddBlock(List);`
	`CRM.Addcontent(Holder.Execute());`
	`Response.Write(CRM.GetPage());`

GetCustomEntityTopFrame(EntityName)

Description	Adds the top content for a custom entity including the icon, caption, and description.
Parameters	String: EntityName.
Example	`CRM.GetCustomEntityTopFrame("EntityName");`

GetPage()

Description	Returns the contents of the page, that has been previously added by AddContent. The contents are in the syntax used by the current device. As GetPage() includes the GetTabs() method, TabGroupName can be passed into this method. If you pass it in, it will show those tabs instead of the current default tabs.
Parameters	TabGroupName(optional, see above).

Example	`MyList = CRM.GetBlock('PersonGrid');`
	`CRM.AddContent(MyList.Execute("Pers_lastname like 'B%'"));`
	`Response.Write(CRM.GetPage());`

GetTrans(Family, Caption)

Description	Returns the translation for a given caption in the given caption family based on the user's current language. For self service users, you need to preset the language using the VisitorInfo method for this to work. CRM.VisitorInfo("Visi_Language")='DE';
Parameters	Family: The name of the family of captions to which the caption relates. You can view all the caption types and family names in Administration | Customization | Translations. Caption: The caption name.

Example	The following example displays hello in German, when the user's current language is set to German, assuming the translation has already been entered in CRM translations.
	`CRM.AddContent(`
	`CRM.GetTrans('GenCaptions','Hello'));`
	`Response.Write(CRM.GetPage());`

RefreshMetaData(Family)

Description	If data in the Custom_Captions table is explicitly edited or new records are added, you use this method to refresh CRM's internal cache with the new information.
Parameters	Family: The translation family (Capt_Family) to update.

Example	The following example adds a new selection entry called 'Open' to the custom caption's comp_status selection.
	
	You use the CreateRecord method to create a record in the Custom_ Captions table. You add entries to relevant fields in that table, save the changes to the database and then use the RefreshMetaData method to refresh the cache.
	
	`NewCaption=CRM.CreateRecord("Custom_Captions");`
	`NewCaption.Capt_FamilyType="Choices";`
	`NewCaption.Capt_Family="Comp_Status";`
	`NewCaption.Capt_Code="Open";`
	`NewCaption.Capt_US="Open";`
	`NewCaption.SaveChanges();`
	`CRM.RefreshMetaData("Comp_Status");`

SetContext(EntityName, EntityID)

Description	Updates the recent list for the specified entity
Parameters	EntityName: The name of the custom entity. EntityId: The custom entity id as it appears in custom_tables.
Example	`CRM.SetContext(EntityName,Id);`

Properties

Mode

Description	Determines how some of the blocks display. Be careful when toggling modes. If the conditionals are not very precise, you can end up locking down a screen.
Values	0=View, 1=Edit, 2=Save

Example	Example 1: The following example changes the CRM mode to Edit if it is in View mode.
	`if (CRM.Mode < Edit) {CRM.Mode=Edit; }`
	Example 2: The following example changes the CRM mode to Save if there is an error, and displays the error.
	`CRM.Mode=Edit;`
	`if (error!="")`
	`{`
	`CRM.Mode=Save;`
	`CRM.AddContent(error);`
	`Response.Write(CRM.GetPage());`
	`}`

CRMBase Object

The CRMBase object provides functionality that is only applicable in the CRM environment, such as the company currently being viewed. This object is often used to set up the context information for the current view and display the tabs for that view.

Methods

Button

Description	Returns the text, image, and link for a CRM button. These buttons are typically the generic buttons that appear on screens and containers. This is useful for adding buttons to screens to link to other Web sites or ASP pages.
Parameters	Caption: The caption for the button. This is translated based on the user's language (providing there is a matching entry in CRM Translations). ImageName: The image you want displayed on the button. All images need to be stored in the CRM directory 'Img' folder. URL: The URL that this button links to. This can either be a link to a Web address or to a custom page in the CRM custom pages folder. (See URL below). PermissionsEntity and PermissionsType: If you want the button to be added based on a users security profile for an entity, use the PermissionsEntity and PermissionsType parameters. PermissionsEntity is the name of the entity. PermissionsType is either VIEW, EDIT, DELETE or INSERT, depending on the action of the button. Target: This allows the TARGET property of the button's anchor () tag to be set.

Example	The following example displays a button called 'test' that includes an image called 'test.gif' and links to an ASP page (test.asp).
	
	`CRM.AddContent(CRM.Button("test",`
	`"test.gif", CRM.Url("test.asp")));`
	`Response.Write(CRM.GetPage());`

GetContextInfo(Context, FieldName)

Description	Returns the named field from the given table based on the current context. You can also use this method to return the RecordID for a given table which you can use to build an SQL query (for example to create charts).
Parameters	Context: The context from which you want to extract the field. Options are: • Person, Company, Opportunity, Lead, Case, Solution • Channels (teams) • Campaigns, Waves, Wave Items • SelectedUser (SelectedUser is applicable when viewing the My CRM list) • User (User is the currently logged on user.) FieldName: The name of the required field in the defined context.

Example	If the user is currently viewing a case this returns the case description:
	`GetContextInfo("case", "case_description");`
	In the following example, the cases 'In Progress' for the current company context are displayed.
	`ThisCompanyId = CRM.GetContextInfo('Company','Comp_CompanyId');`
	`CaseListBlock = CRM.GetBlock('CaseList');`
	`SearchSql = 'Case_PrimaryCompanyId='+ThisCompanyId + " and`
	`Case_Status='In Progress' ";`
	`CRM.AddContent(CaseListBlock.Execute(SearchSql));`
	`Response.Write(CRM.GetPage());`

GetTabs(TabGroup)

Description	Use this method if you are using an old include file (for example the CRM.js file). You don't need to GetTabs if you use the SAGECRM.JS/ACCPACCRM.JS include file. Obtains the tabular information for the current context/view. You include this method in the first few lines of your ASP file (after the CRM.JS include file).The GetTabs method gets the current set of tabs which helps maintain the look of CRM within the ASP pages. You also use this method to set a new tab group to be displayed from a menu option. In this case, you include the new tab group name in the parentheses of the GetTab method.
Parameters	TabGroup: Name of tabgroup that you want to display.

Example	To display a new tab group that you have created, you can add a new menu button that links to an ASP that include the GetTabs method as follows:
	
	`<%= CRM.GetTabs("NewTabGroupName") %>`
	
	Add the following code to the top of your ASP page to display the tabs for the entity currently being viewed. If a company is viewed the company tabs are shown.
	
	`<%= CRM.GetTabs() %>`
	
	or
	
	`<% Response.Write(CRM.GetTabs()) %>`

Logon(LogonId, Password)

Description	Enables you to access and manipulate your CRM data externally from a command prompt. It enables you to utilize CRM as a COM object. This method initializes the CRM object, returns a blank string if successful, and otherwise it returns an error code. Note that for this function to work correctly, you must set the External Logon Allowed option to True, in Administration | Users | Users for the relevant user. This should not be used in ASP or .NET . When this method is used no metadata is available, this technique is for data manipulation only.
Parameters	LogonId: This is the user name as specified in the CRM Logon screen. Password: This is the user's CRM specific password.

Example	To start using the CRM object as a specified external user you use the following code in an external JavaScript page:
	`var CRM= new ActiveXObject('CRM.<crminstalldir>');`
	`CRM.Logon("Administration", "password");`
	
	Note that you can also pass in your encrypted password by copying and pasting it from the system database. Ideally you need to create a specific logon for external use of the system with limited access rights using an encrypted password for extra security.

Url(Action)

Description	Every URL in CRM needs to be specially formatted. You use this method to transform an unformatted URL to the required format. You then use the returned URL whenever you create a link in CRM.
Parameters	Action: Can be a URL, an ASP page or a .NET assembly reference. If it is an ASP file, custompages is prepended and the CRM context information is appended. You can also pass in an action string. Anything else returns the action untouched.

Example	Example 1: The following example displays a button that links to an ASP page.
	`CRM.Ad-`
	`dContent(CRM.Button("Chart","Cancel.gif",CRM.Url("system/InvChart.asp")));`
	`Response.Write(CRM.GetPage());`
	
	Example 2: The following example creates an anchor that links to the CRM Web site.
	`<A HREF='*<%=CRM.Url("http://www.mydomain.com")%>'>Click here to view the Web`
	`site</A>`
	
	Example 3: The following example creates a link that will reference the "RunQuickLook" base method of a .NET Dll called "QuickLook.dll" :
	`myContainer.AddButton(CRM.Button(“Add”,”new.gif”,CRM.Url(“QuickLook.dll-`
	`RunQuickLook”)));`

ConvertValue(Avalue, AfromCurr, AToCurr)

Description	Enables the user to convert a value from one currency to another.
Parameters	AValue : The value to be converted.AFromCurr : The identifier of the currency to convert from. AToCurr : The identifier of the currency to convert to. Returns a string containing the converted value, formatted to the number of decimals of the AToCurr. The AFromCurr and AToCurr must exist in the Curr_CurrencyID field of the Currency table. If either AFromCurr or AToCurr are not a valid currency, the result returns an error message.

Example	The following example converts the value 50,000 from Euro to Sterling.
	
	`var iValue;`
	`var iFromCurr;`
	`var iToCurr;`
	`iValue = 50,000;`
	`iFromCurr = 1; // Assuming this is the euro currency id`
	`iToCurr = 2; // Assuming this is the Sterling currency id`
	`CRM.AddContent("The value in Sterling is " +`
	`CRM.ConvertValue(iValue, iFromCurr, iToCurr));`
	`Response.Write(CRM.GetPage());`

Properties

FastLogon

Description	Prevents CRM from loading the metadata cache when a user logs on externally. Use with the Logon method. This should not be used in ASP or .NET . When this method is used no metadata is available, this technique is for data manipulation only.
Values	1=off, 2=low, 3=high. It is off by default.

Example	`var CRM= new ActiveXObject('CRM.<CRMinstalldir>');`
	`CRM.FastLogon=1;`
	`CRM.Logon("Administration", "password");`

TargetLists

Description	When referenced returns a TargetLists Object.
Value	Read only. Returns a TargetLists Object.
Example	See the CRMTargetLists Object (page 8-99) section for an example.

CRMBlock Object

The CRMBlock object is the base for all CRM blocks. The specific block type called by the CRM Object determines the actual implementation of each of the CRMBlock methods and properties. Preceding code:

```
block=CRM.GetBlock("myblock");
```

Methods

Execute(Arg)

Description	Performs the block action and returns the display contents of the block. The argument is optional and determined by the block type. It is necessary to use this method to execute any block.
Parameters	Arg:(Optional). Any value that relates to the block type.

Example	`block.Execute();`
	The following example executes the personlist block.
	`list=CRM.GetBlock("personlist");`
	`CRM.AddContent(list.execute());`
	`Response.Write(CRM.GetPage());`

Validate()

Description	Performs basic checking on data entries. For example Checks to see if a required field is blank. Normally used in 'if' statements.
Parameters	None

Example	`block.Validate();`
	The following example displays an error message if all relevant fields are not validated.
	`if ((CRM.Mode==Save) && (!block.Validate()))`
	`{error="<FONT color=red><B>Please correct the highlighted`
	`entries</B></FONT>";`
	`CRM.Mode=Edit;}`

Properties

ArgObj

Description	If the block is in a container, this property is an alternative way to pass parameters to the block.
Parameters	None

Example	`block.ArgObj=record;`
	The following examples passes the Search block result as the argument to the list block and displays the relevant search or results list.
	`SearchContainer = CRM.GetBlock('Container');`
	`SearchBlock = CRM.GetBlock('PersonSearchBox');`
	`SearchContainer.AddBlock(SearchBlock);`
	`if (CRM.Mode == 2) {`
	`resultsBlock = CRM.GetBlock('PersonGrid');`
	`resultsBlock.ArgObj = SearchBlock;`
	`SearchContainer.AddBlock(resultsBlock);}`
	`CRM.AddContent(SearchContainer.Execute());`
	`Response.Write(CRM.GetPage)`

CheckLocks

Description	Specifies whether the system checks if a record is in use before allowing it to be edited. Only applicable on blocks that may be used to edit records, that is, EntryGroup or Container Blocks. By default this is True. If the record is in use it displays an error message. You only use this property when you want to set it to false. Note that false must be lowercase.
Values	Boolean: true, false.
Parameters	None

Example	`Block=CRM.GetBlock("companyboxlong");`
	`Block.CheckLocks=false;`

DisplayForm

Description	Specifies if the block wraps itself in a form element.
Parameters	Boolean: True, false. Default is true.

Example	The following example wraps the companyboxlong block in a form element, which means it follows the normal Save/Change steps (such as performing validation tasks). If you set this to false the data is not saved in the form.
	`block=CRM.Getblock("companyboxlong");`
	`block.DisplayForm=true;`
	`CRM.AddContent(block.execute());`
	`Response.Write(CRM.GetPage());`

FormAction

Description	If the DisplayForm property is set to 'true', you can set the action the form takes using this property. By default the form action is blank, which causes a submit to return to the same page.
Parameters	None
Example	`block.FormAction="mypage.asp";`

Height

Description	Sets the block position in pixels or as a percentage of the screen. The value you place in this property determines how far the block appears from the top of the screen.
Parameters	None

Example	`block.Height=400;`
	`block.Height=40%`
	
	The following example sets the block to be 150 pixels from the top of the screen.
	
	`Block=CRM.GetBlock("companyboxlong");`
	`Block.Height="150";`
	`CRM.AddContent(Block.Execute());`
	`Response.Write(CRM.GetPage());`

Name

Description	Sets or gets the name of the current block. Used to retrieve the name of the current block. Note that setting the Name property only changes the name of this instance of the block.
Values	Name: String, the name of the block.

Example	The following example sets the block and returns the block name:
	
	`Block=CRM.GetBlock("entry");`
	`Block.Name="My New Block";`
	`CRM.AddContent(Block.Name);`
	`Response.Write(CRM.GetPage());`

NewLine

Description	If the block is an Entry or EntryGroup in a container, this property determines whether the block appears on a new line. Note this is the same as setting the field NewLine property from Administration | Customization | <entity> | Blocks.
Values	Boolean: True, false. Default is true.

Example	The following example inserts two Entry blocks into a container side-by-side.
	
	`Group=CRM.Getblock("container");`
	`block=CRM.GetBlock("companyboxlong");`
	`Group.AddBlock(block);`
	`block2=CRM.GetBlock("personboxshort");`
	`block2.NewLine=false;`
	`Group.AddBlock(block2);`
	`CRM.AddContent(Group.Execute());`
	`Response.Write(CRM.GetPage());`

ShowValidationErrors

Description	You use this method to enable validation errors to display when a user incorrectly enters information in an entry box. If an entry is invalid, an ErrorString is displayed.
Values	Boolean: True, false. By default this property is set to true.
Parameters	None

Example	`MyBlock=CRM.GetBlock("entrygroup");`
	`MyBlock.ShowValidationErrors=false;`

Title

Description	Used to set the block title. Note that this is the same as setting the block Title property from Administration | Customization | <entity> | Blocks.
Parameters	None

Example	The following example sets the company summary box title.
	
	`block=CRM.Getblock("companyboxlong");`
	`block.Title="Test Block";`
	`CRM.AddContent(block.Execute());`
	`Response.Write(CRM.GetPage());`

ViewName

Description	You use this property with List blocks when you want to define what view the list is based on. You only use this property with a List block that you are creating dynamically.
Values	String: Name of view.

Example	The following example creates a new list based on the vListCommunication view.
	`NewList=CRM.GetBlock("PersonList");`
	`NewList.ViewName="vListCommunication";`
	`NewList.AddGridCol("Pers_FullName");`
	`CRM.AddContent(NewList.Execute(""));`
	`Response.Write(CRM.GetPage());`

Width

Description	Sets the block width in pixels or as a percentage of screen
Parameters	None

Example	`block.Width=400;`
	`block.Width="50%";`
	The following example sets the width of the company summary box to 40%.
	`Block=CRM.GetBlock("companyboxlong");`
	`Block.Width="40%";`
	`CRM.AddContent(Block.Execute());`
	`Response.Write(CRM.GetPage());`

Mode

Description	Describes what state the ASP page is in and controls what happens to certain blocks when they are executed.
Values	Mode holds an integer value. 0 - View 1 - Edit 2 - Save 3 - PreDelete 4 - PostDelete Note that constants are declared for these values in the CRM include files.

Example	The following example changes the default mode of an EntryGroup block from 0 (View) to 1 (Edit). In this example the case entry block is set to display in edit mode.
	`Record = CRM.CreateRecord("Case");`
	`EntryGroup = CRM.GetBlock("CaseDetailBox");`
	`if (CRM.Mode == 0){`
	`CRM.Mode = 1;`
	`}`
	`CRM.AddContent(EntryGroup.Execute(Record));`
	`Response.Write(CRM.GetPage());`

CRMChartGraphicBlock Object

This block inherits all the capabilities of the GraphicBlock and adds to it the ability to generate a variety of different charts. These charts may depend on data retrieved via SQL or added through ASP for their values. In this way, they can be fully dynamic and represent data at a specific moment in time.

To initiate this block:

```
ChartGraph=CRM.GetBlock('chart');
```

You use the ChartGraphicBlock to draw different styles of chart and graph. The author of this ASP page controls the type of chart and all of the associated parameters. In addition, the ASP author chooses what database query to use for any chart or graph.

Methods

BackGradient(Visible, color1, color2)

Description	Applies a gradient to the background of a chart.
Parameters	Visible : True, false. color1 and color 2: Text WideString
Example	To have a blue gradient fading to white use: `ChartGraph.BackGradient(true,'Blue','White');`

ChartTitle(text)

Description	Adds a title to the chart. If text is blank then the title is removed allowing more room for the chart.
Parameters	Text : WideString, Default is Blank
Example	`ChartGraph.ChartTitle('Case Priority');`

ManualChartEntry(Value, MakeNull=true/false)

Description	Creates a chart where the data is not contained in a CRM table. It enables data to be hardcoded into a chart without relying on it being in a table. The parameters passed vary depending on the style of table in use (for example bar). The MakeNull parameter determines if this entry should be blank and is normally set to false.
Parameters	Value : WideString, MakeNull : True, false.

Example	`ChartGraph.ManualChartEntry('10,Jan',false);`
	`ChartGraph.ManualChartEntry('10,Feb',false);`
	`ChartGraph.ManualChartEntry('+5,Feb',false);`
	`ChartGraph.ManualChartEntry('20,Mar',false);`
	`ChartGraph.ManualChartEntry('30,Apr',false);`
	`ChartGraph.ManualChartEntry('-5,Apr',false);`

ShowLegend(true/false)

Description	Determines whether to show the legend for the chart.
Parameters	Boolean: True, false. Default is true.
Example	`ChartGraph.ShowLegend(true);`

Stylename(Style)

Description	Sets the style of the chart.
Values	'Bar': Standard bar chart 'Hbar': Horizontal bar chart 'Line': Line graph 'Stairs': Line graph in the form of stairs 'Pie': Pie chart 'FastLine': More basic line graph 'Area': Filled form of Line graph 'Point': Rectangular points are used 'Arrows': Values are shown with arrows 'Bubbles': Values are shown with bubbles Parameters Style: Text, default is 'Bar'.
Example	`ChartGraph.Style('Pie');`

Properties

LabelX

Description	This is the label given to the horizontal axis of a chart.
Values	Text : WideString
Example	`ChartGraph.LabelX='Date';`

LabelY

Description	This is the label given to the vertical axis of a chart.
Values	Text : WideString
Example	`ChartGraph.LabelY='Certainty%';`

SQLText=Text

Description	Initializes a database using the specified SQL. The first fields in the specified database are used for the chart if X,Y, or XL labels have not been set. CRM navigates through the fields in the table as defined by SQLText and uses the first fields it finds and is able to use.
Values	Text = Widestring

Example	`Chart=CRM.GetBlock('chart'):`
	`Chart.SQLText='Select * from OpportunityProgress Where '+`
	`'Oppo_OpportunityId='+OppId;`
	`CRMChartGraphicBlock Object`

XLProp=text

Description	Contains the field name, where the field data is text, which is used to display labels along the X-axis in place of values.
Values	Text: Widestring
Example	`ChartGraph.XLProp='Fld_Date';`

Xprop=text

Description	Contains the name of the field name that is used along the X-axis.
Values	Text: Widestring
Example	`ChartGraph.Xprop='Fld_Date';`

Yprop=text

Description	Contains the name of the field name that is used along the Y-axis.
Values	Text: Widestring
Example	`ChartGraph.Yprop='Fld_Salary';`

CRMContainerBlock Object

The CRMContainerBlock is used to group other blocks on a screen. Container blocks can also be nested inside other containers. An example of a Container block is a linked search panel and related list. This block contains the standard Sage CRM buttons: Change/Save, Delete, and Continue. Note that you can also configure Workflow buttons if they need to be made available on that screen. If any blocks that have buttons are included, the buttons are shown only once on the container block and then applied to all the internal blocks. A Container Block may have up to three fixed functionality buttons and any number of extra buttons. The three fixed buttons are:

- Standard **Change/Save** button. This button is displayed as 'Change' when the screen is in View mode and clicking it changes it to Edit mode. The button is displayed as 'Save' while in Edit mode and clicking it saves the changes and moves it to View mode. By default this button is always shown.
- **Delete** button. This button is displayed as 'Delete' when the screen is in View mode and clicking it moves to Confirm Delete mode. The buttons shows as 'Confirm Delete' and clicking it deletes the record and moves back to View mode. By default this button is not shown.
- **Continue** button. This button is displayed as 'Continue' when the screen is in View mode or 'Cancel' when the screen is in Edit or Confirm Delete mode. By default this button is not shown.

The CRMContainerBlock can be used as a wrapper for other blocks if you want to put more than one block on the screen at a time.

The Execute function on a block takes only one argument. When the Container Block is executed it passes its argument onto all its items as they are executed in turn. If it is required for item blocks in a Container Block to have different arguments for their Execute functions, their arguments can be set individually by setting the ArgObj property on each item block and not passing in any argument to the Container.

The following is the generic code for creating a container with two blocks:

```
//Create a container
Container = CRM.GetBlock("container");
// get two screens
Screen1 = CRM.GetBlock("Screen1");
Screen2 = CRM.GetBlock("Screen2");
// add them to the container block
Container.AddBlock(Screen1);
Container.AddBlock(Screen2);
// display the container block, which displays the
// two blocks it contains
CRM.AddContent(Container.Execute());
Response.Write(CRM.GetPage());
```

Methods

AddBlock(Block)

Description	Enables blocks to be added to a container.
Parameters	Block: This can be a reference to any block previously created with the GetBlock call.

Example	The following example creates a container block and adds two blocks to it.
	`MyContainer = CRM.GetBlock("container");`
	`MyPerson = CRM.GetBlock("personboxlong");`
	`MyCompany = CRM.GetBlock("companyboxlong");`
	`MyContainer.AddBlock(MyPerson);`
	`MyContainer.AddBlock(MyCompany);`
	`CRM.AddContent(MyContainer.Execute());`
	`Response.Write(CRM.GetPage());`

AddButton(ButtonString)

Description	This method adds an extra button to the Container block. The button string should be HTML to render the desired button. This HTML is added after the other buttons are drawn. Note that the easiest way to get the HTML for the button is by using the CRM.Button method, see below.
Parameters	ButtonString: String, the HTML to render the button. Ideally this should be a link within a <TABLE> tag in the ASP page.

Example	The following example adds a button called 'Try This' to the company summary block.
	`R = CRM.FindRecord('Company','Comp_CompanyId=1');`
	`Holder = CRM.GetBlock('companyboxlong');`
	`Holder.AddButton(CRM.Button("TryThis","new.gif",CRM.Url("Anot`
	`herPage.asp")));`
	`CRM.AddContent(Holder.Execute(R));`
	`Response.Write(CRM.GetPage());`

DeleteBlock(BlockName)

Description	Removes the given block from the container.
Parameters	BlockName: String, the name of the block to be deleted.

Example	In this example, CompanySummaryBlock is a Container block that has been set up within Administration | Customization | Company | Blocks. One of the blocks in it is the AddressBoxShort entry group. This example removes that block for non administration users.
	 `MyC = CRM.GetBlock("CompanySummaryBlock");`
	 `userLevel=CRM.GetContextInfo("User","User_Per_`
	 `Admin");`
	 `if (userLevel > 1)`
	 `{MyC.DeleteBlock("AddressBoxShort");}`
	 `CRM.AddContent(MyC.Execute());`
	 `Response.Write(CRM.GetPage());`

GetBlock(BlockName)

Description	The Container block GetBlock function differs from the Base block GetBlock function in that it returns blocks from within the container. This narrows the search. The GetBlock function returns a pointer to the block object referred to by BlockName if that block exists as one of the blocks within the container.
Parameters	BlockName: String, the name of the block to return

Example	In this example, MyCustomContainer is a Container block that has been set up within the Blocks section of Administration | Customization | Company | Blocks. This block contains three screens. One of the blocks in this Container is the CompanyBoxShort. This example displays that screen only.
	 `MyCustomContainer= CRM.GetBlock("MyCustomContainer");`
	 `R = CRM.FindRecord('Company','Comp_CompanyId=30');`
	 `MyE=MyCustomContainer.GetBlock("CompanyBoxShort");`
	 `CRM.AddContent(MyE.Execute(R));`
	 `Response.Write(CRM.GetPage());`

Properties

ButtonAlignment

Description	This property works with the ButtonLocation property to enable the position of the button(s) on the screen to be fine-tuned.
Values	ButtonAlignment is a numeric value. Button definitions are declared as constants in the CRM include file as follows: Bottom = 0, Left = 1, Right = 2, Top = 3. If ButtonLocation is Top or Bottom, ButtonAlignment may be set to Left, Center, or Right-default is Left if not set. If ButtonLocation is Left or Right then ButtonAlignment may be set to Top, Center, or Bottom-default is Top if not set.

Example	In the following example, the buttons are aligned to the left of the screen.
	`Container = CRM.GetBlock("container");`
	`Container.ButtonLocation = Top;`
	`Container.ButtonAlignment = 1;`
	`CRM.AddContent(Container.Execute());`
	`Response.Write(CRM.GetPage());`

ButtonImage

Description	This property enables you to change the image of the standard Change/Save button from the default image. The image file should be located in the ..img\buttons\.. folder of the CRM root directory. If you wish to store the image elsewhere you must specify the full path in the property.
Parameters	None

Example	The following example sets the image and text to be displayed on the CRM default button.
	`Container=CRM.GetBlock("container");`
	`Container.DisplayButton(Button_Default)=true;`
	`Container.ButtonTitle="My Button Title";`
	`CRM.AddContent(Container.Execute());`
	`Response.Write(CRM.GetPage());`

ButtonLocation

Description	This property enables you to set the location of the buttons. There are four options: 0 = Bottom (Bottom) 1 = Left (Left) 2 = Right (Right) 3 = Top (Top) If the location is Top or Bottom, the buttons are shown in a horizontal line, otherwise they are shown in a vertical line. Note that these options are declared as constants in the CRM include files.
Parameters	None: By default the buttons are displayed on the right of the Container.
Example	In the following example, the buttons are displayed at the top of the container.

```
Container=CRM.GetBlock('container');
Container.DisplayButton(Button_Delete)= true;
Container.DisplayButton(Button_Continue)=true;
Container.DisplayButton(Button_Default)=true;
Container.ButtonTitle="My Button Title";
Container.ButtonLocation=Top
CRM.AddContent(Container.Execute());
Response.Write(CRM.GetPage());
```

ButtonTitle

Description	This property allows the text shown on the Standard Change/Save (Button_Default) button to be overridden.
Parameters	None
Example	The following example sets the title on the default CRM button.

```
Container=CRM.GetBlock("container");
Container.DisplayButton(Button_Default)=true;
Container.ButtonTitle="My Button Title";
CRM.AddContent(Container.Execute());
Response.Write(CRM.GetPage());
```

DisplayButton

Description	This property is used to turn the three fixed buttons on or off as desired. The standard Change/Save button is the default button.
Parameters	The parameter specifies which button to turn on/off. The available options are: 1 = Standard Change/Save button (Button_Default) 2 = Delete Button (Button_Delete) 4 = Continue Button (Button_Continue) Note that the standard Save/Change buttons are declared as constants in the CRM include files.

Example	The following example creates a container block including all the buttons.
	`Container=CRM.GetBlock("container");`
	`Container.DisplayButton(Button_Delete) = true;`
	`Container.DisplayButton(Button_Continue)=true;`
	`Container.DisplayButton(Button_Default)=true;`
	`CRM.AddContent(Container.Execute());`
	`Response.Write(CRM.GetPage());`

Workflow Properties

The Workflow Container Block properties enable you to include the same kinds of buttons in an ASP page for any table in the system database, including new custom tables that have been added for a customer. The Container block has three properties that enable Workflow functionality:

- WorkflowTable
- ShowWorkflowButtons
- ShowNewWorkflowButtons

You can use these properties on tables that you have configured workflow rules and states for, where you want to display these rules and states as workflow buttons. For example, if you have workflow enabled on the Cases table in CRM, a 'New' button is displayed for every Primary workflow rule in the Cases List. When you edit a Case, you see the workflow buttons applicable to that case displayed.

Pre-conditions

In order to use the Workflow properties on a new custom CRM table, the table connection must have the usual CRM required fields, xxx_createdby, xxx_createddate, xxx_updatedby, xxx_updateddate, xxx_timestamp and xxx_deleted (where xxx is the prefix on all the fields in that table). The following conditions also apply:

- There must be a numeric field on the table to hold the workflow Id (this is typically called xxxx_ workflowid)
- When creating the table link, there is a field called 'WorkflowId Field' in the Table Details screen. You fill in the name of your workflow Id field here. For more information refer to Database Customization (page 5-1).
- The workflow rules/states/tree must be configured for the table in Administration | Advanced Customization | Workflow. For more information on Workflow administration please refer to the System Administrator Guide.

The Primary rules for a workflow on a new internal table must:

- Use the Custom File Name property. Typically this points to the edit.asp file page that displays the entry group.

- Have at least one field action, for example all Primary rules.
- The field actions must not include any fields that are already shown by the ASP page.

ShowNewWorkflowButtons

Description	You use this property on a screen showing a list of the records in a table when you want to show the 'New' button that creates a record in a workflow, that is, for all the Primary rules. Note that you must also use the Workflowtable property to set the table name.
Values	Boolean: True, false.

Example	The following example sets the workflow buttons to display in the table.
	`List = CRM.GetBlock('MyTableList');`
	`List.WorkflowTable = 'MyTable';`
	`List.ShowNewWorkflowButtons = true;`
	`CRM.AddContent(List.Execute(''));`
	`Response.Write(CRM.GetPage());`

ShowWorkflowButtons

Description	You use this property to display the Workflow buttons on a view or edit screen for one record. Note that for ShowWorkflowButtons to take effect, you must pass in the Record block as the argument to the containers Execute function. It does not work if the Record object is just set in the ArgObj property.
Values	Boolean value: True, false.

Example	The following example displays the workflow buttons on the EntryGroup.
	`Record = CRM.FindRecord('MyTable','Table_Id=99');`
	`EntryGroup = CRM.GetBlock('MyTableBlock');`
	`EntryGroup.ShowWorkflowButtons = true;`
	`CRM.AddContent(EntryGroup.Execute(Record));`
	`Response.Write(CRM.GetPage());`

WorkflowTable

Description	This property sets the name of the table which the 'ShowNewWorkflowButtons' is to use.
Parameters	Tablename: The name of the table.

Example	The following example forces the ShowNewWorkflowButtons property to use the Company table.
	`Container=CRM.GetBlock('container');`
	`Container.WorkflowTable='company';`

CRMContentBlock Object

The ContentBlock Object is a simple block that takes a string of content(text) and displays it on the page. This block is normally used in conjunction with other blocks on a screen.

The block has only one property "contents", which displays itself on the page. Any content must be formatted using HTML, to appear formatted on-screen.

The following is the generic code for creating a content block:

```
//Create a content block
Content= CRM.GetBlock("content");
Content.contents = "This is the contents";
CRM.AddContent(Container.Execute());
Response.Write(CRM.GetPage());
```

Properties

Contents

Description	Enter text in content block. You use HTML formatting to format the text.
Values	Text:Widestring

Example	This is an example of using a content block as a header for information on-screen
	`test=CRM.GETBlock('content');`
	`test.contents = '<TABLE> <TD CLASS=TABLEHEAD>Testing Details</TD></Table>';`
	`CRM.AddContent(test.Execute());`
	`Response.Write(CRM.GetPage());`

CRMEntryBlock Object

The CRMEntryBlock object corresponds to a single field that is to be displayed or edited on-screen. You can create many different entry types, such as text blocks, multiselect boxes and currency input boxes. You typically add Entry blocks to EntryGroups or Containers. You can use JavaScript scripts on these blocks to perform tasks when they load, change and are validated.

The CRMEntryBlock is a child of the CRM block. You usually add entries to an entrygroup or a container block but the EntryBlock does not inherit the properties or methods of the block it is added to.

The properties that apply to the CRMEntryBlock object are similar to the field properties available when adding entries to a screen within Administration | Customization. Preceding Code:

```
EntryGroup=CRM.GetBlock("GroupBlockName")
EntryGroup.AddEntry("entryname")
```

```
Entry= EntryGroup.GetEntry("entryname")
```

Methods

RemoveLookup

Description	You can set this property on Entry blocks that have entry type = 21 (Selection) to remove certain options from the lists.
Parameters	String : The code for the item to be removed.

Example	The following example removes the list item 'Customer' from the Type list on the Company Entry Screen.
	`r=CRM.FindRecord('Company','Comp_companyid=30');`
	`CompanyBlock=CRM.GetBlock('companyboxlong');`
	`NewE=CompanyBlock.GetEntry('comp_type');`
	`NewE.RemoveLookup("customer");`
	`CRM.AddContent(CompanyBlock.Execute(r));`
	`Response.Write(CRM.GetPage());`

Properties

AllowBlank

Description	This property can be set on any Entry block that is of entry type 21 (Selection). If the AllowBlank property is set to true, a blank option is added to the list to enable the field to be set to blank. If this is set to false, there is no blank option, thus forcing a value to be selected for the field. This is set to true by default, so you only set this property when you want to disable blank entries.
Values	Boolean : True, false.

Example	This example requires the user to select an option from the company revenue field, by setting the AllowBlank property to false.
	`r = CRM.FindRecord('Company','Comp_companyid=44');`
	`EG = CRM.GetBlock('companyboxlong');`
	`NewE = EG.GetBlock('comp_revenue');`
	`NewE.AllowBlank = false;`
	`CRM.AddContent(EG.Execute(r));`
	`Response.Write(CRM.GetPage());`

Caption

Description	Enables you to change the caption on an Entry. You use this property only if the caption is required to be different on this particular screen only. If the caption is to be permanently changed, it should be done in Administration | Customization | <entity> | Fields. The change is automatically reflected throughout the system.
Values	Value: String, new caption code. The translation for the caption on the entry is looked up using the Captions Family and this code.

Example	The following example creates a new caption called New Caption for the comp_revenue field in the company table.
	`r = CRM.FindRecord('Company','Comp_companyid=30');`
	`EG = CRM.GetBlock('companyboxlong');`
	`NewE = EG.GetBlock('comp_revenue');`
	`NewE.Caption = 'New Caption';`
	`CRM.AddContent(EG.Execute(r));`
	`Response.Write(CRM.GetPage());`

CaptionPos

Description	Enables you to re-position the captions on fields to reflect the value in the field. This is usually used with the Javascript Enumerator object.
Values	Numeric, may be one of the following: 1. Puts the caption on top of the values. 2. Puts the caption to the left of the values. 3. Puts the caption to the left of the values, captions left aligned and values left aligned. 6. Puts the caption to the left of the values, captions right aligned, values left aligned.

Example	The following example sets the caption on the company entry field to the left of the values, captions right aligned, values left aligned.
	`r = CRM.FindRecord('Company','Comp_companyid=30');`
	`CompBlock = CRM.GetBlock('CompanyBoxLong');`
	`eEntries = new Enumerator(CompBlock);`
	`while (!eEntries.atEnd()) {`
	`y = eEntries.item();`
	`y.CaptionPos = 6;`
	`eEntries.moveNext();}`
	`CRM.AddContent(CompBlock.Execute(r));`
	`Response.Write(CRM.GetPage());`

CreateScript

Description	Enables you to enter server-side JavaScript that is executed when the entry is created. This is limited to this instance of the entry.
Values	String: JavaScript. Within the JavaScript any of the current Entry block properties can be accessed.

Example	In this example the CreateScript property sets the maximum length of this instance of the Entry block comp_name to 20 characters.
	`r = CRM.FindRecord('Company','Comp_companyid=30');`
	`EG = CRM.GetBlock('companyboxlong');`
	`NewE = EG.GetBlock('comp_name');`
	`NewE.CreateScript = "MaxLength=20";`
	`CRM.AddContent(EG.Execute(r));`
	`Response.Write(CRM.GetPage());`

DefaultType

Description	You use this property to set the default type of the entry. This is used in conjunction with the EntryType and DefaultValue properties.
Values	Numerical: 0 (iDefault_NoDefault). No default value. 1 (iDefault_Value). Use the value set in the DefaultValue property. 2 (iDefault_CurrentUserId). For fields that hold a user id, this defaults to the current logged in user. 6 (iDefault_CurrentDateTime). For date time fields, this defaults them to the current date and time. 14 (iDefault_CurrentDateTimePlus30Mins). Also for date time fields, this defaults them to the current date and time plus 30 minutes.

Example	The following example sets the default text of the Entry Block Company Name to use the value set in the DefaultValue property.
	`R = CRM.CreateRecord('company');`
	`EG = CRM.GetBlock('companyboxlong');RevenueE =`
	`EG.GetEntry('comp_name');`
	`RevenueE.DefaultType = 1;`
	`RevenueE.DefaultValue = 'New company name';`
	`CRM.AddContent(EG.Execute(R));`
	`Response.Write(CRM.GetPage());`

DefaultValue

Description	Specifies the default value given to the field when a new record is created. You can only use this property if you have already set the DefaultType property to iDefault_DefaultValue (1).
Values	Any string value.
Parameters	None

Example	The following example sets the default text of the Entry block Company Name.
	`R = CRM.CreateRecord('company');`
	`EG = CRM.GetBlock('companyboxlong');`
	`E = EG.GetEntry('comp_name');`
	`E.DefaultType = 1;`
	`E.DefaultValue = 'New company name';`
	`CRM.AddContent(EG.Execute(R));`
	`Response.Write(CRM.GetPage());`

EntryType

Description	Used to change the way that the field can be edited. This is only relevant to EntryBlocks not tied to actual fields, that is, EntryBlocks that are the result of a call to CRM.GetBlock("entry").
Values	10 iEntryType_Text . Single line text entry. 11 iEntryType_MultiText. Multi line text entry. 12 iEntryType_EmailText. E-mail address. 13 iEntryType_UrlText. Web URL address. 21 iEntryType_Select. Selection from lookup (combo box). 23 iEntryType_ChannelSelect. Select from channel table. 25 iEntryType_ProductSelect. Select from product table . 28 iEntryType_MultiSelect. Multiple selection from lookup (combo box). 31 iEntryType_Integer. Enter an integer. 45 iEntryType_CheckBox. Checkbox. 49 iEntryType_Password. Password. 51 iEntryType_Currency. Currency.

Example	You specify a FieldName before the EntryType.
	`Entry=CRM.GetBlock('entry')`
	`Entry.FieldName="Check Box";`
	`Entry.EntryType= 45;`

FAM

Description	Enables the family of the EntryGroup to be set. This controls what captions appear on each entry. By default the caption shown is the translation for the caption family (column names) plus the caption code (field name). You can change the caption by changing the FAM value and adding a translation for that FAM and the field name. Note that you can view a list of column names in Administration | Customization | Translations.
Values	Fam : String, the new family to use to find the translation for the caption on the entry.

Example	The following example changes the caption family of the comp_name entry.
	`c=CRM.GetContextInfo('company','Comp_CompanyId');`
	`CompanyRec=CRM.FindRecord('company','Comp_CompanyId='+c);`
	`CompanyBlock=CRM.GetBlock("companyboxlong");`
	`name = CompanyBlock.GetEntry('comp_name');`
	`name.Fam = 'My Family';`
	`Response.Write(CompanyBlock.Execute(CompanyRec));`

FieldName

Description	Specifies the name by which the field is referenced. This property is only relevant on EntryBlocks that are not tied to actual fields, that is, EntryBlocks that are the result of a call to CRM.GetBlock("entry").
Values	Any string value.
Parameters	None

Example	`Entry.FieldName = "Name";`
	The following example sets the name of the new check box entry to Check Box.
	Note that you specify the FieldName before you set the EntryType.
	`Entry=CRM.GetBlock("entry");`
	`Entry.FieldName="Check Box";`
	`Entry.EntryType=45;`
	`CRM.AddContent(Entry.Execute());`
	`Response.Write(CRM.GetPage());`

Hidden

Description	Enables an Entry to be set as hidden, which prevents it from being displayed when the EntryGroup it is in is executed. This is useful if you want to tag an entry to an entry group but do not want customers to view it.
Values	Boolean: True, false.

Example	The following example sets the company_revenue Entry block to be hidden.
	`r = CRM.FindRecord('Company','Comp_companyid=22');`
	`EG = CRM.GetBlock('companyboxlong');`
	`NewE = EG.GetBlock('comp_revenue');`
	`NewE.Hidden = true;`
	`CRM.AddContent(EG.Execute));`
	`Response.Write(CRM.GetPage());`

JumpEntity

Description	Allows the field (in view mode) to be hyperlinked to an entity summary screen. Note that the entity must be relevant to the field, that is, the identity field of the entity selected must exist within the table or view on which the screen is based. In practice, this is only useful when the screen is based on a view that contains fields from multiple tables.
Values	Entity Name: Company, Person, Communication, Case, Opportunity, Solution, Address, Library or Notes.

Example	The following example sets the company name field in the companyboxlong to jump to the company entity.
	`c=CRM.GetContextInfo('company','Comp_CompanyId');`
	`CompanyRec=CRM.FindRecord('company','Comp_CompanyId='+c);`
	`userLevel=CRM.GetContextInfo('user','User_Per_Admin');`
	`//Start with the Company Entry Screen`
	`CompanyBlock=CRM.GetBlock('companyboxlong')`
	`// jump from comp_name to company`
	`name = CompanyBlock.GetEntry('comp_name');`
	`name.JumpEntity = 'Company';`
	`CRM.AddContent(CompanyBlock.Execute(CompanyRec));`
	`Response.Write(CRM.GetPage());`

LookUpFamily

Description	Allows the Lookup family to be set on an entry if entry type is 21 (selection). The lookup family defines what entries appear in the list for this entry. For example, if the lookup family is 'DayName' you get a list of days.
Values	Lookup family: String, the name of the family.
Example	The following example creates a new selection entry group that uses the DayName family for selection items.

```
NewE = CRM.GetBlock("entry");
NewE.EntryType = 21;
NewE.Caption="Days of the week";
NewE.LookupFamily = "DayName";
EG.AddBlock(NewE);
CRM.AddContent(EG.Execute());
Response.Write(CRM.GetPage());
```

MaxLength

Description	Controls the maximum amount of characters that can be entered into an EntryGroup block when editing this entry. This does not change the size of the entry box, only the number of characters that can be entered.
Values	Numeric value, number of characters to edit.
Example	The following example sets the entry block comp_name to allow only five characters.

```
r = CRM.FindRecord('Company','Comp_companyid=22');
EG = CRM.GetBlock('companyboxlong');
NewE = EG.GetBlock('comp_name');
NewE.MaxLength = 5;
CRM.AddContent(EG.Execute(r));
Response.Write(CRM.GetPage());
```

MultipleSelect

Description	This property can be set on any Entry Block that has Entry type = 21 (selection), and defines whether you can select more than one item from the list. If you use this property you must save the multiple values in a relevant location, for example in a link table. If you set the MultipleSelect property to 'true', you should ensure that the size of the block is sufficient to accommodate all the entries.
Values	Boolean value: True or false.

Example	The following example sets the company territory Entry block of the companyboxlong entry group to be multiple select, and sets the size of the entry to 10.
	`b = CRM.GetBlock('companyboxlong');`
	`e = b.GetBlock('comp_source');`
	`e.MultipleSelect=true;`
	`e.Size = 10;`
	`r = CRM.FindRecord('company','comp_companyid=892');`
	`CRM.AddContent(b.Execute(r));`
	`Response.Write(CRM.GetPage());`

OnChangeScript

Description	Specifies the JavaScript to be executed when the value in the field is edited. You can only use this property when the ReadOnly property is set to false.
Values	String of JavaScript.

Example	`Entry.OnChangeScript = "alert('field changed')";`
	The following example displays an alert box when a user's first name is changed.
	`ThisPersonId = CRM.GetContextInfo('Person','Pers_PersonId');`
	`ThisPersonRecord=CRM.FindRecord('Person','Pers_PersonId=17');`
	`PersonBlock = CRM.GetBlock('PersonBoxLong');`
	`FirstName = PersonBlock.GetEntry('Pers_FirstName');`
	`FirstName.OnChangeScript="alert('Name Changed')";`
	`CRM.AddContent(PersonBlock.Execute(ThisPersonRecord));`
	`Response.Write(CRM.GetPage());`

ReadOnly

Description	Specifies that a field is read-only. If this is set to true, the value in the field is not editable.
Values	Boolean: True, false.

Example	`Entry.ReadOnly = 'false';`
	 The following example sets the TitleCode field in the PersonBoxLong entrygroup to be read only.
	 `Record=CRM.FindRecord('person', 'pers_personid=17');`
	 `PersonBlock = CRM.GetBlock('PersonBoxLong');`
	 `Title = PersonBlock.GetEntry('Pers_Titlecode');`
	 `Title.ReadOnly = 'true';`
	 `CRM.AddContent(PersonBlock.Execute(Record));`
	 `Response.Write(CRM.GetPage());`

Required

Description	Specifies that a value must be entered in this field. If no value is entered a validation error is displayed.
Values	Boolean: True, false.

Example	`Entry.Required = true;`
	 The following example makes the pers_title field a required field.
	 `Block=CRM.GetBlock('PersonBoxShort');`
	 `Title=Block.GetEntry('Pers_Title');`
	 `Title.Required = true;`
	 `CRM.AddContent(Block.Execute());`
	 `Response.Write(CRM.GetPage());`

Size

Description	Specifies the size of the field displayed on-screen. This is the length of the field in characters.
Values	Any integer.

Examples	`Example Entry.Size = 40;`
	
	The following example retrieves the FirstName field from the PersonBoxLong screen and sets its size to 40 characters.
	
	`ThisPersonId = CRM.GetContextInfo('Person','Pers_PersonId');`
	`ThisPersonRecord =`
	`CRM.FindRecord('Person','Pers_Personid='+ThisPersonId);`
	`PersonBlock = CRM.GetBlock('PersonBoxLong');`
	`FirstName = PersonBlock.GetEntry('Pers_FirstName');`
	`FirstName.Size = 40;`
	`CRM.AddContent(PersonBlock.Execute(ThisPersonRecord));`
	`Response.Write(CRM.GetPage());`

ValidateScript

Description	Enables you to set a validation script on an Entry block. You use this to enter server-side JavaScript that is executed when the entry is executed in save mode.
Values	String : JavaScript. You can set the ErrorStr variable within the script to display a customized error message. The script should set the Valid variable to true or false. If Valid is set to false, the screen remains in edit mode and displays an error message. In addition, the field on which the Validate script is set is be shown with an orange question mark symbol (?) beside it.

Example	The following example sets the Valid variable to true if the company name field does not contain a value.
	
	`r = CRM.FindRecord('Company','Comp_companyid=30');`
	`EG = CRM.GetBlock('companyboxlong');`
	`NewE = EG.GetBlock('comp_name');`
	`NewE.ValidateScript = "Valid = (comp_name.value != '');if (!Valid)`
	`ErrorStr = 'Please correct the highlighted entries';";`
	`CRM.AddContent(EG.Execute(r));`
	`Response.Write(CRM.GetPage());`

AllowUnassigned

Description

This property can be set on Entry blocks where the entry type is TableSelect, that is, for lists to select Users. It allows you to change the caption used for the 'None' item in the list to be 'Unassigned' instead. The effect of setting AllowUnassigned, depends on the value of the AllowBlank property as follows:

AllowUnassigned	**Allow Blank**	**Result**
false	false	No 'None' option
true	false	Unassigned option
false	true	None' option
true	true	None' option

Values

Boolean: True, False.

Example

The following example displays a company block. When you click the Change button the Account Manager selection includes the option Unassigned (instead of None).

```
EntryGroup = CRM.GetBlock('companyboxlong');
Record = CRM.FindRecord('Company','Comp_CompanyId=30');
UserSelect = EntryGroup.GetBlock('comp_primaryuserid');
UserSelect.AllowUnassigned = true;
UserSelect.AllowBlank = false;
CRM.AddContent(EntryGroup.Execute(Record));
Response.Write(CRM.GetPage());
```

Restrictor

Description

You can set this property on Entry blocks that have entry type = 56 (Advanced Search Select) to select another Advanced Search Select field that will restrict the searched values for the field.

Values

WideString: read Get_Restrictor write: Set_Restrictor.

Example

If you are creating a new Advanced Search Select field on a communication screen, for example, you can restrict it based on an existing Advanced Search Select field. Let's assume that the existing field mentioned is used to search for companies.

```
Restrictor="cmli_comm_companyid".
```

This means if the existing field is filled in, the new field will only search for records belonging to the selected company.

CopyErrorsToPageErrorContent

Description	When set to true, validation errors from this block will be shown at the top of the page, rather than beside (each) block on the page. Use this property when you have multiple blocks on a single page.
Values	Boolean

Example	This script will create two entry blocks, one of which will fail validation. The CopyErrorsToPageErrorContent setting will ensure that the resulting error is displayed at the top of the page
	`CompanyEntryGroup=eWare.GetBlock("CompanyBoxLong");`
	`CompanyEntryGroup.Title="Company";`
	`AddressEntryGroup = eWare.GetBlock("AddressBoxLong");`
	`AddressEntryGroup.Title = "Address";`
	`//Set Valid=false so that this field will always fail validation no matter what is entered`
	`var address1field = AddressEntryGroup.GetEntry("addr_address1");`
	`address1field.ValidateScript = "Valid=false;ErrorStr='Value not correct'";`
	`//Set CopyErrorsToPageErrorContent = true for both of the blocks so that the error message`
	`//will appear at the top of the page`
	`CompanyEntryGroup.CopyErrorsToPageErrorContent = true;`
	`AddressEntryGroup.CopyErrorsToPageErrorContent = true;`
	`container=eWare.GetBlock("container");`
	`container.AddBlock(CompanyEntryGroup);`
	`container.AddBlock(AddressEntryGroup);`

CRMEntryGroupBlock Object

The EntryGroupBlock object corresponds to a screen in CRM. The methods of the EntryGroupBlock Object are used to group entries to create screens and control custom data entry and editing. You can generate many different kinds of entry groups, such as a Company Search Box, a Person Entry Box, and a Case Detail Box. This block also contains the standard CRM buttons.

Methods

AddEntry(EntryName, Position, Newline)

Description	Allows new entries to be added dynamically to EntryGroups. The changes do not apply outside the ASP page in which they are used. Values Returns an EntryBlock object.
Parameters	EntryName: Specifies the entry to be added. Can be passed in as either the name of the field or as an actual EntryBlock that already exists. In both cases, the field must be relevant to the existing EntryGroup Block, that is, it must exist on the table that the EntryGroup Block is based on. Position (Optional): Optional numerical parameter that specifies the position in which to add the entry. Passing in a value of 0 (zero), inserts the entry into the first position, minus one (-1) adds the entry to the end of the group and so on. One (1) is the default behavior if the position is not specified. NewLine (Optional): Optional boolean parameter specifies if the entry is to be shown on a new line or not. By default it is true.

Example	This example adds the Fax Number and phone Number fields to the start of the entry group.
	
	`EntryGroup = CRM.GetBlock("personboxlong");`
	`EntryGroup.AddEntry("pers_faxnumber", 0, false)`
	`EntryGroup.AddEntry("pers_phonenumber",0,false);`
	`CRM.AddContent(EntryGroup.Execute())`
	`Response.Write(CRM.GetPage());`

DeleteEntry(EntryName)

Description	Deletes the specified entry from the group.
Values	No return value.
Parameters	EntryName: String, the name of the field within the EntryGroup that is to be deleted.

Example	This example removes the Revenue field from CompanyBoxLong for non administration users.
	
	`R = CRM.FindRecord('Company','Comp_CompanyId=30');`
	`MyC = CRM.GetBlock('CompanyBoxLong');`
	`userLevel=CRM.GetContextInfo('user','User_Per_Admin');`
	`if (userLevel < 3) {MyC.DeleteEntry('Comp_Revenue');}`
	`CRM.AddContent(MyC.Execute(R));`
	`Response.Write(CRM.GetPage());`

GetEntry

Description	This method returns a reference to the Entry specified.
Values	Returns an EntryBlock object.
Parameters	EntryName: The name of the field within the EntryGroup that is required. If the field does not exist a nil object is returned.
Example	The following example retrieves the Pers_FirstName entry from the PersonBoxShort entrygroup and sets it to be read only.

```
ThisPersonId = CRM.GetContextInfo('Person','Pers_PersonId');
ThisPersonRecord =
CRM.FindRecord('Person','Pers_Personid='+ThisPersonId);
PersonBlock = CRM.GetBlock('PersonBoxShort');FirstName =
PersonBlock.GetEntry('Pers_FirstName');
FirstName.ReadOnly = true;
CRM.AddContent(PersonBlock.Execute(ThisPersonRecord));
Response.Write(CRM.GetPage());
```

Properties

ShowSavedSearch

Description	Use this property to show the Saved Search functionality as part of the entry group. This properly is only applicable to Entry Group Blocks where the screen type is Search Block and the Block must have an associated List. When you set this property to true, the execution of the block will also show a list of the saved searches for that entity and allow you to create and edit them. This would typically be used in a page that contains an entry block and a list and is used to do searches.
Values	Boolean: default value is false, set to true to enable.
Example	

```
searchEntry=CRM.GetBlock("ProjectsSearchBox");
searchEntry.ShowSavedSearch=true;
searchList=CRM.GetBlock("ProjectsGrid");
searchContainer=CRM.GetBlock("container");
searchContainer.ButtonTitle="Search";
searchContainer.ButtonImage="Search.gif";
searchContainer.AddBlock(searchEntry);
if( CRM.Mode != 6)
searchContainer.AddBlock(searchList);
searchContainer.AddButton(CRM.Button("Clear", "clear.gif",
"javascript:document.EntryForm.em.value='6';document.EntryForm.su
bmit();"));
searchList.ArgObj=searchEntry;
CRM.AddContent(searchContainer.Execute(searchEntry));
Response.Write(CRM.GetPage());
```

CRMFileBlock Object

The CRMFileBlock object provides access to external files that are not part of the system. It allows these files to appear as if they are part of the system and to be called upon using ASP in the same way as any other CRM page. Note that the files need to be formatted for HTML appearance. If you don't specify where the file is stored in the directory path property, the system looks for the file in the CRM Reports directory.

An example of the use of this block is as follows:

```
var afile
afile=CRM.GetBlock('file');
afile.FileName='general.htm';
afile.Translate=false;
afile.ProperCase=false;
afile.DirectoryPath='C:\\directory\\folder\\';
CRM.AddContent(afile.Execute());
Response.Write(CRM.GetPage());
```

You can also select a translate option which dynamically chooses a file based on the user's language code, or a propercase function which displays the text with initial caps. These simple ASP statements includes the named file on the page. If the author chooses to set translate to 'true', the file name included is changed from filename to filename_US or filename_DE, depending on the User's language code in CRM. If the filename suffix is not supplied, .txt is assumed. This means you need to specify ".htm" and format the text.

Properties

DirectoryPath

Description	Specifies the directory where the files are contained. If you do not specify the full directory path, the CRM Reports directory is assumed.
Values	Text : WideString

Example	`afile=CRM.GetBlock('file');`
	`afile.FileName='Results.htm';`
	`afile.DirectoryPath='c:\\directory\\folder';`

FileName

Description	Specifies the file name to be used by the FileBlock.
Values	Text : WideString

Example	`afile=CRM.GetBlock('file');`
	`afile.FileName='Results.htm';`

ProperCase

Description	Applies proper case formatting to the text, that is, the first letter of every word is in uppercase.
Values	Boolean: True, false. Default is false.

Example	The following example sets the text of the file general.htm to propercase.
	`var afile`
	`afile=CRM.GetBlock('file');`
	`afile.FileName='general.htm';`
	`afile.ProperCase=true;`
	`CRM.AddContent(afile.Execute());`
	`Response.Write(CRM.GetPage());`

Translate

Description	When set to 'true' the block searches for a filename_userslanguage. In the above example this is results_US.htm or results_FR.htm, depending on the user's chosen language.
Values	Boolean: True, false. Default is false.

Example	`afile=CRM.GetBlock('file');`
	`afile.FileName='Results.htm';`
	`afile.Translate=true;`

CRMGraphicBlock Object

Description

The CRMGraphics Block object is a child of the CRM Block and parent of the PipeLineGraphic, OrgChartGraphic and ChartGraphicBlock.

The block enables images to be displayed through an ASP page. Graphics Blocks are more powerful than standard static images because variables can be used in their creation. These variables may represent live data from a database or incorporate details of the current user such as their privileges or settings. Graphics created by the Graphics Block are recreated every time they are requested so, where variables are used, the graphic is based on real time data.

The option to load in previously created images means that backgrounds can be employed or other images can be altered to represent a new situation.

The graphics can consist of basic details, such as text and lines, or more complex graphics employing various effects such as gradients and rotation.

To initiate this block:

```
graphic=CRM.GetBlock('graphic');
```

Methods

Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4)

Description	Draws an elliptically curved line. The arc traverses the perimeter of an ellipse that is bound by the points (X1,Y1) and (X2,Y2). The arc is drawn following the perimeter of the ellipse, counter clockwise, from the starting point to the ending point. The starting point is defined by the intersection of the ellipse and a line defined by the center of the ellipse and (X3,Y3). The ending point is defined by the intersection of the ellipse and a line defined by the center of the ellipse and (X4, Y4).
Parameters	X1,Y1,X2,Y2,X3,Y3,X4,Y4 : integer

Example	`Graphic=CRM.GetBlock("graphic");`
	`Graphic.ImageWidth=70;`
	`Graphic.ImageHeight=50;`
	`Graphic.Effect('ChangeColor','White,Red');`
	`Graphic.Arc(10,10,25,25,30,30,40,40);`
	`CRM.AddContent(Graphic.execute());`
	`Response.Write(CRM.GetPage());`
	`CRMGraphicBlock Object`

Animation(Mode, Value)

Description	The Graphics Block supports animation. Frames contained in an animation can be shown at varying intervals using the Delay mode. Using 'Add', the current state of the image is saved as a frame to be shown after the specified delay. The whole animation can be looped for a definite or indefinite number of times. This animation technique can also be used for charts. The delay is specified where 100=1 second and indefinite loops can be obtained by setting the Loop value to 0.
Parameters	Mode : WideString, Value : Integer

Example	`Graphic.Animation('Delay','100'); Graphic.Animation('Loop','0');`
	`Graphic.Animation('Add','100');`

Brush(Mode, Value)

Description	Changes the color and pattern used when drawing the background or filling in graphical shapes. The pattern can be one of a predetermined list using the Style mode or can be from an image using the Load mode.
Parameters	Mode : WideString, Value : WideString

Example	`Graphic.Brush('Load','c:\\winnt\\soap bubbles.bmp');`
	`Graphic.Brush ('Color','Blue');`
	`Graphic.Brush ('Fill','10,10,50,50');`
	`Graphic.Brush ('Style','cross');`

Chord(X1,Y1,X2,Y2,X3,Y3,X4,Y4)

Description	Creates a shape that is defined by an arc and a line that joins the endpoints of the arc. The chord consists of a portion of an ellipse that is bound by the points (X1,Y1) and (X2,Y2). The ellipse is bisected by a line that runs between the points (X3,Y3) and (X4,Y4). The perimeter of the chord runs counter clockwise from (X3, Y3), counterclockwise along the ellipse to (X4,Y4), and straight back to (X3,Y3). If (X3,Y3) and (X4,Y4) are not on the surface of the ellipse, the corresponding corners on the chord are the closest points on the perimeter that intersect the line.
Parameters	X1,Y1,X2,Y2,X3,Y3,X4,Y4 : integer.
Example	`Graphic.Chord(10,10,25,25,30,30,40,40);`

Effect(Mode, Value)

Description	Makes various effects available. The name of the effect is passed through Mode with parameters it may require being passed through Value.
Parameters	Mode : WideString, Value : WideString

Example	`Graphic.Effect('Zoom','200');`
	`Graphic.Effect('Transparent','True');`
	`Graphic.Effect('Dither','Floyd');`
	`Graphic.Effect('Merge','c:\\winnt\\winnt.bmp, White,0,0');`
	`Graphic.Effect('DisplayErrors','false');`
	`Graphic.Effect('Clear','');`
	`Graphic.Effect('ChangeColor','White,Red');`

Ellipse(X1,Y1,X2,Y2)

Description	Draws a circle or ellipse. Specify the bounding rectangle by giving the top left point at pixel coordinates (X1, Y1) and the bottom right point at (X2, Y2). If the bounding rectangle is a square, a circle is drawn. The ellipse is drawn using the current pen width and color.
Parameters	X1,Y1,X2,Y2 : integer

Example	`Graphic=CRM.GetBlock("graphic");`
	`Graphic.ImageWidth=70;`
	`Graphic.ImageHeight=50;`
	`Graphic.Effect('ChangeColor','White,Red')`
	`Graphic.Ellipse(10,10,50,50);`
	`CRM.AddContent(Graphic.execute());`
	`Response.Write(CRM.GetPage());`

FlipHoriz()

Description	Flips the image horizontally.
Parameters	None.
Example	`Graphic.FlipHoriz();`

FlipVert()

Description	Flips the image vertically.
Parameters	None.
Example	`Graphic.FlipVert();`

Font(Mode, Value)

Description	Allows for various changes to be carried out on the current font depending on the value specified in Mode. These changes then take effect with TextOut commands. To ensure success, use True Type fonts. Modes available include: • Name. Change the current typeface. • Size. Size of font to use. • Color. Color of font to use. • Bold. Toggle between a bold typeface and normal. • Italic. Toggle between the use of italics and normal. • Underline. Toggle between using underline and normal. • Strikeout. Toggle between having text striked out and normal. • Rotate. Performs a rotation on the font, although this does not work for all fonts.
Parameters	Mode : WideString, Value : WideString.

Example	`Graphic.Font('Name','Times New Roman');`
	`Graphic.Font('Color','Blue');`
	`Graphic.Font('Size','24');`
	`Graphic.Font('Bold','True'); Graphic.Font('StrikeOut','True');`
	`Graphic.Font('Rotate','45');`

Graphic Font Properties Example:

```
<%
Graphic=CRM.GetBlock("graphic");
Graphic.LoadImage("go.gif");
Graphic.ImageWidth=130;
Graphic.ImageHeight=50;
Graphic.vSpace=30;
Graphic.hSpace=30;
Graphic.Border=3;
Graphic.Font('Name','Times New Roman');
Graphic.Font('Color','Blue');
Graphic.Font('Size','24');
Graphic.Font('Bold','True');
Graphic.Font('StrikeOut','True');
CRM.AddContent(Graphic.execute());
Response.Write(CRM.GetPage());%>
```

FontColor(Color)

Description	Enables the color of the current font to be changed depending on the value specified. This is the same as using the Font(Mode, Value) except it refers only to the color mode. It is quicker to use if this is the only mode you are resetting.
Parameters	Color: Text, widestring, color of font to use.
Example	`Graphic.FontColor('blue');`

FontSize(Size)

Description	Enables the size of the current font to be changed depending on the value specified. This is the same as using the Font(Mode, Value) except it refers only to the size mode. It is quicker to use if this is the only mode you are resetting.
Parameters	Size: The size of the font to use, in pixels.
Example	`Graphic.FontSize(24);`

GradientFill(StartColor, EndColor, Direction, Colors)

Description	Fills the graphic with a gradient of colors starting from the color specified in StartColor and ending with the one specified in EndColor. The Direction can be TopToBottom, BottomToTop, LeftToRight or RightToLeft. 'Colors' is a numeric parameter that specifies the number of colors to use when creating your gradient. This defaults to 64. The more colors used, the better the gradient effect. Gradients usually look better in 24 bit JPEG images, as the colors that can be used with GIFs is more limiting.
Parameters	StartColor : WideString, EndColor : WideString, Direction : WideString, Colors : Integer

Example	`Graphic.GradientFill('Yellow','White','LeftToRight');`
	`Graphic.GradientFill('Blue','White','TopToBottom',256);`

GrayScale()

Description	Converts an image to grayscale. It does not reduce the number of colors in use.
Parameters	None
Example	`Graphic.Grayscale();`

LoadBMP(Filename)

Description	This method is the same as the LoadImage Graphic block method except that it enables you to specifically load a bitmap file. Loads the file specified as the new image. The ImageWidth and ImageHeight changes to the dimensions of the new image.
Parameters	Filename: Text, widestring, absolute server address.

Example	This example loads a bitmap image called redmarble.bmp.
	`Graphic=CRM.GetBlock('graphic');`
	`Graphic.LoadBMP("D:\\Program`
	`Files\\Sage\\CRM\\CRM58\\WWWRoot\\Img`
	`\\plain.bmp");`
	`CRM.AddContent(Graphic.Execute());`
	`Response.Write(CRM.GetPage());`

LoadImage(text)

Description	Loads the file specified as the new image. The ImageWidth and ImageHeight changes to the dimensions of the new image. The following image formats are supported: • .BMP, Windows Bitmap • .ICO, Icon • .GIF, 256 color compressed image. • .JPG, 24 bit color compressed image • .WMF / .EMF, Windows / Enhanced Metafile If you store the image in the Img folder of the CRM directory, you need only name the file. Otherwise, you must include the complete path.
Parameters	Text : Widestring.

Example	On the server:
	`Graphic.LoadImage('cancel.gif');`
	`\\Server absolute address:`
	`Graphic.LoadImage('c:\\winnt\\cancel.gif');`

LoadJPG(Filename)

Description	This method is the same as the LoadImage method except that it enables you to specifically load a JPEG file. Loads the file specified in text as the new image. The ImageWidth and ImageHeight changes to the dimensions of the new image.
Parameters	Filename: Text, widestring

Example	This example displays a JPEG file called dashboard.jpg.
	`Graphic=CRM.GetBlock('graphic');`
	`Graphic.LoadJPG("D:\\Program`
	`Files\\Sage\\CRM\\CRM58\\WWWRoot\\Img\\`
	`recentbackground.jpg");`
	`CRM.AddContent(Graphic.Execute());`
	`Response.Write(CRM.GetPage());`

LineTo(X,Y)

Description	Draws a line from the current pen position up to, but not including the points specified by the numbers in X and Y. This method also changes the pen position to the co-ordinates specified in (X,Y). The line is drawn using the current pen width and color.
Parameters	X,Y : Integer
Example	`Graphic.LineTo(50,50);`

Monochrome()

Description	Converts an image to monochrome using only two colors, black and white. Note that when an image is set to monochrome, the changes that occur are irreversible unless the image is redrawn.
Values	Boolean: True, false.
Example	`Graphic.Monochrome(true);`

MoveTo(X,Y)

Description	MoveTo changes the pen position to co-ordinates specified in (X,Y). Use MoveTo to set the current pen position before calling LineTo.
Parameters	X,Y : integer
Example	`Graphic.MoveTo(50,50);`

Pen(Mode, Value)

Description	Allows for various changes to be carried out on the current pen, depending on the value specified in Mode. Any line drawing commands, such as Arc and Rectangle, used after this command are affected. Modes available include: • Style. Allows for different line styles, for example DashDot. • Color. Color of pen drawings. • Width. Determines the thickness of the pen in pixels.
Parameters	Mode : WideString Value : WideString

Example	`Graphic.Pen('Style','DashDot');`
	`Graphic.Pen('Color','Blue');`
	`Graphic.Pen('Width','3');`

PenColor(Color)

Description	Enables the color of the current pen to be changed depending on the value specified. This is the same as using the Pen(Mode, Value) except it refers only to the color mode.
Parameters	Color: Text, widestring, color of pen to use.
Example	`Graphic.PenColor('green');`

PenWidth(Width)

Description	Enables the width of the current pen to be changed depending on the value specified. This is the same as using the Pen(Mode, Value) except it refers only to the width mode.
Parameters	Width: The width of pen to use, in pixels.
Example	`Example Graphic.PenWidth('3');`

PieShape(X1,Y1,X2,Y2,X3,Y3,X4,Y4)

Description	Draws a pie-shaped wedge on the image. The wedge is defined by the ellipse bound by the rectangle determined by the points (X1, Y1) and (X2, Y2). The section drawn is determined by two lines radiating from the center of the ellipse through the points (X3, Y3) and (X4, Y4).
Parameters	X1,Y1,X2,Y2,X3,Y3,X4,Y4 : integer.
Example	`Graphic.PieShape(10,10,25,25,30,30,40,40);`

Rectangle(X1,Y1,X2,Y2)

Description	Draws a rectangle. Specify the rectangle by giving the top left point at pixel coordinates (X1, Y1) and the bottom right point at (X2, Y2). The rectangle is drawn using the current pen width and color.
Parameters	X1,Y1,X2,Y2 : integer

Example	`Graphic.Rectangle(10,10,100,100);`
	`CRMGraphicBlock Object`

Resize(Width, Height)

Description	Specifies the new width and height of your image. Unlike ImageWidth and ImageHeight, the image is scaled to this new size. You should not set the ImageWidth and ImageHeight properties in the same block as they take precedence.
Parameters	Width, Height: Integers.
Example	`Graphic.Resize(150,100);`

Rotate(Number)

Description	Use this function to rotate an image by a specified number of degree points. The corners of a rotated image are colored in the current brush color.
Parameters	Number : Integer (0-360, corresponds to degrees).
Example	`Graphic.Rotate(90);`

RoundRect(X1,Y1,X2,Y2,X3,Y3)

Description	Use RoundRect to draw a rounded rectangle. The rectangle has edges defined by the points (X1,Y1), (X2,Y1), (X2,Y2), (X1,Y2), but the corners are shaved to create a rounded appearance. The curve of the rounded corners matches the curvature of an ellipse with width X3 and height Y3. The rounded rectangle is drawn using the current pen width and color.
Parameters	X1,Y1,X2,Y2,X3,Y3 : integer
Example	`Graphic.RoundRect(10,10,12,12,15,15);`

SaveAsGifs

Description	Determines whether an image should be stored as a GIF (256 colors) or a JPEG image (16m colors). If the server's display adapter is set to allow for 16m colors, this property is set to false by default, otherwise it is set to true.
Values	Boolean: True, false.
Example	`Graphic.SaveAsGifs=true;`

SaveAsJPG(text)

Description	Saves the current image in the JPEG image file format. Images are stored using 16 million colors. Note that JPEG images do not have an option for transparency or animation.
Parameters	Text : Widestring
Example	`Graphic.SaveAsJPG('c:\\cancel.jpg');`

TextOut(X, Y, Text, transparent=True/False)

Description	You use TextOut to write some text in your image. As an option, the text can be made transparent. By default, the text creates a blank rectangle where it is placed. It is written in co-ordinates specified in (X,Y) and is written in the current font color and size.
Parameters	X,Y : integer text : WideString transparent: true/false
Example	`Graphic.TextOut(10,10,'test',true);`

TextOutCenter(Left, Top, Right, Bottom, Text, Transparent, Ellipse)

Description	Writes text to your image in much the same way as TextOut but can center it in a rectangle area defined by the parameters passed to it. If Ellipse is set to true, it can also add '.' to the end of text if that text cannot fit into the rectangle without being truncated.
Parameters	Left,Top,Right,Bottom : integer, Text : WideString, Transparent=true/false, Ellipse=true/false
Example	`Graphic.TextOutCenter(10,10,100,30,'hello',true,true);`

Properties

Border

Description	Controls the thickness of the border around the image.
Values	Integer, default is 0.
Example	`Graphic.Border=1;`

Description

Description	Specifies the description of the image. For browser users with image loading switched off, the description specified in this parameter takes its place.
Values	Text : WideString.
Example	`Graphic.Description="Image description";`

hSpace

Description	Controls the horizontal space above and below the image.
Values	Integer, default is 0.
Example	`Graphic.hSpace=10;`

ImageHeight

Description	Specifies the height of the image. Dimensions in pixels. This is the height of the box in which the image is loaded.
Values	Integer, default is 0.
Example	`Graphic.ImageHeight=200;`

ImageWidth

Description	Specifies the width of the image. Dimensions in pixels. This is the width of the box in which the image is loaded.
Values	Integer, default is 0.
Example	`Graphic.ImageWidth=200;`

SaveAsGIF(text)

Description	Saves the current image in the GIF image file format. Images are stored using 256 colors.
Parameters	Text : Widestring
Example	`Graphic.SaveAsGif('c:\\test.gif')`

vSpace

Description	Controls the vertical space above and below the image.
Values	Integer, default is 0.
Example	`Graphic.vSpace=10;`

CRMGridColBlock Object

The CRMGridColBlock is used to set the properties of an individual column within a list. The GridColBlock object is related to the List Block but is a child of the CRM block. The properties that apply are similar to the fields available when adding columns to a Custom List within Administration | Customization | <Entity> | Lists.

Preceding Text:

```
ListBlock=CRM.GetBlock("companygrid")
```

```
ListBlock.AddGridCol("gridcolname")
```

```
ListBlock.GetGridCol("gridcolname")
```

Properties

Alignment

Description	Specifies the alignment of text within the column.
Values	Left, right, center. Default is left.

Example	The following example sets the text in the Source column to be right-aligned.
	`CaseListBlock = CRM.GetBlock('CaseListBlock');`
	`Source = CaseListBlock.AddGridCol('Case_Source');`
	`Source.AllowOrderBy = true;`
	`Source.Alignment = 'right';`
	`CRM.AddContent(CaseListBlock.Execute());`
	`Response.Write(CRM.GetPage());`

AllowOrderBy

Description	Specifies that the list can be sorted by the values in the column.
Values	Boolean: True, false.

Example	`GridCol.AllowOrderBy = True;`
	The following example adds a new column to a list that can sort the list.
	`CaseListBlock = CRM.GetBlock('CaseListBlock');`
	`FoundIn = CaseListBlock.AddGridCol('Case_FoundVer');`
	`FoundIn.AllowOrderBy = true;`
	`CRM.AddContent(CaseListBlock.Execute());`
	`Response.Write(CRM.GetPage());`

CustomActionFile

Description	This property is relevant when the JumpAction property is set to '430'. Enables a column to be hyperlinked to an ASP file. When an item in this column is selected the ASP file is called up, passing in the value of the field set in the CustomIdField property in the query string.
Values	The name of the ASP file.

Example	The following example sets the custom jump to the invoices ASP page.
	`list = CRM.GetBlock('CompanyGrid');`
	`g = list.GetGridCol('comp_name');`
	`g.JumpAction = 430;`
	`g.CustomIdField = 'comp_companyid';`
	`g.CustomActionFile = 'invoices.asp';`
	`CRM.AddContent(list.Execute("comp_name like 'eu'"));`
	`Response.Write(CRM.GetPage());`
	In the ASP file, the code can pick up on the unique Id as follows:
	`ThisComp = Request.QueryString('comp_companyid');`
	`Note that when you reference a value from the QueryString (or a form field),`
	`always reference the value rather than the object itself, as above or a=Request.QueryString("field")()`
	If there is a possibility of a QueryString field being duplicated, test its length and reassign the variable.

CustomIdField

Description	If CustomActionFile is set on a column, this property allows a value to be passed to the custom file when the column is selected. The value is passed on the query string in the form "FieldName=Value".
Values	The name of any field within the view for the list.
Parameters	None

Example	The following example sets the Id field for the custom jump to the comp_companyid field.
	`list = CRM.GetBlock('CompanyGrid');`
	`g = list.GetGridCol('comp_name');`
	`g.JumpAction = 430;`
	`g.CustomIdField = 'comp_companyid';`
	`g.CustomActionFile = 'test.asp';`
	`CRM.AddContent(list.Execute("comp_name like 'o%'"));`
	`Response.Write(CRM.GetPage());`

JumpEntity

Description	Allows a column to contain hyperlinked values to another entity summary screen. The entity must be relevant to the list, that is, the column of the entity must exist within the view or table on which the list is based. For example, it is possible to jump to an Opportunity from an Opportunity List but not from a Case List.
Values	Entity Name: Company, Person, Communication, Case, Address, Library, Notes, or Custom table.

Example	The following example sets the jump on the pers_firstname column.
	`PersonList=CRM.GetBlock("persongrid");`
	`GridCol=PersonList.GetGridCol("pers_firstname");`
	`GridCol.JumpEntity="person";`
	`CRM.AddContent(PersonList.Execute(''));`
	`Response.Write(CRM.GetPage());`

ShowHeading

Description	Specifies whether the heading should be shown on the column.
Values	Boolean: True, false. Default is true.

Example	The following example adds a column 'case_source' to the case list and disables the heading display.
	`CaseListBlock = CRM.GetBlock('CaseListBlock');`
	`Source =`
	`CaseListBlock.AddGridCol('Case_Source');Source.ShowHeading =`
	`false;`
	`Source.Alignment = 'LEFT';`
	`CRM.AddContent(CaseListBlock.Execute());`
	`Response.Write(CRM.GetPage());`

ShowSelectAsGif

Description	Specifies whether the values in the column should be shown as GIF images instead of Text. This is relevant if the column is a Select type and there are GIF files in the folder for each option on the list.
Values	Boolean: True, false. Default is false.
Example	`GridCol.ShowSelectAsGif = true;`

CRMListBlock Object

You use the CRMListBlock object to create and display lists. This block is a child the of CRMBlock and parent of the GridColBlock. You can link the list block to a search EntryGroup Block and use the search result as the argument for the List Block.

Preceding code:

```
ListBlock=CRM.GetBlock('companygrid');
```

Methods

AddGridCol(ColName, Position, AllowOrderBy)

Description	Enables you to add new grid columns dynamically to List blocks. The changes do not apply outside the ASP pages they are used in.
Parameters	ColName: Specifies the column to be added. This should be passed in as the name of the field. The field must be one that is relevant to the List block-it must be available within the table or view on which the List Block is based. Position (Optional): Numeric parameter that specifies the position in which to add the grid column. Passing in a value of 0 (zero), inserts the column into the first position and so on. Passing in-1 (minus one) adds the column to the end of the List. This is the default behavior if the Position is not specified. AllowOrderBy (Optional): Boolean parameter specifies if the column may be ordered or not. By default it is false.

Example	The following example adds a new orderable column 'comp_revenue' to the end of the company grid list.
	`MyList=CRM.GetBlock('CompanyGrid');`
	`MyList.AddGridCol('comp_revenue', -1, true);`
	`CRM.AddContent(MyList.Execute());`
	`Response.Write(CRM.GetPage());`

DeleteGridCol(ColName)

Description	This property deletes the specified column name from the list.
Parameters	ColName: The name of the column within the list that is to be deleted.

Example	The following example deletes the comp_website column in the company list
	`ListBlock = CRM.GetBlock("companygrid");`
	`ListBlock.DeleteGridCol("comp_website");`
	`CRM.AddContent(ListBlock.Execute());`
	`Response.Write(CRM.GetPage());`

Execute(Arg)

Description	Displays a list depending on the argument entered.
Parameters	Arg: Argument. If the argument is a string it is taken to be the WHERE clause of an SQL statement. If the argument is an entrygroup, the entry is used to form the SQL WHERE clause.

Example	Example 1: This examples lists all the companies of type 'Customer'.
	`ListBlock=CRM.GetBlock("companygrid");`
	`CRM.AddContent(ListBlock.Execute("comp_type='Customer'"));`
	`Response.Write(CRM.GetPage());`
	Example 2: This example uses the result of the search entrygroup as the argument for the list.
	`SearchContainer = CRM.GetBlock('Container');`
	`SearchBlock = CRM.GetBlock('PersonSearchBox');`
	`SearchContainer.AddBlock(SearchBlock);`
	`if (CRM.Mode == 2) {`
	`resultsBlock = CRM.GetBlock('PersonGrid');`
	`resultsBlock.ArgObj = SearchBlock;`
	`SearchContainer.AddBlock(resultsBlock);}`
	`CRM.AddContent(SearchContainer.Execute());`
	`Response.Write(CRM.GetPage());`

GetGridCol

Description	Returns a reference to the grid column specified. You can then set the individual properties for the column using the properties of the CRMGridColBlock Object.
Values	Returns a GridColBlock object.
Parameters	GridColName: The name of the column within the List that is required. If the column does not exist a nil object is returned.

Example	`Col = List.GetGridCol("pers_firstname");`
	The following example returns the company name column and enables the list to be ordered by this column.
	`ListBlock = CRM.GetBlock("companygrid");`
	`Column=ListBlock.GetGridCol("comp_name");`
	`Column.allowOrderby =true;`
	`CRM.AddContent(ListBlock.Execute());`
	`Response.Write(CRM.GetPage());`

Properties

CaptionFamily

Description	Enables the caption family on a list to be set so that valid translations can be added for the captions at the top of the list. When the CaptionFamily is set then translations are added to that family using the following codes: • Name: The name of the caption family • NoRecordsFound: The caption when there are no records in the list. • RecordsFound: The caption when there are records in the list. • RecordFound: The caption when there is one record in the list. • PreRecordsFound: The caption before the number when records are found. • PreRecordFound: The caption before the number when 1 record is found.
Value	String : The new family for the list.

Example	The following example changes the translation family for the company list to 'Campaigns'.
	`MyList = CRM.GetBlock('CompanyGrid');`
	`MyList.CaptionFamily="Campaigns";`
	`CRM.AddContent(MyList.Execute());`
	`Response.Write(CRM.GetPage());`

PadBottom

Description	Displays empty rows so that the number of rows set by RowsPerScreen always displays.
Values	Boolean: True or false. Default is true.

Example	`Listblock.PadBottom=true;`
	The following example disables the PadBottom property so that empty rows are not shown at the end of a list. If there are no rows to be displayed the column headers are still displayed.
	`List=CRM.GetBlock('companygrid');`
	`List.RowsPerScreen=8;`
	`List.PadBottom=false;`
	`CRM.AddContent(List.Execute());`
	`Response.Write(CRM.GetPage());`

prevURL

Description	You use this property if any of the columns in the List block have links to a main entity-Company, Person, Opportunity, Case, Lead, Solution.
Value	The property should be set to the URL for the ASP page which draws the list block.
Example	The following example tells the DLL that the previous dominant key was a custom page and also where to go back to. `&Key-1=iKey_CustomEntity&PrevCustomURL=PrevUrl`

RowsPerScreen

Description	Sets the number of rows displayed on each screen. You use this property to limit the number of rows displayed per screen, and then use the forward and back buttons to display next or previous screens. Note that each user has a Grid Size setting in their Preferences. This setting takes precedence over the RowsPerScreen setting (except where you are using the ListBlock in a CRMSelfService Object (page 8-95)).
Parameters	None

Example	The following example displays a list of companies eight rows at a time.
	`ListBlock = CRM.getBlock("CompanyGrid")`
	`ListBlock.RowsPerScreen = 8;`
	`CRM.AddContent(ListBlock.Execute());`
	`Response.Write(CRM.GetPage());`

SelectSql

Description	Changes the SQL used to select what items appear in the list. The property can only be used (and must be used) when the List block is not based on an existing grid or list. For example, you use this property when the List Block is a result of a call to CRM.GetBlock('List').
Value	String: A SQL SELECT clause in the form: 'SELECT * FROM table or view name' You should not put anything after the table or view name, the list takes care of the WHERE clause.

Example	The following example displays a list of the company names from the view vCompany.
	`NewList=CRM.GetBlock("list");`
	`NewList.SelectSql="Select * from vCompany";`
	`NewList.AddGridCol("Comp_Name");`
	`CRM.AddContent(NewList.Execute());`
	`Response.Write(CRM.GetPage());`

CRMMarqueeBlock Object

You use the MarqueeBlock object to add scrolling text, for example a news ticker, to a page. It is a child of the CRM Block object. The Marquee block reads from the Custom Captions table for news headlines and news story links and builds a scrolling display. You can control the direction of the scrolling, the positioning, the speed, and the style sheet used. The news content is maintained in CRM in Administration | Customization | Translations. The object provides a dismiss button which is overwritten when the news changes.

You call the Marquee block from an ASP page as follows:

```
var Marquee
Marq=CRM.GetBlock('marquee');
Marq.VerticalMinimum=150;
Marq.VerticalMaximum=150;
Marq.HorizontalMinimum=70;
Marq.HorizontalMinimum=70;
CRM.AddContent(Marq.Execute());
Response.Write(CRM.GetPage());
```

The block has six properties that can be modified:

- VerticalMinimuminteger
- VerticalMaximuminteger

Where VerticalMinimum and VerticalMaximum differ, the marquee moves vertically when scrolling.

- HorizontalMinimuminteger
- HorizontalMaximuminteger

Where HorizontalMinimum and HorizontalMaximum differ, the marquee moves horizontally when scrolling.

- StyleSheetstring, allows you to specify the style
- ScrollSpeedinteger

For horizontal scrolling, you set VerticalMinimum and VerticalMaximum to the same value. For vertical scrolling you set HorizontalMinimum and HorizontalMaximum to the same value.

The block expects the news headlines and news stories to be created using CRM translation handling. You access this in CRM through Administration | Customization | Translations. The caption family for news headlines should be news_headline and the link for a news story has a caption family of news_story. News stories must have the same caption code as their associated headline.

Properties

HorizontalMaximum

Description	Specifies the X boundary to the right of the screen for the marquee.
Values	Integer, default is 800

Example	`Marq=CRM.GetBlock('marquee');`
	`Marq.HorizontalMaximum=800;`

HorizontalMinimum

Description	The horizontal minimum is the starting point of the marquee on the X-axis.
Values	Integer, default is 0.

Example	`Marq=CRM.GetBlock('marquee');`
	`Marq.HorizontalMinimum=0;`

ScrollSpeed

Description	Determines the speed of the text as it moves on the screen.
Values	Integer, default is 120.

Example	`Marq=CRM.GetBlock('marquee');`
	`Marq.ScrollSpeed=200;`

StyleSheet

Description	Allows the alteration of the way the text is displayed. You provide a link to a cascading style sheet that includes the styles you require for the marquee.
Values	Text : WideString, default is DiagonalText

Example	`Marq=CRM.GetBlock('marquee');`
	`Marq.StyleSheet='NewStyle.css';`

VerticalMaximum

Description	Specifies the maximum vertical position of the marquee before returning to the VerticalMinimum value.
Values	Integer, default is 300

Example	`Marq=CRM.GetBlock('marquee');`
	`Marq.VerticalMaximum=100;`

VerticalMinimum

Description	Specifies the Y location of the marquee as it appears in the browser window, where 0 is at the top of the screen.
Values	Integer, default is 300

Example	`Marq=CRM.GetBlock('marquee');`
	`Marq.VerticalMinimum=100;`

CRMMessageBlock Object

You use the CRMMessageBlock object to send messages in SMS and e-mail format. It is a child of the CRM Object. The block can be included in ASP pages to show a simple e-mail form or to automate the message sending in response to a certain event. It can be used in visual and in hidden mode, see DisplayForm and Mode properties.

To initiate this block:

```
MessageBlock=CRM.GetBlock('messageblock');
```

Messaging Components and Configuration

Messaging needs the following additional system components:

- An e-mail server configured to redirect all incoming messages with a specified domain to the same folder (*).
- An SMS gateway referring to the folder mentioned above and the related mobile phone connection.

The following options must be set in Administration | E-mail And Documents | E-mail Configuration to allow the object to function:

- **Outgoing Mail Server (SMTP)**. The IP Address of the mail server.
- **SMTP Port**. Usually 25.
- **SMS Domain Name**. The mail domain used to hold the SMS messages, for example sms.domain.com.
- **SMTP Server For SMS Messaging**. For example 212.120.152.148 or any valid server name such as mail.sms.domain.com.
- **Use SMS Feature**. Set to Yes.
- **A proper sender address**. A valid e-mail address must be specified in the person profile.

The following points need to be noted:

- The message details (Recipients, CC, BCC, Subject, Body) are retrieved from the form content, if the DisplayForm property is set to true.
- The properties specified in the ASP page are defaults for the first value of the entry components of the form, unless the Mode property is set to 2 (send).
- The addresses specified in the form's fields can be phone numbers or e-mail addresses (separated by a comma or semicolon). The object automatically distinguishes the mode.
- The messages sent as SMS are truncated up to 160 characters, due to SMS format specifications.

Properties

DisplayForm

Description	Toggles visual/non visual mode. If the message contains errors, the form is displayed regardless of this parameter.
Values	Boolean: True, false. Default is true.

Example	The following example sends a message without displaying the form unless there are errors.
	`MailObj=CRM.GetBlock("messageblock");`
	`MailObj.Mode=2;`
	`MailObj.DisplayForm=false;`
	`CRM.AddContent(MailObj.Execute());`
	`Response.Write(CRM.GetPage());`

mAddressFrom/mNameFrom

Description	Sender Name and e-mail address.These properties are only used in Self Service mode, when the user is logged in and the name and the e-mail address are retrieved from the current user details.
Values	Any valid e-mail address.

Example	`Message.mAddressFrom='messagesender@domain.com';`
	`Message.mNameFrom='George Smith';`

mBody

Description	The content of the message. The body is truncated at 160 characters for SMS messages.
Values	Any text.

Example	`mailObj=CRM.GetBlock("messageblock");`
	`mailObj.mBody='This is where you put the content of the message';`
	`CRM.AddContent(mailObj.execute());`
	`Response.Write(CRM.GetPage());`

mErrorMessage

Description	After execute is invoked reports the detailed error string for the message just sent.
Values	Text (read only).

Example	The following example returns a value of true and displays the error message when the message is not sent successfully.
	`if(!mSentOK)`
	`{`
	`//if errors occurred then show the proper message`
	`CRM.AddContent('ERROR: '+mErrorMessage);`
	`}`
	`else`
	`{`
	`CRM.AddContent('Message Sent OK'());`
	`}`
	`Response.Write(CRM.GetPage());`

mSentOK

Description	Reports the status of the message sent after execute is invoked. It returns a value of true when the message has been sent successfully.
Values	Boolean: True, false (read only)

Example	The following example displays a message when the message is sent or an error if the delivery fails.
	`{`
	`if(!mSentOK)`
	`//if errors occurred then show the proper message`
	`CRM.AddContent('ERROR: '+mErrorMessage);`
	`}`
	`else`
	`{`
	`CRM.AddContent('Message Sent OK');`
	`}`
	`Response.Write(CRM.GetPage());`

mShowCC/mShowBCC

Description	Enables the display of carbon copy (CC) and blind carbon copy (BCC) fields in the graphical interface.
Values	Boolean: True, false. Defaults: mShowCC is true, mShowBCC is false.

Example	The following example displays the CC and BCC fields.
	`mailObj=CRM.GetBlock("messageblock");`
	`mailObj.mSubject='New Message';`
	`mailObj.mBody="This is where you put the content of the message.";`
	`mailObj.mShowCC=true;`
	`mailObj.mShowBCC=true;`
	`CRM.AddContent(mailObj.execute());`
	`Response.Write(CRM.GetPage());`

mSubject

Description	The subject of the message.
Values	String: Any text.

Example	`mailObj=CRM.GetBlock("messageblock");`
	`mailObj.mSubject='New Message';`
	`CRM.AddContent(mailObj.execute());`
	`Response.Write(CRM.GetPage());`

CRMOrgGraphicBlock Object

The organizational graphic is an implementation of the Graphic Block that is used for organizational charting. These diagrams can be drawn from data supplied to them from an ASP page or from data stored in a table. Other parameters can also be set to describe the look of the diagram. The most common use of these diagrams is to display an employee hierarchy for a company. Currently, all the parameters and data are set through the 'OrgTree' command. As with the Graphics Block, the organizational graphic is recreated every time it is requested and can therefore be based on real time data.

To initiate this block:

```
OrgGraph=CRM.GetBlock('orgchart');
```

Methods

OrgTree(Mode, Value)

Description	Currently, all properties and data are set through the 'OrgTree' command. It returns a string value as is required for some of the commands that may be passed to it. Some of the commands alter the appearance while others may be used to obtain counts on the branches in use.
Parameters	Mode : WideString, Value : WideString

Example	`OrgGraph.OrgTree('Add',',Top Level,True');`
	`OrgGraph.OrgTree('Add','Top Level,Child,True');`
	`OrgGraph.OrgTree('GetLevelCount', '1');`
	`OrgGraph.OrgTree('GetLargestLevelSize','');`
	`OrgGraph.OrgTree('Animated','False');`
	`OrgGraph.OrgTree ('FullBoxWidth','88');`
	`OrgGraph.OrgTree ('FullBoxHeight','50');`
	`OrgGraph.OrgTree('BoxWidth','40');`
	`OrgGraph.OrgTree('BoxHeight','25');`
	`OrgGraph.OrgTree('EntityIcon','c:\\person.bmp');`
	`OrgGraph.OrgTree('EntityImage','c:\\back.bmp');`
	`OrgGraph.OrgTree('BoxStyle','Square');`
	`OrgGraph.OrgTree('LineStyle','Ray');`

CRMPipelineGraphicBlock Object

The pipeline graphic is an implementation of the Graphic Block that includes extra functionality. You use the Pipeline Graphic to create cross-sectional diagrams that can represent data from an ASP page or data from a table. You use the parameters of this block to change the look and feel of the pipeline.

You can customize individual sections of the pipeline graphic to appear differently as the user selects them (by clicking on them). Similar to the Graphics Block, the Pipeline graphic is recreated every time it is requested and can therefore be based on real time data. It can also use all of the features of the Graphics Block.

The default size of the image created by the pipeline is set at 600 pixels wide and 100 in height, however it can be changed using the Graphics block's 'Resize' command.

To initiate this block:

```
MyObj=CRM.GetBlock('pipeline');
```

Methods

AddPipeEntry(Name, Value, Description)

Description	The easiest way to create a pipeline diagram is to build it up one section at a time using the AddPipeEntry command.
Parameters	Name: Text, widestring. The name of the section of the pipe that is shown in the Legend for the pipeline. Value: Integer, determines the size that this particular pipeline takes. Each section of pipe fills a percentage of the image width that is directly determined by its value. Description: Text, widestring. The text that appears when the user hovers over that section of pipe. Url: Text, widestring. The Web address (or ASP page) to link to should the user click on that section of the pipe.

Example	`MyPipe=CRM.GetBlock('pipeline');`
	`MyPipe.AddPipeEntry('Sold', 100,'100 items sold', 'http://www.mydomain.com');`
	`MyPipe.AddPipeEntry('Prospect', 40,'40 prospects', 'http://www.yahoo.com');`
	`CRM.AddContent(MyPipe.Execute());`
	`Response.Write(CRM.GetPage());`

ChooseBackGround(Value)

Description	Sets the background of the pipeline graphic.
Parameters	Value: Integer value for different background images. The images are loaded by default into the CRM images directory during installation. Default is white. For example: • 1= accpacblue.gif • 2 = accpacwhite.gif • 5 = listrow1gif • 8 = lightpurplemarblebright.gif • 14 = accpaccream.gif • 15 = listrow2.gif
Example	`Pipe.ChooseBackGround(8);`

PipelineStyle(Mode, Value)

Description	You can set various parameters of the PipelineGraphic block to change the appearance and size of individual sections of the pipeline. These parameters include adding gradients, displaying legends and adjusting diameters.
Parameters	Mode : WideString
Example	see below

```
MyPipe=CRM.GetBlock('pipeline');
MyPipe.AddPipeEntry('Sold', 100,'100 items sold', 'http://
www.CRM.com');
MyPipe.AddPipeEntry('Prospect', 40,'40 prospects', 'http://
www.yahoo.com');
MyPipe.PipelineStyle('Shape','Circle');
MyPipe.PipelineStyle('UseGradient','False');
MyPipe.PipelineStyle('Animated','False');
MyPipe.PipelineStyle('Selected','Sold');
MyPipe.PipelineStyle('SelectedWidth','10');
MyPipe.PipelineStyle('SelectedHeight','10');
MyPipe.PipelineStyle('PipeWidth','40');
MyPipe.PipelineStyle('PipeHeight','60');
MyPipe.PipelineStyle('ShowLegend','True');
CRM.AddContent(MyPipe.Execute());
Response.Write(CRM.GetPage());
```

Properties

Pipe_Summary

Description	Enables you to enter HTML text that displays to the right of a pipeline section when the section is selected. You can also use this property to display a legend/description of what is selected.
Parameters	Value: Text, HTML

Example	`Pipleline=CRM.GetBlock('pipeline');`
	`Pipe=Pipleline.Selected(1);`
	`Pipe.Pipe_Summary='<TABLE><TD CLASS=TABLEHEAD>Negotiating Selected (70)</TD></TABLE>';`

Selected

Description	Sets a section of the pipeline so that you can alter the style of that section when it is clicked on.
Parameters	Value: The number of the section that you are selecting.

Example	`Pipeline=CRM.GetBlock('pipeline');`
	`Pipeline.Selected(1);`

CRMQuery Object

The CRMQuery Object is used to enter and execute SQL statements against a known system database. The database can either be the system database or an external database connected to CRM. An external database must be made known to CRM before you can use the CRMQuery object with it.

You can use the CRMQuery object to perform more powerful queries than you can with the Record object. This object can be used to execute SQL statements that return results, for example, SELECT statements, or statements that don't return results, for example, DELETE statements. You can run any SQL statements, even INSERT and DROP TABLE. You use the SelectSql method to run SELECT statements and the ExeqSql method to run functions that don't return a result (such as DELETE).

Preceding code:

```
Query:=CreateQueryObj('Select * from tablename',
'databasename');
```

Note: The 'databasename' is an optional parameter. If it is not set, the default database is assumed.

Methods

ExecSql()

Description	Executes the SQL statement. You use this method to execute statements that do not return rows, for example DELETE, INSERT, UPDATE. You use SelectSQL to execute statements that do return rows (SELECT statements).
Parameters	None
Example	`Query.ExecSql();` The following example executes the SQL UPDATE statement. `var sql="UPDATE Company SET Comp_PrimaryUserID='"+AccountMgr+"' WHERE "+" Comp_CompanyId="+Values('Comp_CompanyId');` `CRM.ExecSql(sql);`

```
Query.ExecSql();
```

The following example executes the SQL UPDATE statement.

```
var sql="UPDATE Company SET Comp_PrimaryUserID='"+AccountMgr+"'
WHERE "+" Comp_CompanyId="+Values('Comp_CompanyId');
CRM.ExecSql(sql);
```

Next()

Description	Selects the next row or SELECT statement in the Query.
Parameters	None
Example	See below

```
Query.Next();
```

This example returns the next row in the query that displays company identifiers and names.

```
Query=CRM.CreateQueryObj("Select * from company", "");
Query.SelectSql();
while (!Query.eof)
{CRM.Addcontent(Query("comp_companyid") + " = " +
Query("comp_name") + "
");
Query.Next();}
Response.Write(CRM.GetPage());
```

NextRecord()

Description	Moves the specified query onto the next record.
Parameters	None
Example	`Query.NextRecord();`

Previous()

Description	Selects the previous row or SELECT statement in the query.
Parameters	None
Example	`Query.Previous();`

SelectSql()

Description	Executes the SQL. You use this method to execute statements that return rows (SELECT statements).
Parameters	None

Example	`Query.SelectSQL();`
	The following example displays the company identifier and name field from the selected SQL query until the end of the query.
	`Query=CRM.CreateQueryObj("Select * from company", "");`
	`Query.SelectSql();`
	`while (!Query.eof)`
	`{CRM.Addcontent(Query("comp_companyid") + " = " +`
	`Query("comp_name")+"`
	`");`
	`Query.NextRecord();}`
	`Response.Write(CRM.GetPage());`

Properties

Bof

Description	Returns whether you are at the beginning of the Query.

Example	The following example displays the company name if it exists, using the bof property to verify that it is not just at the beginning of the file.
	`comp = CRM.CreateQueryObj('select * from Company where`
	`Comp_CompanyId=12');`
	`comp.SelectSql();`
	`if ((!comp.eof) && (!comp.bof)) {`
	`CRM.AddContent(comp.comp_name);}`
	`else {CRM.AddContent('Company does not exist');}`
	`Response.Write(CRM.GetPage());`

DatabaseName

Description	When you use a Query object, the default database is the system database. You use this function to point it to another database.
Parameters	Name: String, database name.

Example	The following example changes the database to which the query is pointing.
	`Query=CRM.CreateQueryObj('Select * from company', 'crm');`
	`Query.SelectSQL();`
	`Query.DatabaseName(crm);`

Eof

Description	Returns whether or not you are at the last row of the query.
Example	The following example displays the company identifiers and name fields from the selected SQL query until the end of the query.

```
Query=CRM.CreateQueryObj("Select * from vCompany");
Query.SelectSql();
while (!Query.eof) {
Response.Write (Query("comp_companyid") + " = " +
Query("comp_name")+'
'); Query.NextRecord();
}
```

FieldValue

Description	Retrieves or sets individual fields in a query.
Parameters	FieldName: The name of the field that you want to retrieve.
Example	You don't need to specify the FieldValue property. These two examples are the same:

```
1) value=Query.FieldValue("somefield");
2) value=Query("somefield");
```

The following example displays the company identifier and the name field from the selected SQL query.

```
Query=CRM.CreateQueryObj("Select * from company", "");
Query.SelectSql();
while (!Query.eof)
{CRM.AddContent(Query("comp_companyid") + " = " +
Query("comp_name") + "
");
Query.NextRecord();}
Response.Write(CRM.GetPage());
```

RecordCount

Description	This property returns an integer value that is the number of records referred to by the Query object. This function can be used on a Query Object or a Record Object.
Parameters	None
Example	The following example displays a record count of all the records in the company table of the default database.

```
Query = CRM.CreateQueryObj("Select * from company");
Query.SelectSQL();
CRM.AddContent("There are " +Query.RecordCount+ " records.");
Response.Write(CRM.GetPage());
```

SQL

Description	Sets the SQL statement for this Query. The SQL statement is usually passed in when the object is created, but you can change it using this property. The SQL is not executed until you call one of the execute methods, SelectSql or ExecSql.

Example	This example resets the SQL SELECT statement to query the person table instead of the company table.
	`Query=CRM.CreateQueryObj("Select * from company", "");`
	`Query.SQL="Select * FROM person";`
	`Query.SelectSql();`
	`while (!Query.eof)`
	`{CRM.AddContent (Query("pers_personid")+" = "`
	`+Query("pers_lastname") +"`
	`");`
	`Query.NextRecord();}`
	`Response.Write(CRM.GetPage());`

CRMRecord Object

The CRMRecord Object represents records in a table. This object is an enumerator that returns all the specified fields in a table.

The Record object contains a higher-level understanding of the columns than a query object. You use the properties and methods of this object to manipulate information in columns and save any edits.

You use the CRM Object's CreateRecord or FindRecord methods to return the record that you manipulate using the Record object. In all of the examples that follow, code (similar to the examples below) is used to create the record object:

```
record=CRM.CreateRecord("cases");
```

```
record=CRM.FindRecord("cases","case_caseid=20");
```

Methods

FirstRecord()

Description	Moves the record to point to the first record that matched the SQL passed in when the record object was created. Note, when the record object is created, it automatically points at the first record, so you only need to use this if you want to reset it.

Example	The following example displays companies starting with the letter 'o' and writes out the first record.
	`o = CRM.FindRecord("company","comp_name like 'o%'");`
	`while (!o.eof) {`
	`CRM.AddContent(o.comp_name+'`
	`');`
	`o.NextRecord()}`
	`o.FirstRecord();`
	`CRM.AddContent('The first company is '+o.Comp_Name);`
	`Response.Write(CRM.GetPage());`

NextRecord()

Description	Returns the next record (if any).

Example	This example displays a list of people in the person table-first and last names.
	`People = CRM.FindRecord('Person','Pers_Deleted is null');`
	`while (!People.Eof) {`
	`CRM.AddContent(People.Pers_FirstName+'`
	`'+People.Pers_LastName+'`
	`');`
	`People.NextRecord(); }`
	`Response.Write(CRM.GetPage());`

RecordLock

Description	Locks the current Record object. If the record is already locked (someone else is using it), an error message is returned. Note that locking is usually automatically handled by the Container blocks. The RecordLock function should only be used when the standard container locking functionality has been disabled (by setting the relevant Container block property CheckLocks to false).
Parameters	None: If the record is locked, outputs an error message.

Example	The following example locks the record. It displays an error and places the record in view mode if the record cannot be locked-that is, if it is already locked by somebody else.
	`var r=CRM.FindRecord('company','comp_companyid=30');`
	`CompBlock = CRM.GetBlock('CompanyBoxLong');`
	`CompBlock.CheckLocks = false;`
	`if (CRM.Mode == 1)`
	`{e = r.RecordLock();`
	`if (e != '')`
	`{CRM.Mode = 0; // keep in view mode`
	`CRM.AddContent(e+'`
	`');}`
	`}`
	`CRM.AddContent(CompBlock.Execute(r));`
	`Response.Write(CRM.GetPage());`

SaveChanges()

Description	Saves any changes made to the current record, in the database. You must call this method to save changes to the database. Note that SaveChanges() refreshes the RecordObject to point back to the beginning of the record set selected. Therefore SaveChanges cannot be used on a RecordObject where the same RecordObject is being used in the condition in a while loop. See Example 2 below for workaround for using SaveChanges() in a loop.

Example	`Record.SaveChanges();`
	This example adds and saves a new record to the company table and displays in a list.
	`Comp = CRM.CreateRecord('company');`
	`Comp.item('comp_Name') = '4D Communications International';`
	`Comp.SaveChanges();`
	`block=CRM.GetBlock("companygrid");`
	`CRM.AddContent(block.execute(''));`
	`Response.Write(CRM.GetPage());`

Example 2	`var companies = CRM.FindRecord("company","comp_name like 'Gate%'");`
	`while (!companies.eof)`
	`{`
	`Response.Write(companies('comp_name')+'')`
	`Response.Flush();`
	`var company = CRM.FindRecord('company', 'comp_companyid=' +`
	`companies.comp_companyid);`
	`company.comp_type = 'Member';`
	`company.SaveChanges();`
	`companies.NextRecord();`
	`}`
	`Response.Write('End');`

SaveChangesNoTLS()

Description	Saves any changes made to the current record in the database, but does not trigger any Table Level scripts that exist for the table that is being updated.

Example	`Record.SaveChangesNoTLS();`
	This example adds and saves a new record to the company table but does not call the company table level script.
	`Comp = CRM.CreateRecord('company');`
	`Comp.item('comp_Name') = '4D Communications International';`
	`Comp.SaveChangesNoTLS();`
	`block=CRM.GetBlock("companygrid");`
	`CRM.AddContent(block.execute(''));`
	`Response.Write(CRM.GetPage());`

SetWorkflowInfo(vWorkflowName, vWorkflowState)

Description	Allows you to save a new record into a workflow tree. This function works when the Record object is a result of a CreateRecord call. Note that this function can also be used when the Record object is being used as the ArgObj of an EntryGroup block, or passed into the Execute function of an EntryGroup block.
Parameters	vWorkflowName - This is the description of the workflow into which you want the record to be saved, that is, the value that was entered as the workflow description when the workflow was created. vWorkflowState - This is the name of the state in the relevant workflow at which the record is to be saved, that is, the State Name value entered when the state is created.

Example	The following example creates a new opportunity and makes it part of the 'SalesOpportunityWorkflow' workflow in the state 'In Progress'. When the opportunity is viewed, the valid actions for that state are then available.
	`NewOppo = CRM.CreateRecord("Opportunity");`
	`NewOppo.SetWorkflowInfo("SalesOpportunity Workflow","Lead");`
	`NewOppo.Item ("oppo_description") = "My new Oppo";`
	`NewOppo.SaveChanges();`

Properties

DeleteRecord

Description	This property is Boolean which flags a record for deletion. If this property is set to True, then when the SaveChanges method is called, the record is deleted. This is known as a soft delete. Deletes are not cascaded to all child records. This means that there may be some orphaned records. To avoid this, run a version of this query using either the QueryObject or the RecordObject. `select bord_name, bord_companyupdatefieldname from custom_tables where bord_companyupdatefieldname is not null` This will help you identify the child/orphaned records.
Values	Boolean: True or false.

Example	`DeleteRecord=true;`
	The following example deletes a record in the company table.
	`Comp = CRM.FindRecord('company', "comp_name='Eurolandia'");`
	`Comp.DeleteRecord=true;`
	`Comp.SaveChanges();`

Eof

Description	Tests to see if a loop has reached the last record. It returns true if you are at the last record or if there are no records. If there is only one record this becomes true after one call to NextRecord.

Example	This example retrieves the next record if the last record has not been retrieved
	`(eof).`
	`while (!record.eof) {`
	`record.NextRecord();}`

IdField

Description	Returns the name of the identification (primary key) field for the current table of the record object. This is normally the first fieldname in a table.

Example	The following example returns the Id field of the company called 'Design Right Inc.'.
	`Comp = CRM.FindRecord('company', "comp_name='Design Right Inc.'");`
	`var idname=Comp.IdField;`
	`CRM.AddContent(Comp.item(idname));`
	`Response.Write(CRM.GetPage());`

Item

Description	Returns or sets the given fieldname.
Parameters	FieldName: The name of the field you want to retrieve or add to the column in the table.

Example	You are not required to specify the item property for record objects. These two examples are the same:
	`record.Item("item");`
	and
	`record("item");`
	The following example creates a new record in the company table, names the company '3D Communications International', and displays it in a company list.
	`Comp = CRM.CreateRecord('company');`
	`Comp.item('comp_Name') = '3D Communications International';`
	`Comp.SaveChanges();`
	`block=CRM.GetBlock("companygrid");`
	`CRM.AddContent(block.execute(''));`
	`Response.Write(CRM.GetPage());`

ItemAsString

Description	Returns the field value as a string. The Item property returns the item in its native format. The ItemAsString property uses metadata to convert the item to a string.
Parameters	FieldName: The field you want returned.

Example	The following example finds and displays the name of the user assigned to a case from their userid.
	`Case = CRM.FindRecord('cases', "case_assigneduserid=5");`
	`CRM.AddContent(Case.itemasstring("case_assigneduserid"));`
	`Response.Write(CRM.GetPage());`

OrderBy

Description	This is a string that makes up the fieldname or names under which the record object is ordered. The value contained in OrderBy is used to build up an SQL statement for the record. This means that you can use parameters such as 'ASC' or 'DESC' for ascending and descending orders in the statement.

Example	The following example orders a record in descending order by people first name, then last name.
	`People = CRM.FindRecord('Person','Pers_Deleted is null');`
	`People.OrderBy = 'Pers_LastName, Pers_FirstName';`
	`while (!People.Eof) {`
	`CRM.AddContent(People.Pers_FirstName+'`
	`'+People.Pers_LastName+'`
	`');`
	`People.NextRecord();`
	`Response.Write(CRM.GetPage());`
	`}`

RecordCount

Description	This property returns an integer value, that is the number of records referred to by the object. This function can be used on a Record Object or a Query Object.

Example	The following example displays the number of current system users.
	`Users = CRM.FindRecord('users','');`
	`CRM.AddContent("There are " +Users.RecordCount+ " system`
	`users.");`
	`Response.Write(CRM.GetPage());`

RecordID

Description	Returns the identifier for the current record. A unique identifier is created automatically for each record when the record is created.

Example	To find the identifier of the current record
	`Response.Write(Record.RecordID);`
	The following example displays the name and identifier of the current record.
	`Record=CRM.FindRecord("company","");`
	`CRM.AddContent(Record("comp_name"));`
	`CRM.AddContent(Record.RecordID);`
	`Response.Write(CRM.GetPage());`

CRMSelfService Object

The CRMSelfService object is similar to the CRM Object, but allows access to the CRM database, and to many methods of the CRM object, from outside the CRM application. You could use it in a web application to allow visitors to your website have access to, and interaction with, some aspects of your CRM system. For example, visitors to your website might be allowed to log cases, or to update their addresses or other contact information directly. These visitors do not have to be CRM Users, but may be "People" in your CRM database.

There is a sample Self Service application that can be installed as part of CRM setup (you CRM license key must include Self Service). More details on the Self Service sample application can be found in the Self Service Guide.

Although Self Service is a COM based API like the main ASP API for building application extensions it is actually separate and the usage of the blocks can be quite different. Because the Self Service environment lacks a 'logon' that generates a CRM SID (Session ID) or context you can't use any API objects/methods that rely on this for building URLs (for example eWare.Button(), eWare.GetTabs() and eWare.URL() cannot be used).

For example, this will result in an error:

```
CRM.AddContent(myBlock.Execute(Arg));
Response.Write(CRM.GetPage())
```

```
Response.Write(myBlock.Execute(Arg));
```

Self Service Method and Property Differences

The following table lists methods and properties that are unique to, or have special application in, the Self Service object:

Methods and properties that are unique to, or have special application in, the Self Service object:

- Init(QueryString, ContentString, Cookie) (page 8-97)
- EndSSSession(QueryString, ContentString, Cookie) (page 8-97)
- Authenticated (page 8-98)
- AuthenticationError (page 8-98)
- VisitorInfo (page 8-99)
- RowsPerScreen (page 8-76) (see note below)

The following CRM objects and methods are not available in Self Service:

- AddContent(Content) (page 8-17)
- GetCustomEntityTopFrame(EntityName) (page 8-20)
- GetPage() (page 8-20)
- SetContext(EntityName, EntityID) (page 8-21)
- FastLogon (page 8-27)
- Button (page 8-23)
- GetContextInfo(Context, FieldName) (page 8-24)
- GetTabs(TabGroup) (page 8-25)
- Logon(LogonId, Password) (page 8-26)
- Url(Action) (page 8-26)
- Email Object (page 8-12)
- AddressList Object (page 8-9)
- MailAddress Object (page 8-15)
- AttachmentList Object (page 8-11)
- Attachment Object (page 8-10)
- MsgHandler Object (page 8-15)
- CRMGraphicBlock Object (page 8-59)
- CRMChartGraphicBlock Object (page 8-33)
- CRMOrgGraphicBlock Object (page 8-82)
- CRMPipelineGraphicBlock Object (page 8-82)

RowsPerScreen Property in Self Service

When using a CRMListBlock Object (page 8-72) on your Self Service page, you can use the RowsPerScreen property to set the number of rows that will appear in the ListBlock grid. Unlike normal CRM ListBlocks, the RowsPerScreen property of Self Service ListBlocks is not affected by any user's Grid Size preference.

This example creates a list of cases related to the current Self Service Visitor, and sets the row count to 22:

```
<%
myListBlock=CRM.GetBlock("sscaselist");
myListBlock.RowsPerScreen = 22;
Response.Write(myListBlock.Execute("case_primarypersonid="+CRM.VisitorInfo('Pers_
PersonID')));
%>
```

Note on Instantiating the CRMSelfService Object

For legacy reasons, the CRMSelfService Object is instantiated as:

```
CRM = Server.CreateObject("eWare.eWareSelfService");
```

Note that in the Self Service demo site that ships with CRM, the CRMSelfService Object is instantiated as:

```
eWare = Server.CreateObject("eWare.eWareSelfService");
```

And so, in the demo site, the code will reference the object as eWare, for example:

```
record=eWare.FindRecord("cases","case_caseid="+caseid);
```

Methods

EndSSSession(QueryString, ContentString, Cookie)

Description	This method ends the Self Service session and resets the CRM cookies at the end of the session.
Parameters	QueryString: Pass in the current page's querystring, Request.Querystring. ContentString: Pass in the current page's form string, Request.Form. Cookie: Pass in a reference to the CRM cookies object,Request.Cookies("CRM").

Example	The following example ends the Self Service session, requests the CRM cookies, and displays a goodbye message to the user.
	`CRM.EndSSSession(Request.Querystring, Request.Form, Request.Cookies("CRM"));`
	`Response.Write("Goodbye "+CRM.VisitorInfo("visi_FirstName")+" "+CRM.VisitorInfo("visi_LastName"));`

Init(QueryString, ContentString, Cookie)

Description	The CRM SelfService object must be initialized and connected to the database before it can be used. The Init method initializes the Self Service session and CRM cookies. If you have installed the CRM Self Service Demo site, you will see that the Init method is called in the ewaress.js file. This file can be included at the top of each of your Self Service ASP pages, for example: `<!-- #include file ="ewaress.js" -->` Note that there is also an Init method for normal CRM sessions, but it is internal to CRM.
Parameters	QueryString: The name of the query string. ContentString: The name of the content string, usually a form. Cookie: The name of the initial CRM cookies.

Example	The following example initializes the CRM Self Service object and CRM cookies.
	`CRM = Server.CreateObject("eWare.eWareSelfService");`
	`CRM.init(Request.Querystring,Request.Form,Request.Cookies("CRM"));`
	`Response.Expires=-1;`

Properties

Authenticated

Description	Returns true if the current user is authenticated.
Parameters	None

Example	
	```
	if (CRM.Authenticated)
	{
		//do some action only for authenticated users
	}
	```
	 The following example enables a user who is authenticated to access to a member menu, and users that are not authenticated to be taken back to an index page.
	```
	if (CRM.Authenticated)
	{
		//This could be any function
		getmembermenu();
	}
	else
	{
		Response.Redirect("index.asp");
	}
	```

AuthenticationError

Description	If there is an authentication error, the text is given by this property. The error displayed is the SQL reason or IIS reason for the error.
Parameters	None

Example	
	```
	if (CRM.Authenticated)
	{
		//perform action for authenticated users
	}
	else
	{
		Response.Write('You are not a valid user' +
	CRM.AuthenticationError);
	}
	```

VisitorInfo

Description	Returns or sets the value associated with a given key for the current authenticated visitor. The key can be either a column on the visitor table, beginning with 'Visi', or any text.
Parameters	Key: Either a column in the visitor table or a string variable.
Example	CRM.VisitorInfo("Visi_FirstName") = firstname; `CRM.VisitorInfo("haircolor") = "blue";` The following example allows access to any visitor user who has been authenticated and has filled in the notification criteria. `if((CRM.Authenticated)&&(CRM.VisitorInfo("Visi_NotificationCriteria")!="")) { //This could be any function getmembermenu(); };`

```
CRM.VisitorInfo("haircolor") = "blue";
```

```
if((CRM.Authenticated)&&(CRM.VisitorInfo("Visi_
NotificationCriteria")!=""))
{
       //This could be any function
       getmembermenu();
};
```

CRMTargetListField Object

Fields to be included in the Target List. The actual field names in the CRM database need to be specified.

Note that in version 6.0 and above, target lists are now referred to as "groups." However, to ensure that legacy code continues to work with new installations, the older term, "target lists," is maintained in the API terminology.

Properties

DataField

Description	The name of the field to be displayed on the Target Lis
Value	String.
Example	See CRMTargetLists Object (page 8-99).

CRMTargetLists Object

Used for creating and saving a Target List in conjunction with CRM TargetListFields and CRM TargetListField. The target list must be based on a Company, Person, or Lead.

Note: In release of version 6.0 and above, target lists are referred to as "groups." However, to ensure that legacy code continues to work with new installations, the older term, "target lists", is maintained in the API terminology.

Methods

Save()

Description	Saves the target list. If the TargetListID property was set to zero, then a new target list is saved, otherwise the target list specified by the TargetListID property is updated.
Parameters	n/a
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

Include(ATargetID)

Description	Includes a target in the Target List.
Parameters	ATargetID. Integer.
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

Exclude(ATargetID)

Description	Excludes a target from the Target List.
Parameters	ATargetID. Integer.
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

Retrieve()

Description	Retrieves a target from the Target List.
Parameters	n/a
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

Properties

TargetListID

Description	The identifier of the target list.
Value	Integer.
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

Category

Description	The category of the target list.
Value	String.
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

Name

Description	The name of the target list.
Value	String
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

Description

Description	The description of the target list.
Value	String
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

ViewName

Description	The view used by the target list.
Value	String
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

Fields

Description	Link to list of display fields. Read-only.
Value	CRMTargetListFields
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

OrderByFields

Description	Link to list of order by fields. Read-only.
Value	CRMTargetListFields
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

WhereClause

Description	The Where Clause used to filter the list of targets. Must set this if creating or modifying a target list.
Value	String
Example	See Example: Creating and Saving a Target List (page 8-102) and Example: Retrieving a Target List (page 8-103).

Example: Creating and Saving a Target List

```
// Shows an example of creating and saving a target list
// All steps are compulsory and should be in this order
<!-- #include file ="sagecrm.js" -->
<%
TargetBlock = CRM.TargetLists; // Get the TargetBlock COM Object from the CRM base
object
TargetBlock.TargetListID = 0; // Set the id to zero, to indicate a new target list
TargetBlock.Category = "Person"; // Set the category. Other valid categories are
Company and Lead
TargetBlock.Name = "COM List 1"; // Set the name of the target list, should be
unique
TargetBlock.ViewName = "vTargetListPerson"; // Set the view to be used
TargetBlock.WhereClause = "Addr_City = N'London'"; // If required, then specify a
where clause.
// You must specify at least one display field. All fields must be returned by the
view.
TargetField = TargetBlock.Fields.New(); // Create a new display field
TargetField.DataField = "Comp_Name"; // Specify its database fieldname
TargetField = TargetBlock.Fields.New(); // Create a second display name, optional
TargetField.DataField = "Pers_LastName";
TargetField = TargetBlock.Fields.New(); // Create a third display field, optional
TargetField.DataField = "Pers_FirstName";
// Add more fields as desired. You may add order by fields to sort the target list
TargetField = TargetBlock.OrderByFields.New(); // Create a new order by field
TargetField.DataField = "Pers_LastName"; // Specify its database fieldname
TargetQuery = TargetBlock.Retrieve(); // Create and return the target list based
on the above settings
// This demonstrates cycling through the returned targets, and setting every tenth
target to be excluded
while (!TargetQuery.EOF)
{
I = 1;
while ((!TargetQuery.EOF) &amp;&amp; (I < 10))
{
TargetQuery.Next();
I++;
}
if (!TargetQuery.EOF)
{
j = TargetQuery.FieldValue("Pers_PersonID");
TargetBlock.Exclude(j);
}
}
// For the moment, we always return to the Actions page whether successful or not.
// 580 is the action number to go back to the target list browser page
// 585 is the action number to go back to the target list actions page
if (TargetBlock.Save()) // Save the target list
```

```
{
Response.Redirect(CRM.URL(580));
}
else
{
Response.Redirect(CRM.URL(580));
}
%>
```

Example: Retrieving a Target List

```
// This shows an example of retrieving a target list,cycling through the targets
// and marking any excluded targets as being included
<!-- #include file ="sagecrm.js" -->
<%
TargetBlock = CRM.TargetLists;
// Get the TargetBlock COM Object from the CRM base object
TargetBlock.TargetListID = Request.QueryString("Key25");
// Set the id that we want to look for
TargetQuery = TargetBlock.Retrieve();
// Retrieve the target list
while (!TargetQuery.EOF)
{
if (TargetQuery.FieldValue("DData_ShortStr") == "Excluded") // If this target is
excluded, then
{
TargetBlock.Include(TargetQuery.FieldValue("Pers_PersonID")); // Include this
target
// This particular target list is a Person target list
// If a Company target list, then use the Comp_CompanyID field
// If a Lead target list, then use the Lead_LeadID field
}
TargetQuery.Next(); // Move to next target
}
// For the moment, we always return to the Actions page whether successful or not.
// 580 is the action number to go back to the target list browser page
// 585 is the action number to go back to the target list actions page
if (TargetBlock.Save())
{ // Save the target list
Response.Redirect(CRM.URL(585));
}
else
{
Response.Redirect(CRM.URL(585));
}
%>
```

CRMTargetListFields Object

This Object is a container for a list of CRMTargetListField Objects. There are two instances of this object, one for the display fields, and the other for the order fields. Note that in version 6.0 and above, target lists are now referred to as "groups."

However, to ensure that legacy code continues to work with new installations, the older term, "target lists," is maintained in the API terminology.

Methods

New(CRMTargetListField)

Description	Creates and returns a new field.
Value	CRMTargetListField
Example	See CRMTargetLists Object (page 8-99).

Delete(Index)

Description	Deletes the field specified by the passed index.
Value	Index.
Example	See the CRMTargetLists Object (page 8-99) page for an example.

Properties

Parent

Description	A pointer to the CRMTargetLists object.
Value	Read only. Returns a CRMTargetLists object.
Example	See CRMTargetLists Object (page 8-99).

Count

Description	The new of fields in the list.
Value	Integer (read only)
Example	See the CRMTargetLists Object (page 8-99) page for an example.

Item

Description	Returns the field specified by the Index. The index is an integer.
Value	Read only. Returns a CRMTargetLists object.
Example	See CRMTargetLists Object (page 8-99).

Chapter 9: Web Services

In this chapter you will learn how to:

- Describe CRM Web Services.
- Discuss how to set up Web Services.
- Get an overview of objects and functions.
- List Web Services functions.
- List Web Services objects.
- Describe the CRM RecordType object.
- Get an overview of selection fields in Web Services.
- Understand Web Services examples.
- Create SOAP requests.

Introduction to 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, Cases, Quotes and Orders.

Please refer to List of Web Services Objects (page 9-7) for information more details on inserting and updating Quote and Order Item fields.

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

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

General Overview of 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 Services Capabilities

In Sage CRM, the ability to manipulate records remotely affords the following 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 (Enterprise Resource Planning) systems, with the Sage CRM server or hosted system.
- 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.

Setting Up CRM Web Services

Prerequisites

To set up Web Services, you will need to have the following installed on the server:

- CRM with a standard license key
- MSXML 4 Service Pack 2 (can be downloaded from Microsoft website)

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) and Microsoft Visual C# 2005 Express Edition.

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.
2. Specifying Web Services configuration settings.
3. Accessing the WSDL file.
4. Preparing the request and submitting it to Web Services.
5. Handling the response—returned values or error message.

Steps 1 to 2 are described below. For information on preparing the request and handling the response see Objects and Functions Overview (page 9-5) and the Web Services Examples.

Web Services User Setup

Before Web Services can be accessed, a user account 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.
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.

Note: The Field Level Security feature affects which 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 the System Administrator Guide.

Specifying Web Service Configuration Settings

To access web service configuration settings select Administration | System | Web Services.

The table below explains the fields on the Web Services 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 number you enter 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. you can find the URL at which to view the WSDL file by going to **Main Menu | System Help | Account Update | Web Service Connection String**.
Enable Web Services	Set to Yes to enable the Web Services functionality. Set to No to disable Web Services. To enable or disable Web Services for an individual Table or Entity go to **Administration | Customization** | [*Entity/Table Name*] | **External Access** and set the **Web Services** field to Yes to enable or No to disable.
Dropdown Fields As Strings In WSDL File	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 Objects and Functions for more details.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".
Send And Return All Dates And Times In	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.

Field	Description
Universal Times	
Accept Web Request From IP Address	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.
Force Web Service Log On	If the connection between the web service client and the service is unexpectedly broken, that client remains logged onto 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.

Recommended Configuration Settings

These are the recommended settings to allow your client to access the Web Service during development:

1. Set the **Enable Web Services** field to **Yes**.
2. Select **Yes** from the **Make WSDL To All** field.
3. Set the **Force Webservice Log On** field to **Yes**.

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

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 from the client application, open the CRMWebService.WSDL file at your install address. For example:

http://CRM*servername*/*CRMinstallname*/eWare.dll/webservice/webservice.wsdl

On SageCRM.com, you can find the URL to view the WSDL file by going to **Main Menu | System Help | Account Update | Web Service Connection String**. The URL will look something like this:

https://[*region*].sagecrm.com/[*username1234*]/eware.dll/webservices/CRMwebservice.wsdl

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.

For example, if you are using Microsoft Visual Studio to create a client application, your Visual Studio project should contain a Web Reference to e.g.
http://CRM
servername
/
CRMinstallname

/eWare.dll/webservice/webservice.wsdl
https://[*region*].sagecrm.com/[*username1234*]/eware.dll/webservices/CRMwebservice.wsdl.

When you add the reference in Visual Studio, the main pane lists the methods available from the web service.

If you name the service **CRMWebServices** then a new folder called CRMWebServices, containing the files webservice.discomap and webservice.wsdl, is 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.

Note: In Visual Studio 2008, to add a Web Reference you must select **Add Service Reference | Advanced | Add Web Reference**.

Objects and Functions Overview

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, Sage CRM Web Services 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.

See List of Web Services Functions (page 9-5) for a full list.

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.

See List of Web Services Objects (page 9-7) for a full list, and see also The CRM RecordType Object (page 9-10) and Selection Fields in Web Services (page 9-11).

List of Web Services 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

Function	Description
	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 an object of type *aisid* (see List of Web Services Objects (page 9-7)). 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.
queryidnodate	Returns an object of type *aisid* (see List of Web Services Objects (page 9-7)). 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!.

Function	Description
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, pricinglist.
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 CRM RecordType Object (page 9-10).
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 CRM RecordType Object (page 9-10).
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 CRM RecordType Object (page 9-10).
getallmetadata	Returns a list of fields associated with all tables along with some type information.
getversionstring	Returns the version of CRM. For example, Version 6.2.

List of Web Services Objects

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.See The CRM RecordType Object (page 9-10) for more.
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.

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.
quotes	This Object represents the Quotes entity in CRM.
orders	This Object represents the Orders entity in CRM.
quoteitem	This Object represents the quote lineitems entity in CRM.
orderitem	This Object represents the order 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.

Inserting and Updating Quote and Order Items

When inserting and updating fields for quote and order items, note that different line item types require certain fields. The web service will create an exception if they are not found.

When inserting a new standard line item, the following fields are required:

- orderquoteid
- opportunityid
- lineitemtype (either ‘i’, ‘f’ or ‘c’)
- productid

- uomid
- quantity
- quotedprice

When inserting a new free text line item, the following fields are required:

- orderquoteid
- opportunityid
- lineitemtype (either ‘i’, ‘f’ or ‘c’)
- description
- quantity
- quotedprice

When inserting a new comment line item, the following fields are required:

- orderquoteid
- opportunityid
- lineitemtype (either ‘i’, ‘f’ or ‘c’)
- description

When updating a standard line item, the following fields require a value:

- quantity
- quotedprice
- uomid

When updating a free text line item, the following fields require a value:

- description
- quantity

When updating a comment line item, the following fields require a value:

- description

The following two fields are can not be updated, and will create an exception:

- linetype
- orderquoteid

In addition, certain fields are calculated or overridden by CRM in the web service code, the values that the user passes into them will be ignored. These fields are:

- quotedpricetotal
- listprice
- discount
- discountsum

The CRM RecordType 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) sobject, 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.

```
Please refer to Developer Help files for code sample.
```

```
Private static void CallQueryRecordOnCompanyEntity()
{
       String companyid =  ReadUserInput("Please enter a company name: ");
       Queryrecordresult aresult = Binding.queryrecord("comp_companyid,address","comp_
name='compo1'","company","comp_companyid");
}
```

Selection Fields in Web Services

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 tables below list the CRM selection fields. 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 are returned in the default system language.

```
<s:simpleType  name="case_problemtype">
<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>
```

List of Selection Fields

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_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_problemtype
- case_productarea
- case_solutiontype
- case_source
- case_stage

- case_status
- case_targetver

Address and Product Selection Fields

- addr_country
- prod_uomcategory

Using GetDropDownValues

Use the *getdropdownvalues* function. See List of Web Services Functions (page 9-5) to get the list of the drop-down fields in a table and the list of values that CRM expects for that field.

This is an example in C# of a function to populate a ComboBox with selection values from a given field.

```
private void LoadDropDowns(string entity, string fieldname, ComboBox controlname,
WebService WS)
{
       dropdownvalues[] DropDowns;
       DropDowns = WS.getdropdownvalues(entity);
       controlname.Items.Clear();
       for (int i = 0; i < DropDowns.Length; i++)
       {
               if (DropDowns[i].fieldname == fieldname)
               {
                       for (int x = 0; x < DropDowns[i].records.Length; x++)
                       {
                               controlname.Items.Add(DropDowns[i].records[x].ToString());
                       }
               }
       }
       controlname.SelectedIndex = 0;
}
```

To use the function to display the comp_sector selection values in a ComboBox called 'comboSector' (where the web service object is called oWebService):

```
LoadDropDowns("company", "sector", comboSector, oWebService);
```

Introduction to Web Services Examples

There are Sample SOAP requests and a suite of Visual Studio C# application examples to assist in the development of Web Services applications for Sage CRM.

Sample SOAP requests can be found here: Sample SOAP Requests (page 9-15)

C# applications are provided with your CRM installation and can be found in the *WWWRoot\Examples\Webservices\* folder (usually *C:\Program Files\Sage\CRM\CRM\WWWRoot\Examples\Webservices\*). A summary of each of the sample applications is given below. In each case the code is written in C# Visual Studio 2005.

You must log on for each example before you run it. To do this, enter your CRM username and password and click the Log On button. When you are finished running your example click the Log Off button. Remember to change the webservice location to

match your installation.
http://<webserver>/<crminstall>/eware.dll/webservice/webservice.wsdl

CRM Add Resource

This is an example of adding a resource (user) to CRM through web services. The sample code lets you enter a first name and a last name for the resource. It creates a record in the User table.

CRM AlterColumnWidth

This example demonstrates how to alter the width of a column on a table in CRM. The Company table and column comp_practicefield is hard coded for this example.

CRM Create

This example shows how to create a company, person, address and phone record.

CRM Delete

This example shows how to delete a company from CRM. The user must enter a company id for this to work.

CRM Logon and Logoff

This example shows how to log on and log off. When the user logs onto CRM, a session id is returned and displayed on the form.

CRM MetaData

This example shows how to get information about CRM tables. The user can select from five entities (company, person, case, opportunity and communication) in a list. By clicking the MetaData button, all the table columns are displayed on the left-hand pane. Highlighting any of the columns will display information about that column. For example, field name, field type and field length. There is also a button called AllMetaData. When selected, the button displays all the main CRM tables in a drop-down list. The user can then select from the list and see all columns associated with that table.

CRM QueryEntity

This example shows how to query a table—in this case the company table. The user enters a known company id and clicks the Search button. The query returns all people associated with the specified company id.

CRM QueryIdNoDate

This example allows the user to enter a date or chose a date from a calendar. The user can then click the Query N/D button. A list of company ids—where the update date is greater than or equal to the date entered—is returned.

CRM SelectionLists

This example demonstrates the use of selection lists. The user can select from a list of seven tables (company, person, opportunity, case, communication, solution, or library) and then clicks the button List. If the user selects company, for example, and clicks List, the Lists drop-down list is populated with all selection fields in that table. For example, Revenue, Sector, and Status. On choosing a selection field from this list, the left-hand pane is populated with values from the selection field when the user clicks Lists Items. For example, if the Status field from the company is chosen, the values on the left-hand pane display the following: Archive, Closed, and Inactive.

CRM SID Grabber

This example shows how to grab the session id from the logged on CRM install. At least one user must be logged on for this to work. It displays the session id and the version number.

CRM SID_Key

This is an example shows how to grab the session id from the logged on CRM install. At least one user must be logged onto CRM for this to work. It displays the session id on the form.

CRM Update

This is an example of how to update a CRM table. For this example, the user enters a valid company id. The fields that can be updated are Source and Website for purposes of the demo. When the user clicks the Update button a popup box is displayed letting the user know that the record has been updated.

CRM Version

This is an example of how to get the version number of the CRM install. It displays the session id on the form and the version number in a popup box.

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:

```
ewarebase[] idList = new ewarebase[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>
```

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";
       ewarebase[] companyList = new ewarebase[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" mlns: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>
                       <typens:timestamp>2004-08-30T18:10:00</typens:timestamp>
               </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>
                       <typens:updateddate>2004-08-30T18:10:00</typens:updateddate>
                       <typens:timestamp>2004-08-30T18:10:00</typens:timestamp>
               </records>
       </address>
</records>
</result>
</queryentityresponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Chapter 10: SData Read-only

In this chapter you will learn how to:

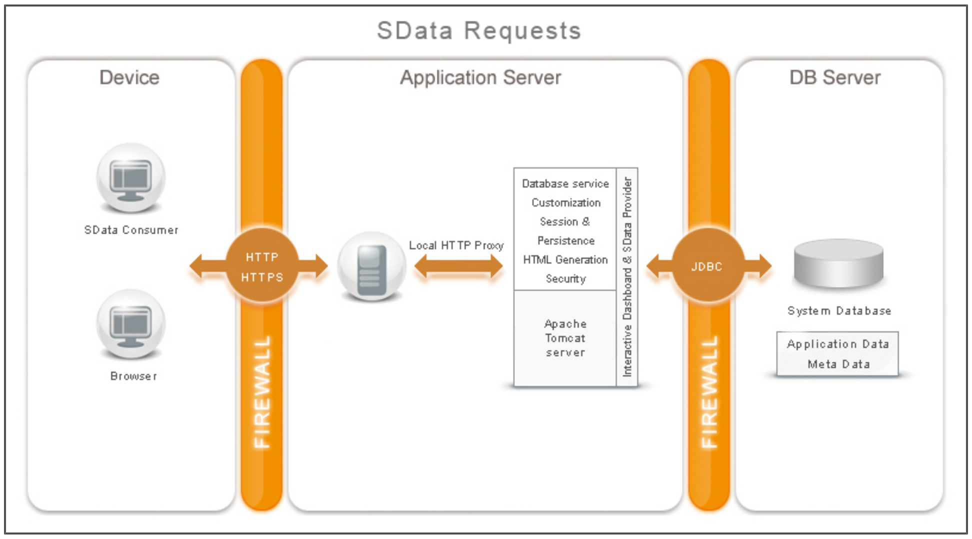
- Understand what SData is.
- Get an overview of SData within Sage CRM.
- Switch on SData for Entities and Views.
- Construct SData URLs.

Introduction to SData

SData is a standard for reading and writing data between Sage applications, enabling desktop, server, and web-based Sage applications to communicate with each other as well as third-party applications and the Web.

Although the user of SData in Sage CRM is limited to **read operations**, the standard covers basic reading, writing, updating and deleting of data between and across products as well as more complex functions such as synchronization of data, security, discoverability of services, single sign-on, error handling, and paging and batching of information for increased performance.

SData is built on top of leading industry standards including HTTP, XML, REST, and Atom/RSS.



SData Requests

Overview of SData within Sage CRM

Sage CRM is a read-only SData provider. This means entities from the Sage CRM system are exposed using the SData standard so that they can be accessed (read and queried) by third-party applications using ATOM feed technology.

Each entity and view within Sage CRM can be exposed for SData access (see Switching on SData). An XSD schema definition is available to allow 3rd party application to know what entities are exposed.

The SData request results in an XML return. This is either a single record or a list/collection of records. Embedded URLs can be used to further requests, for example drill-downs (see SData URLs).

Note: If the page size for an SData request for CRM data exceeds 100 records, it will default to 100 records. If a user is trying to access SData via an external system, and they define a page count of 200 records in the URL, the payload will return all the information in blocks of 100 (i.e. 100 records per page). The same will apply if the user tries to define a URL with no pagination.

Agreeing to the SData License Agreement (http://sdata.sage.com/sdatacore_licensing.html) is a prerequisite to working with SData.

Sage CRM can consume its own SData feeds and display them on the interactive dashboard. See more information on this in the Sage CRM *User Guide*.

SData Prerequisites

To set up SData, you will need to have the following installed on the server:

- CRM with a standard license key
- Apache Tomcat

Apache Tomcat is a servlet container developed by the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java" HTTP web server environment for Java code to run. It is installed as part of Sage CRM from version 7.0 onwards.

If requests are issued for the SData Provider then the request is redirected to the Tomcat Server.

If you working with Sage CRM 7.0 and you encounter a problem that requires a web server reset you may need to reset both IIS and Apache Tomcat. For more information on this please refer to the *System Administrator Guide*.

Switching on SData

This section shows how to switch on SData in Sage CRM.

All primary and secondary entities can be exposed for SData access. They are enabled by default. Custom entities, external tables and user views can also be enabled for SData access (see next sections).

To enable an entity for SData access:

1. Select **Administration | Customization |** <Entity> **| External Access**.
2. Click on the **Change** button.
3. Set the **Read-only SData** field to Yes or No.
4. Click **Save**.

To enable a custom table for SData access:

1. Select **Administration | Advanced Customization | Tables And Databases | Create Table**.
2. Set the **Read-only SData** field to Yes.

To enable User views (views created by you) for SData access:

1. Select **Administration | Customization |** [Entity] **| Views**.
2. Select an existing view or create a new one.

3. Select the **SData View** check box, and **Save**.

> **Note**: Disabling an entity for SData access does not override views exposed for SData access.

To select SData access during the Advanced Customization Wizard steps:

1. Make sure the 7.0 Advanced Customization Wizard zip is in the INF directory of your CRM install.
2. Select **Administration | Customization | Component Manager**.
3. Browse to the **INF** directory of your install and upload the Advanced Customization Wizard zip file.
4. Click on the component, and select **Install Component**.
5. Select the **Allow Read-only SData Access** check box on the Component Parameters page.

Constructing SData URLs

The URL to make a CRM view or entity available to another application would take the following format:

http://*myserver*/sdata/*installname*j/sagecrm/-/*entity*

- **myserver** is the name of the Sage CRM server
- **installname** is the name of the Sage CRM install
- **entity** is the name of the entity or view you want to access.

We will see some examples in the next section.

SData URL Examples

The following section shows some examples of SData URLs that would return Sage CRM data to a third party application. The SData request results in an XML return.

```
Please refer to Developer Help files for sample URLs.
```

SData Authentication

All requests must be authenticated. SData “data” access is subject to the same territory model as normal users, and profile security (CRUD rights per entity), and access rights depending on the user’s type — Admin or Non-admin. CRM authentication should be encrypted within the HTTP header.

The authentication uses Base64 encoding to encode the user name/password in the header of the URL request (with a name of X-Sage-Authorization) . The example below shows how this might work:

```
request.Headers.Add("X-Sage-Authorization", "Basic "+
Convert.ToBase64String(Encoding.ASCII.GetBytes(this.userName + ":" + this.password)));
```

Any CRM user is able to use SData — it doesn’t use up a user license so a user can be logged into CRM, and still be able to access CRM data via a third party, external application.

Chapter 11: .NET

In this chapter you will learn how to:

- Discuss how to extend Sage CRM with .NET.
- Use .NET Application Extensions.
- Debug the .Net API.
- Get an overview of the .NET Class Library documentation.
- Install the .NET SDK
- Troubleshoot a .NET SDK install.
- Uninstall the .NET SDK.
- Get an overview of .NET examples.
- Create a simple CRM interface.
- Create a more complex CRM interface.
- Create a project based on a Sage CRM template.
- Describe the CRM Basic Template.
- Describe the CRM Entity template.

Extending Sage CRM With .NET

The Sage CRM .NET API allows developers to create customizations using Visual Studio (or other .NET authoring tools) which can then be run inside the Sage CRM application. This is an alternative to the traditional ASP-based implementation of customizations.

The Sage CRM .NET API conforms to the .NET 2.0 Framework. It provides a type library that exposes the Sage CRM objects, properties, and methods. Through its core libraries the Sage CRM .NET Component manages both data access and web interface generation.

Projects developed using the Sage CRM .NET Component will be compiled into a DLL and called directly from within Sage CRM. By using Sage CRM meta data Application Extensions constructed using the Sage CRM .NET API will look, feel and perform exactly like core system pages.

To get started with the Sage CRM .NET API, see the following sections:

To do this...	See this section...
Install the .NET SDK	Installing the .NET SDK (page 11-8)
Understand the .NET Templates	Creating a Project Based on a Sage CRM Template (page 11-16)
See some examples	.NET Examples (page 11-10)
See the Namespaces and Classes	.NET API Class Library Documentation (page 11-6)
Find out about debugging	.NET API Debugging (page 11-4)
See how to call your .NET assemblies in CRM	Using .NET Application Extensions (page 11-2)

Sage CRM .NET API and ASP.NET

Reference to the Sage CRM .NET component from within ASP.NET projects is not supported, so it is not possible to use the API to create ASP.NET pages and run them inside CRM.

Programming Languages

Any programming language that conforms with the .NET 2.0 Framework can be used for the development of Sage CRM Application Extensions (E.g. J#, C# VB.NET etc)

Note: Due to changes in the Sage CRM .NET API for version 6.2, any assemblies created for versions prior to 6.2 will not work with Sage CRM 6.2 and vice-versa.

Component Manager

It is possible to use Component Manager to transfer .NET application assemblies to other CRM systems. See Component Manager (page 6-1).

Connection Pool

A System Parameter, **DotNetConnectionPool**, allows .NET Connections to choose whether to use the connection pool. The default is ‘N’.If set to N, this will release the connection when the Object is destroyed.The number of connections will increase and decrease in accordance with thread execution. If set to 'Y', CRM will use OLEDB to open database connections, and OLEDB has its own database pool implementation.

For example, executing the following query sets this System Parameter to Y:

```
UPDATE Custom_sysparams
SET Parm_Value ='Y'
WHERE Parm_Name ='DotNetConnectionPool'
```

After a change to the System Parameters, refresh the System Parameters Metadata from within Sage CRM in **Administration | System | Refresh Metadata**.

Using .NET Application Extensions

When you have created a .NET assembly for CRM, you must first copy it to the CustomDotNet folder. You can then reference the assembly from buttons, menus etc within your CRM install.

The CustomDotNet Folder

The CustomDotNet folder is located here:

<CRM install path>\<install name\CustomDotNet

For example *C:\Program Files\Sage\CRM\CRM\CustomDotNet.*

There are two ways to add your custom .NET file to the CustomDotNet folder:

- Change the build location of the DLL.
- Copying the DLL file to the CustomDotNet Folder.

Changing the Build Location of the DLL

After creating your new project you can change the build location of the DLL to be your Custom DotNet folder.

To change the build location of the DLL in Visual Studio:

1. Go to **Project | ProjectName | Properties**.
2. Edit the Output path.
3. **Save** the project and build it.

The DLL will be created in your Sage CRM CustomDotNet folder.

Copying the DLL file to the CustomDotNet Folder

If you have not changed the build location while working on the DLL file you can copy the file from your build location to the CustomDotNet folder. You can check the default location of the DLL files by going to Project | ProjectName Properties in Visual Studio. The file can be copied via windows explorer to your CustomDotNet folder.

Calling the .NET Application Extension

Once the DLL is in place, you can use the following methods to launch the new custom Application Extension:

- Call the .NET Application Extension from a Tab or Menu
- Call the .NET Application Extension from a hyperlink in a List Block
- Call the .NET Application Extension from within another custom .NET Application
- Call the .NET Application Extension from an ASP Page

Each of these methods is described below.

Calling the .NET Application Extension From Tabs/Menus

To call the .NET Application from a Tab or Menu, set the tab or menu Action to "customdotnetdll", and enter the name of your file and your primary method in the "Custom Dot Net Dll Name" and "Method Name" fields respectively.

For example, if you have created an assembly called QuickLook.dll, and you want to launch the 'RunQuickLook' method from a tab in the Company context, then follow these steps:

1. Select **Administration | Customization**.
2. Select the **Company** entity. A sequence of tabs indicating customizable areas of the selected entity is displayed.
3. Select the **Tabs** tab.
4. In the table displayed, click the **Customize** icon beside the hyperlink for Company. The Customize tabs page, which allows you to customize existing tabs or create new ones, is displayed.
5. In this case, we want to add a new tab to run our .NET assembly. So in the **Properties** panel, enter the name **QuickLook** in the Caption field and click on the **Add** button. A tab named Searchbox has been added to the list under the heading Desktop HTML Tab Group Contents. You can adjust this new tab's position in the tab sequence by using the up and down arrows beside the list.
6. Next, we need to specify that the new tab runs the DLL that we have placed in our CustomDotNet folder. In the Actions drop-down field, select the **customdotnetdll** option.
7. Next enter the DLLname (in this case **QuickLook**) into the **Custom Dot Net Dll Name** field.
8. Finally, enter the base method name, in this case **RunQuickLook** into the Method Name field.
9. Click on the **Update** button to add these details to QuickLook tab.
10. Click on the **Save** button to confirm the changes.

Calling the .NET Application Extension From List Block Hyperlinks

You can allow users to run your custom .NET assembly by clicking on a hyperlink in a list block. To set this up:

1. Open the List Definition for your List Block. For example to open the standard Company List Definition, go to **Administration | Customization | Company** and click on the **Lists** tab, and then click on **Company Grid**.
2. Select the field that you want to add the hyperlink to.
3. In the Field Properties panel, set the **Hyperlink To** drop-down to **Custom Dot Net DLL**.
4. In the **Custom Dot Net DLL Name** field type the name of your DLL file (e.g. **QuickLook**).
5. In the **Custom ID Field**field enter the Id field name that needs to be passed to the DLL method (e.g. **Comp_Companyid**).
6. In the **Method Name** field enter the method name in your Base class file (e.g. **RunQuickLook**).
7. Click on the **Update** button to save these updates to your field.
8. Click on the **Save** button to confirm the changes.

Calling the .NET Application Extension From Within Another Custom .NET Application

If you want to call your custom .NET assembly from within another custom .NET assembly, use the UrlDotNet method (part of the Web class in the Sage.CRM.WebObject Namespace). For example:

```
string sUrl = UrlDotNet(ThisDotNetDll, "RunViewOpportunity");
AddUrlButton("Cancel", "Cancel.gif", sUrl);
```

You can also use the Dispatch.Redirect method within another DLL:

```
Dispatch.Redirect(UrlDotNet(“NewCompany.dll”, “RunDataPage”));
```

Note: The.NET API Redirect works differently from the Redirect in ASP. The redirect will only happen when the .NET dll is finished processing the code, and it needs to provide the HTTP response before being unloaded from memory. This function should only be used inside of BuildContents and returned after the redirect is set, otherwise system performance could be seriously impacted. It is also recommended that only one Redirect is used within the code, and the URL to be used is set within any previous branches in the code.

Calling the .NET Application Extension From ASP Pages

To call a custom .NET assembly from within an ASP page, use the CRM.Url method. Please see Url(Action) (page 8-26) method in the ASP Object reference. For example:

```
myContainer.AddButton(CRM.Button(“Add”,”new.gif”,CRM.Url(“CompanyNew.dll-
RunDataPage”)));
```

.NET API Debugging

There are the two different ways to setup the environment to allow debugging of the .Net API.

Note: Never use these configurations on a production system. They are strictly intended for development or testing systems only!

Method 1: Change IIS Security

These steps are for IIS 6 and for the CRM DLL (EWARE.DLL) virtual directory *only*.

1. Go to: *Control Panel* | *Administrative Tools* | *Internet Information Services*
2. Select your Sage CRM Install | *Right click – Properties* | *Directory Security* | *Anonymous Access and Authentication Control – Edit.*
3. Remove the **Anonymous Access**. Anonymous Access gives the .Net API process the same permission as an IIS user, and this user hasn't permission enough to allow debugging.
4. Make sure to check **Integrated Windows Authentication**.
5. Reset IIS.
6. In Visual Studio, add a break point to your code.
7. Select **Debug** | **Attach To Process**.
8. Find the **inetinfo.exe** process that shows as **Managed** in the type column, and click **Attach**.
9. Run your code and it should stop at the breakpoint.

Method 2: Use COM+

1. Go to **Control Panel** | **Administrative Tools** | **Component Services**.
2. Expand **Computers** | **My Computer** | **COM+ Applications**.
3. Right-click on **COM+ Applications** and select **New** | **Application**.
4. Select **Create an Empty Application** .
5. Enter a name, e.g. DotNetServerApp and select **Server Application**.
6. Click **Next**.
7. Under Account, select **This User** and enter the user name and password of an Administrator account. The .Net API needs run under Administrator permissions to allow the VS Studio debugger to be attached.
8. Click **Next**.
9. Click **Finish**. Your new application should appear under COM+ Applications.
10. Click the **+** sign beside your application to expand it, and right click **Components** and select **New** | **Component**.
11. Select **Import Components that are Already Registered**.
12. Find the component Sage.CRM.Wrapper.**SageCRMBase**.
13. Click **Next** and click **Finish**.
14. On the Component Services screen, expand **Running Processes**. Note that your newly created application should not be in the list of running applications.
15. Now go to CRM and launch your .NET application from within CRM. Return to Component Services and you should now see your application in the Running Processes list. Note that there is a Process ID assigned to the application (in brackets beside the application name). This is the process we will use in Visual Studio.
16. In Visual Studio, add a break point to your code.
17. Select **Debug** | **Attach to Process**.

18. Find the process with the same Process ID noted above, and click **Attach**.
19. Run your code and it should stop at the breakpoint.

Troubleshooting

Problem	Solution
Visual Studio doesn't stop at your break point.	Verify that the yourdll.pdb file is in <installation>\CustomDotNet, the Microsoft framework needs this file to find the Source Code.
The "managed" option didn't appear on inetinfo.exe process.	Reset IIS and run any .Net API Code. It's necessary to run .Net API code at least once before you can make the managed option available in Visual Studio.
The process isn't created in Component Services.	Verify if you have the correct version of DLLs and if you have set the correct password.
The message "Interface Not Supported" appears when the .Net Code is invoked.	The COM+ Debug setup described here works only for CRM 6.2 and later versions. Previous version will result in this error message. If you are using the CRM 6.2, try to un-register the DLLs and register again. Please see Installation Troubleshooting (page 11-9).
The Debug stops working after a few days.	Probably after setting up the application you have changed your password and you didn't update the password on Component Service.
The application process is created correctly on Component Services, but CRM shows the message "Interface Not Supported".	On Internet information service add the "Integrated Windows Authentication" see: http://msdn2.microsoft.com/en-us/library/x8a5axew(vs.80).aspx

.NET API Class Library Documentation

The .NET Class Library documentation is provided with the CRM .NET SDK. It is in the form of a Microsoft Help file and can be found in your CRMDotNet folder (normally *C:\Program Files\Sage\CRM\CRMDotNet\Class Library Documentation\Documentation.CHM*).

You can also use the Object Browser in Visual Studio to view the descriptions of the Namespaces, Classes, Methods and Properties.

The namespaces are listed below with a list of the principal classes under each.

Sage Namespace

- Address Types
- CaptionPositions
- Captions
- KeyList
- ParamNames
- RunExternalMethod
- SageCRMBase

- Styles
- UserOptions

Sage.CRM.WebObject Namespace

- DataPage
- DataPageBase
- DataPageDelete
- DataPageEdit
- ListPage
- SearchPage
- Web

Sage.CRM.Controls Namespace

- Entry
- EntryGroup
- EntrySelect
- GridCol
- List
- EntryAdvSearchSelect
- EntryCustom
- GridColCheckBox
- CountSql. Set a Custom count SQL to help the grid pagination. This can be used to improve the grid performance. The SQL should return only one record which need to be named fcount and represent the number of record returned by SelectSql + Filter properties. For example:

  ```
  Select count(*) as fcount from Company
  ```

Sage.CRM.Data Namespace

- CommunicationEntity
- Entity
- EntityCollection
- QuerySelect
- Record

Sage.CRM.Utils Namespace

- Dispatch
- Keys
- Metadata
- TableInfo
- TranslationFamily
- UserSession

Sage.CRM.Blocks Namespace

- SageCrmBlock
- SageCrmChartGraphicBlock
- SageCrmFileBlock
- SageCrmGraphicBlock
- SageCrmMarqueeBlock
- SageCrmMessageBlock
- SageCrmOrgGraphicBlock
- SageCrmPipelineGraphicBlock

Sage.CRM.HTML Namespace

- HTMLBuilder

Sage.CRM.UI Namespace

- ComplexBox
- ContentBox
- HorizontalPanel
- HTMLString
- Hyperlink
- ImageLink
- ImageObject
- Panel
- UIEntry
- UIObject
- VerticalPanel

Installing the .NET SDK

The .NET SDK is available to Sage CRM Development Partners and contains the following:

- Visual Studio Templates
- Extensive Code Snippets
- Source Code samples

Please note that the SDK is only distributed to Development Partners. The SDK Requires a DPP License. Only DPP staff who have attended the official .NET training program will be able to place support requests.

Pre-Installation Checklist

To develop applications with the Sage CRM .NET SDK, you will need:

- CRM installed on a server with a Developer license key.
- The .NET Setup package (CRMSDK.zip) comprising the CRMDOTNETSETUP.MSI file and the SETUP.EXE executable.
- Visual Studio 2005 or 2008 (recommended, other development tools including Visual Studio Express Editions, may also be suitable).

Servers on which compiled .NET Application Extensions are deployed require:

- Sage CRM 6.1 or later installation with a developer license key. Sage CRM 6.2 or later is recommended for creating new .NET based customizations.
- .NET Framework 2.0.

Installing the SDK and .NET Templates

To install the CRM .NET SDK, extract all the files from CRMSDK.zip and run the SETUP.EXE file.

The SETUP.EXE installs the files(SageCrmEntityWizard, SageCrmWrapper and SageCRMNet) into the Global Assembly Cache ("GAC") and registers the SAGECRMNET.DLL. These files are copied into C:\Program Files\Sage\CRM\CRMDotNet\<version> by the installer.

When creating a new project in Visual Studio, you should see icons for CRM Basic Template and CRM Entity Template under the **My Templates** section of the New Project Window. If these do not appear, see Installation Troubleshooting (page 11-9).

Installation Troubleshooting

This section describes how to manually register the .NET assemblies, and also how to manually copy the CRM .NET Templates to the correct location.

Manually registering the DLL

After running the SDK setup, the following assemblies should be visible in the Global Assembly Cache (c:\windows\assembly):

- SageCrmEntityWizard
- SageCRMNet
- SageCrmWrapper

Note: SageCrmWrapper and SageCRMNet are installed during the main Sage CRM installation and not during the SDK installation.

If you encounter difficulty with the installation process, or if you receive an error message such as '*This template attempted to load an untrusted component...*' when attempting to create a new project, you may need to manually register the DLL.

The following steps show how to manually (re)register the SageCRMNet assembly using the Visual Studio 2005 command prompt (the procedure for Visual Studio 2008 is identical, except for the program name):

1. Go to **Start | Programs | Microsoft Visual Studio 2005 | Visual Studio Tools | Visual Studio Command Prompt**.
2. Navigate to the folder where the SAGECRMNET.DLL has been installed, and type:

   ```
   cd \program files\sage\crm\crmdotnet\<version number>
   ```

   Replace <version number> with the actual CRM version number, e.g. '6.2'.

3. Force reinstallation of the assembly to the Global Assembly cache

   ```
   gacutil /if sagecrmnet.dll
   ```

4. Register it by typing:

   ```
   regasm sagecrmnet.dll
   ```

5. Install SageCrmWrapper.dll:

```
gacutil /if SageCrmWrapper.dll
```

Manual Installation of the CRM Visual Studio .NET Templates

When you attempt to create a new project in Visual Studio, you should see icons for CRM Basic Template and CRM Entity Template under the **My Templates** section of the New Project Window. If these do not appear, you may have to copy the templates manually.

When you install the SDK, the installer will determine the version of Visual Studio that you are using and it will copy the Template Zip files to the appropriate template folder. For example if you are using Visual Studio 2008 then the Zip files will be copied to *...\My Documents\Visual Studio 2008\Templates\ProjectTemplates\Visual C#*

If you are using another Visual Studio version, you can copy the Zip files from the above location and copy them to the appropriate location for your development environment. See your Visual Studio documentation for more information.

If the Entity Template appears in 'My Templates' but does not function properly, then you can reinstall the SageCrmEntityWizard Dll.

1. Go to **Start | Programs | Microsoft Visual Studio 2005 | Visual Studio Tools | Visual Studio Command Prompt**.
2. Force re-installation of SageCrmEntityWizard.dll:

```
gacutil /if SageCrmEntityWizard.dll
```

Uninstalling the .NET SDK

To uninstall the SDK components follow these steps:

1. Go to **Start | Control Panel | Add or Remove Programs**.
2. Remove the **CRMDotNetInstall**.
3. Go to **Start | Run** and type **c:\winnt\assembly** or **c:\windows\assembly**for Windows XP.
4. View the GAC(Global Assembly Cache) and check that CRMEntityWizard has been removed. If not, it can be uninstalled by right clicking and selecting Uninstall from the pop up menu.

DotNetWrapper and SageCRMNet are installed by the main CRM setup and should not be removed.

.NET Examples

These Visual Studio C# example code demonstrates how to use the .NET API. It is assumed that you have installed the SDK and have verified that the CRM Project Templates are available to you in Visual Studio. Please see Installing the .NET SDK (page 11-8).

Creating a Simple Sage CRM Interface (page 11-11)
Creating a More Complex CRM Interface (page 11-12)

The CRM .NET SDK also includes two complete Visual Studio sample application. After installing the SDK, these can be found in your Visual Studio Projects folder, for example: *\My Documents\Visual Studio 2008\Projects\*

Creating a Simple Sage CRM Interface

In this example we will build a screen that displays two blocks. The first block will display the current Company details and the second will display the company's primary person details.

1. Within Visual Studio create a new project using the CRM Basic template (see Creating a Project Based on a Sage CRM Template (page 11-16)).
2. Open the **CustomPage.cs** file.
3. Remove all the lines of code in this file that start with **AddContent** (i.e. the sample code that we will not be using). Leave the GetTabs line as this will retrieve and display the Company tabs.
4. In order to use the Record object you must add in a reference to the Sage.CRM.Data namespace at the top of the file:

```
 using Sage.CRM.Data;
```

5. To be able to use the object EntryGroup you must add references to the Sage.CRM.Controls and Sage.CRM.Utils namespaces to the top of the file:

```
using Sage.CRM.Controls;
using Sage.CRM.UI;
```

6. Get the current company and person records, to do this we use the Sage.CRM.Data Record object and method FindCurrentRecord.

```
//Retrieve Records
 Record recCompany = FindCurrentRecord("Company");
 Record recPerson = FindCurrentRecord("Person");
```

Note that the company and person records could also be retrieved by using GetContextInfo to get the current company.

```
//Establish Context
string strCompID = GetContextInfo("Company", "Comp_Companyid");
string strPersonID = GetContextInfo("Company", "comp_primarypersonid");
//Retrieve Records
Record recComp = FindRecord("Company", "comp_companyid = " + strCompID);
Record recPers = FindRecord("Person", "pers_personid = " + strPersonID);
```

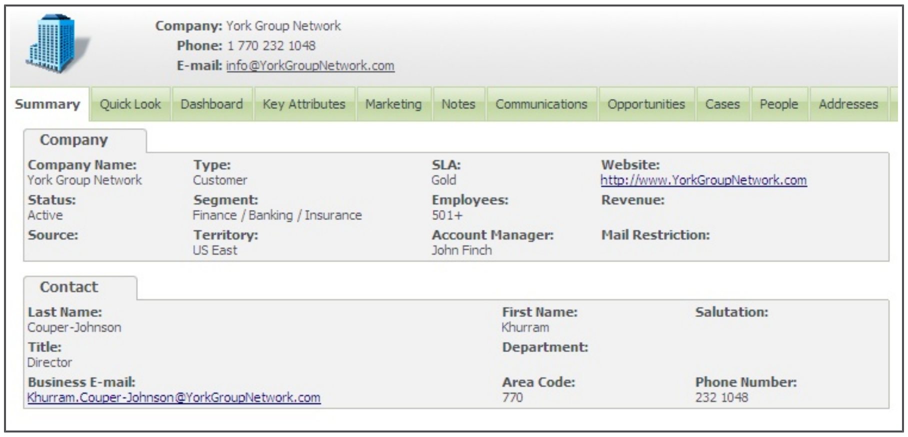
7. Once we have the current records we need to get the screens which will display the information. To do this we use the Sage.CRM.Controls EntryGroup object and the Sage.CRM.Utils MetaData.GetScreen method.

```
//Get the screens
EntryGroup screenCompanyBoxLong = new EntryGroup("CompanyBoxLong");
screenCompanyBoxLong.Fill(recComp);
EntryGroup screenPersonBoxShort = new EntryGroup("PersonBoxShort");
screenPersonBoxShort.Fill(recPers);
```

8. The last thing to do is display the screens with the current company and person record information. *HTML.StartTable* begins the formatting of the table in which the blocks will be displayed. *HTML.BoxTitle* adds a caption to the block. *HTML.BoxContent* adds the block to the main area of the table. *GetHtmlInViewMode* is used to pass the record to the block. *HTML.BlankRow* adds a blank line for formatting purposes.

```
//display the screens
VerticalPanel vpMainPanel = new VerticalPanel();
vpMainPanel.AddAttribute("width", "100%");
vpMainPanel.Add(screenCompanyBoxLong);
vpMainPanel.Add(screenPersonBoxShort);
AddContent(vpMainPanel);
```

9. Build the Solution in Visual Studio.
10. Copy the DLL to the CustomDotNet folder. Please see Using .NET Application Extensions (page 11-2).
11. Create a tab to call the DLL. Please see "Calling the .NET Application Extension From Tabs/Menus" in Using .NET Application Extensions (page 11-2).
12. When launched from the tab, the screen that you have created is displayed.



Company: York Group Network
Phone: 1 770 232 1048
E-mail: info@YorkGroupNetwork.com

Company

Company Name: York Group Network	**Type:** Customer	**SLA:** Gold	**Website:** http://www.YorkGroupNetwork.com
Status: Active	**Segment:** Finance / Banking / Insurance	**Employees:** 501+	**Revenue:**
Source:	**Territory:** US East	**Account Manager:** John Finch	**Mail Restriction:**

Contact

Last Name: Couper-Johnson	**First Name:** Khurram	**Salutation:**
Title: Director	**Department:**	
Business E-mail: Khurram.Couper-Johnson@YorkGroupNetwork.com	**Area Code:** 770	**Phone Number:** 232 1048

Company and Person details

Creating a More Complex CRM Interface

In this example we are going to create a screen that displays company, person and opportunity entry blocks, allowing three records to be entered and saved at the same time.The user will then be redirected to the company summary screen of the newly created company. The CRM .NET SDK includes the complete code for this example. After installing the SDK, the solution file can be found in your Visual Studio Projects folder, for example: \My Documents\Visual Studio 2008\Projects\ .

1. Within Visual Studio open a new project using the CRM Basic Template. Please see Creating a Project Based on a Sage CRM Template (page 11-16).
2. Within the Solution Explorer at the right-hand side of the screen right click on the name of your new project and select **Add | Class**.
3. Select the first type on the list (C# class) and name your class. This will add a new blank C# class to your project. Now we add code to build and display our blocks and save the information entered by the user.
4. First we must add references to the Sage.CRM namespaces we will use at the top of the file. These allow us access to the Sage CRM .NET objects, methods and properties we will need.

```
using Sage.CRM.Blocks;
using Sage.CRM.Controls;
using Sage.CRM.Data;
using Sage.CRM.HTML;
using Sage.CRM.Utils;
using Sage.CRM.WebObject;
using Sage.CRM.UI;
```

5. Now we will start to build the screen. The class EntryScreen has been created for us. We want this to be an instance of the Sage.CRM webobject class called Web. This will allow us write the HTML that builds the screen using the SageCRM API calls. To do this change the class definition line as below:

```
class EntryScreen: Web
```

6. The Web class provides us with one main method to write our HTML which will display the screen we want to create. This is called BuildContents. We need to override the base class in order to implement our own code. To this we add this code within the EntryScreen class:

```
public override void BuildContents(){
}
```

7. It is within the above curly braces that we add our code. In the above code statement *Public* indicates that variables declared are visible to all methods; *Override* indicates that the method has the same name as one in a base class and is to be used instead of the version in the base class and the *Void* return type indicates that a method does not have a return value.
8. Now we want to display the screens. First we set up the form in which they will be displayed:

```
AddContent(HTML.Form());
```

9. Now we will add code to the class that will get the blocks, display and format them and save the data entered by the user. The first step is to get the blocks or screens that will allow the user to enter data:

```
EntryGroup screenCompanyBoxLong = new EntryGroup("CompanyBoxLong");
screenCompanyBoxLong.Title = Metadata.GetTranslation("tabnames", "company");
EntryGroup screenPersonBoxLong = new EntryGroup("PersonBoxLong");
screenPersonBoxLong.Title = Metadata.GetTranslation("tabnames", "person");
EntryGroup screenOppo = new EntryGroup("OpportunityDetailBox");
screenOppo.Title = Metadata.GetTranslation("tabnames", "opportunity");
```

10. We determine which mode we are in - i.e. are we editing the page or are we saving data. To do this we need to check the hidden field HiddenMode to see if it has been set to Save, if so then we need to create and save a new record for company, person and opportunity and in the process saving the user entered data.

```
string hMode = Dispatch.EitherField("HiddenMode");
if (hMode == "Save")
{
       //Code for saving
}
else
{
       //Code for displaying the forms
}
```

11. If we are in save mode then we populate the three tables with the data from our forms:

```
Record recCompany = new Record("Company");
screenCompanyBoxLong.Fill(recCompany);
recCompany.SaveChanges();
Record recPerson = new Record("Person");
recPerson.SetField("pers_companyid",recCompany.GetFieldAsInt("comp_companyid"));
screenPersonBoxLong.Fill(recPerson);
recPerson.SaveChanges();
```

```
recCompany.SetField("comp_primarypersonid",recPerson.GetFieldAsInt("pers_
personid"));
recCompany.SaveChanges();
Record recOppo = new Record("Opportunity");
recOppo.SetField("oppo_primarycompanyid",recCompany.GetFieldAsInt("comp_
Companyid"));
screenOppo.Fill(recOppo);
recOppo.SaveChanges();
```

12. Now the records have been created and saved. We want to redirect the user to the new company summary screen. To do this we use the we use the Dispatch.Redirect method. We need to pass it the Sage CRM action key 200 which is for the company summary screen. We also need to pass in the company id for the newly created company record. To do this we need to split action key url and add in the key1=CompanyId, see the code below:

```
Please refer to Developer Help files for code sample
```

Note: The.NET API Redirect works differently from the Redirect in ASP. The redirect will only happen when the .NET dll is finished processing the code, and it needs to provide the HTTP response before being unloaded from memory. This function should only be used inside of BuildContents and returned after the redirect is set, otherwise system performance could be seriously impacted. It is also recommended that only one Redirect is used within the code, and the URL to be used is set within any previous branches in the code.

13. If we are in edit mode, then we create a vertical panel and add the three entry boxes to it. In order for the form to work properly we need to add the hidden field HiddenMode to the class.

```
AddContent(HTML.InputHidden("HiddenMode", ""));

VerticalPanel vpMainPanel = new VerticalPanel();
vpMainPanel.AddAttribute("width", "100%");
screenCompanyBoxLong.GetHtmlInEditMode();
screenPersonBoxLong.GetHtmlInEditMode();
screenOppo.GetHtmlInEditMode();
vpMainPanel.Add(screenCompanyBoxLong);
vpMainPanel.Add(screenPersonBoxLong);
vpMainPanel.Add(screenOppo);
AddContent(vpMainPanel);
```

14. The code above will display the three blocks in edit mode. We have not added any buttons or any code to save the data entered. We need a Save and a Clear button. The clear button will take us to the same DLL method:

```
AddUrlButton("Cancel", "cancel.gif", UrlDotNet(ThisDotNetDll, "RunEntryScreen"));
```

The clear button calls ThisDotNetDll, meaning the DLL we are in and then calls the method RunEntryScreen (which is the method we will create in the Base.cs file that displays the interface we are building).

15. The Save button will refresh the screen and pass in a value to hidden field we will create. This will indicate the screen is now in Save mode.

```
string sUrl ="javascript:document.EntryForm.HiddenMode.value='Save';";
AddSubmitButton("Save", "Save.gif", sUrl);
```

16. We also add a Help button that links to a html page in the help system:

```
string strHelpUrl = "/Main Menu/Default_CSH.htm?href=AI_FAQs.html";
AddHelpButton(strHelpUrl);
```

> As an alternative, you can use AddHelpButton("help.htm"); This gives you access to the list of help files in inline translation mode. Please refer to the System Administrator Guide for more information.

17. Now the records have been created and saved. We want to redirect the user to the new company summary screen. To do this we use the we use the Dispatch.Redirect method. We need to pass it the Sage CRM action key 200 which is for the company summary screen. We also need to pass in the company id for the newly created company record. To do this we need to split action key url and add in the key1=CompanyId, see the code below:

```
Please refer to Developer Help files for code sample.
```

18. Now we must add a method to the Base.cs file which will allow us to run this class from within Sage CRM.
19. Open Base.cs, copy the public method RunMyCustomPage, paste it in below the exising RunMyCustomPage. Rename this to be something meaningful like RunEntryScreen and change the line of code to call an instance of the class EntryScreen:

```
public static void RunEntryScreen(ref Web AretVal)
{
    AretVal = new EntryScreen();
}
```

20. Build the Solution in Visual Studio.
21. Copy the DLL to the CustomDotNet folder. Please see Using Using .NET Application Extensions (page 11-2).
22. Create a tab to call the DLL. Please see "Calling the .NET Application Extension From Tabs/Menus" in Using .NET Application Extensions (page 11-2).

Here is the complete code for the **EntryScreen.cs** file:

```
Please refer to Developer Help files for code sample.
```

And here is the complete code for the Base.cs file:

```
using System;
using System.Collections.Generic;
using System.Text;
using Sage.CRM.WebObject;
namespace CompoundEntryScreen
{
    public static class AppFactory
    {
        public static void RunEntryScreen(ref Web AretVal)
        {
            AretVal = new EntryScreen();
        }
    }
}
```

Creating a Project Based on a Sage CRM Template

After the successful installation of the SDK components, you can launch Visual Studio to create a project based on a Sage CRM template.

To create a new project:

1. Select **New | Project** from the menu bar.
2. In the New Project dialog box, select **Visual C#** from the Project Types pane.
3. Select either the **CRM Basic Template** or the **CRM Entity Template** from the My Templates section.
4. Type the application's name in the **Name** field.
5. Click **OK**.

The files, references, and stub code generated by a CRM Extension template provides the developer with a compilable application that is open to customization and extension.

CRM Basic Template

The CRM Basic Template generates files and code that enables developers to build code which implements customized versions of standard CRM interfaces and functionality.

The following class files are created by the CRM Basic Template:

- Base.cs
- CustomPage.cs

Base.cs holds the methods that will be called from within Sage CRM, i.e. the method name that will be used to call the application from the CRM tab or menu etc.

Base.cs contains the following code initially:

```
using System;
using System.Collections.Generic;
using System.Text;
using Sage.CRM.WebObject;

namespace Crm_Basic_Template1
{
    //static class AppFactory is REQUIRED!
    public static class AppFactory
    {
        public static void RunMyCustomPage(ref Web AretVal)
        {
            AretVal = new MyCustomPage();
        }
    }
}
```

The public class AppFactory is required in every Sage CRM Application Extension. This is where the methods to be called from within Sage CRM reside.

The only method, RunMyCustomPage, will run the custom interface that you create in the second class file CustomPage.cs.

CustomPage.cs contains this code initially:

```
using System;
using System.Collections.Generic;
using System.Text;
using Sage.CRM.WebObject;
```

```
namespace Crm_Basic_Template1
{
    public class MyCustomPage : Web
    {
        public override void BuildContents()
        {
            //Add your content here!
            GetTabs();
            AddContent("My Custom Page");
            AddContent("<BR>");
            //how to show translated values - maybe
            AddContent(Metadata.GetTranslation(Sage.Captions.sFam_GenMessages,
"HelloWorld"));
            AddContent("<BR>");
            //how to check sys param values
            AddContent("The Base Currency is: " +
Metadata.GetParam(Sage.ParamNames.BaseCurrency));
            AddContent("<BR>");
            //...etc
            //Dispatch.
        }
    }
}
```

The statement using Sage.CRM.WebObject has been included in CustomPage.cs so that we can implement an instance of a web class. Of course, other 'Using' statements can be added. See .NET Examples (page 11-10).

Web classes give access to methods to build HTML to be returned to the browser. The public class MyCustomPage is an instance of the general Web class simply called Web. This is used for building screens from scratch. In the CRM Entity Template (page 11-17) you will see the use of the specialized Web classes such as DatePage, DataPageEdit, ListPage etc.

In this class the public method BuildContents () is overwritten. This means that this version of the method will be used instead of that in the base class. BuildContents() is the main method that will always be called - override this method to build your own page. The main function of this method is to build up the HTML that creates the screen that is shown to the user.

The BuildContents method in CustomPage.cs initially contains some methods to add tabs (GetTabs) and content (AddContent) to the screen. These can be replaced with your own methods and code.

CRM Entity Template

The CRM Entity Template allows the rapid creation of a .NET Application Extension that provides a customized graphical user interfaces for CRM Entities.

Creating a new CRM Entity Application in this way does not create a new CRM entity table(s), screens and listblocks. This should be done first in the Sage CRM front end, or through the Advanced Customization Wizard. Please see Database Customization (page 5-1) and Advanced Customization Wizard (page 5-17) for more details.

To create a new CRM Entity Template:

1. Select **New | Project** from the menu bar.
2. In the New Project dialog box, select **Visual C#** from the Project Types pane.
3. Select the **CRM Entity Template** from the My Templates section. The CRM .NET Entity Wizard dialog box is displayed.

4. Fill out the fields as follows:
   - **New Entity Name**. The name of the entity you are creating. For example "Project".
   - **Id field**. The identity field in the table you created for this entity. For example "proj_projectid".
   - **Prefix**. The prefix the fields in the new table have. For example "proj".
5. Click **GO!**

The following files are created by the CRM Entity Template:

File	Description
CrmBase.cs	Similar to Base.cs created in the CRM Basic Template. Contains seven methods, one to call each of the class files created by the template.
EntityDataPage.cs	Displays the screen that allows a user to view an entity in Sage CRM. The IdField and EntityName are taken from those entered on setting up the CRM Entity Template. The AddEntryGroup specifies what entry group will be displayed. It defaults to EntityNameNewEntry, this can be changed to whatever you have called your block in Sage CRM.
EntityDataPageDelete.cs	The files EntityDataPageNew.cs, EntityDataPageEdit.cs and EntityDataPageDelete.cs are similar to EntityDataPage.cs. They create an instance of their respective specialized Web classes and again the IdField, EntityName and AddEntryGroup are set based on the information entered when the template was created.
EntityDataPageEdit.cs	See above
EntityDataPageNew.cs	See above
EntityListPage.cs	The EntityListPage.cs displays a list of the entities within Sage CRM. Again it creates an instance of a specialized Web class ListPage. The EntityName is taken from the information entered when the entity template was created. The list name and filter box name specify what screens should be displayed. The FilterByField and FilterByContextId allow the list to be filtered dynamically, in this case by user id.
EntitySearchPage.cs	The EntitySearchPage displays the search or find screen for the entity. It creates an instance of the specialized Web class SearchPage.

Index

.

.NET 11-1
- architecture 2-3
- class library documentation 11-6
- creating complex interface 11-12
- creating simple interface 11-11
- CRM basic template 11-16
- CRM Entity Template 11-17
- debugging 11-4
- examples 11-10
- extending Sage CRM 11-1
- installation troubleshooting 11-9
- installing the SDK 11-8
- Sage CRM template 11-16
- vs classic ASP 2-4

.NET application extensions
- using 11-2

.NET SDK
- uninstalling 11-10

3

3rd party gadget
- adding to interactive dashboard 4-16

A

Accessing
- user information 3-9

add 9-6

AddAddress(Address, Name) 8-9

AddBlock(Block) 8-37

AddButton(ButtonString) 8-37

AddCoachingCaptions 6-15

AddColumn 6-16

AddContent(Content) 8-17

AddCustom 6-16

AddCustom_ContainerItems 6-16

AddCustom_Data 6-17

AddCustom_Databases 6-18

AddCustom_Edits 6-18

AddCustom_Lists 6-19

AddCustom_Report 6-20

AddCustom_ReportBand 6-21

AddCustom_ReportChart 6-21

AddCustom_ReportField 6-22

AddCustom_ReportGroup 6-22

AddCustom_ScreenObjects 6-23

AddCustom_Screens 6-24

AddCustom_Scripts 6-25

AddCustom_Tables 6-26

AddCustom_Tabs 6-27

AddEntry(8-56

AddFile(Value) 8-12

AddGridCol 8-73

Adding
- 3rd party gadget to interactive dashboard 4-16
- buttons to button groups 4-8
- charts to classic dashboard 4-12
- content block to classic dashboard 4-11
- content block to interactive dashboard 4-15
- frames 7-17
- help to custom pages 4-3

list block to classic dashboard 4-11
new chart 7-4
new data entry screens 5-7
new graphic 7-10
report view to entity 5-25
tab linking to ASP page 4-25
AddLPCategory 6-27
AddLPGadget 6-28
AddLPLayout 6-28
AddLPUserLayout 6-28
AddMessage 6-29
AddPipeEntry 8-83
AddProduct 6-29
addrecord 9-7
addresource 9-6
address 9-9
Address 8-15
AddressList Object 4-22, 8-9
methods 8-9
properties 8-10
AddView 6-29
Admin menu button 4-21
Advanced component manager options 6-9
Advanced customization wizard
example 5-26
parameters 5-18
Advanced Customization Wizard 5-17
aisid 9-8
Alignment 8-70
AllowBlank 8-44
AllowOrderBy 8-70
AllowUnassigned 8-54
altercolumnwidth 9-6
Animation 7-17
adding frames 7-17
delay 7-17
loops 7-17
sample ASP 7-18
Animation(Mode, Value) 8-60
Application level security 2-5
Arc 8-59
ArgObj 8-29
ASP
vs .NET 2-4
ASP Object Reference 8-1
introduction 8-2
ASP page
building 3-1
integrating into CRM 3-4
ASP Quick Reference Table 8-3
Attachment Object 4-22, 8-10
methods 8-10
properties 8-11
AttachmentList Object 4-22, 8-11
properties 8-11
Attachments 8-14
Authenticated 8-98
AuthenticationError 8-98

B

BackGradient(Visible, color1, color2) 8-33
BCC 8-15
Block
creating new 4-18
customizing 4-19
displaying 4-19

Blocks 4-18
  overview 4-1
Body 8-13
Bof 8-87
Border 8-68
Brush 7-15
Brush(Mode, Value) 8-60
Building
  ASP page 3-1
Button 8-23
Button groups
  adding buttons to 4-8
  creating 4-8
  restricting access 4-9
  viewing 4-9
Button Groups 4-8
ButtonAlignment 8-39
ButtonImage 8-39
ButtonLocation 8-40
Buttons 4-8
  creating new admin menu 4-21
  main menu 4-20
ButtonTitle 8-40

C

Caption 8-45
CaptionFamily 8-75
CaptionPos 8-45
cases 9-9
Category 8-101
CC 8-15
Changing
  entity logo 5-25
  image color 7-14
Changing the current component 6-4
Chart
  adding 7-4
  examples 7-4
Charts
  example 7-5
  external data 7-3, 7-5
  overview 7-1
  special effects 7-3
ChartTitle(text) 8-34
CheckLocks 8-29
ChooseBackGround(Value) 8-84
Chord 8-60
Classic Dashboard 4-9
  adding charts 4-12
  adding content block 4-11
  adding list block 4-11
  customizing 4-9
Clear() 8-12
Clearing
  image 7-14
Client-side
  scripting 3-7
Color 7-14
communication 9-9
company 9-9
Component details screen 6-2
Component Manager 6-1
  adding a message example 6-36
  adding a new column example 6-37
  adding a new view example 6-38
  adding customizations 6-3

advanced options 6-9
changing current component 6-4
chaning a screen name example 6-36
copying an ASP page example 6-37
creating a new table example 6-37
error handling 6-12
installing a component 6-7
introduction 6-1
log file 6-8
methods 6-14
modifying scripts 6-11
multi-stage customizations 6-5
multiple installs example 6-36
preview changes 6-4
recording 6-2
replacing text in ASP page example 6-37
saving a component 6-7
script parameters 6-13
scripting changes 6-4
scripting customizations 6-4
scripting examples 6-36
scripting workflows 6-6
starting 6-3
stopping 6-3
using the DataFile object example 6-38
what can be recorded 6-1

Configuring
web services 9-3

Connecting
database 5-5
external database 5-5

Content block
adding to classic dashboard 4-11
adding to interactive dashboard 4-15

Contents 8-43
Context 3-7
ConvertValue 8-27
CopyAndDropColumn 6-30
CopyAspTo 6-30
CopyErrorsToPageErrorContent 8-55
CopyFile 6-30
Count 8-10, 8-11, 8-104
CreateNewDir 6-30
CreateQueryObj 3-5
CreateQueryObj(SQL, Database) 8-18
CreateRecord 3-6
CreateRecord(TableName) 8-18
CreateScript 3-10, 8-46
CreateTable 6-31

Creating
admin menu button 4-21
button groups 4-8
complex CRM interface 11-12
custom page 5-5, 5-7
custom queries 3-5
entity level scripts 5-14
external link menu 4-21
invoices tab 5-6
list 4-3
list object 5-5
main menu button 4-20
new block 4-18
new database connection 5-3
new main entity 5-17
new tab group 4-24
new table connection 5-4

screen 4-5, 5-7
simple CRM interface 11-11
tab to display list 5-4
table 5-1
table level script 5-14
CRM Add Resource 9-14
CRM AlterColumnWidth 9-14
CRM Blocks Overview 4-1
CRM Create 9-14
CRM Delete 9-14
CRM Entity Template 11-17
CRM interface
creating complex 11-12
CRM Logon and Logoff 9-14
CRM MetaData 9-14
CRM Object 4-21, 8-17
methods 8-17
properties 8-22
CRM Objects Overview 4-21
CRM QueryEntity 9-14
CRM QueryIdNoDate 9-14
CRM SelectionLists 9-14
CRM SID Grabber 9-14
CRM SID_Key 9-15
CRM TargetListField Object 4-22
CRM TargetListFields Object 4-22
CRM TargetLists Object 4-22
CRM Update 9-15
CRM Version 9-15
CRMBase Object 4-21, 8-22
methods 8-23
properties 8-27
CRMBlock Object 4-23, 8-27
methods 8-28
properties 8-29
CRMChartGraphicBlock Object 4-24, 8-33
methods 8-33
properties 8-35
CRMContainerBlock Object 4-23, 8-36
methods 8-37
properties 8-39
workflow properties 8-41
CRMContentBlock Object 4-24, 8-43
properties 8-43
CRMEmail Object
methods 8-12
CRMEntryBlock Object 4-23, 8-43
methods 8-44
properties 8-44
CRMEntryGroupBlock Object 4-23, 8-55
methods 8-56
properties 8-57
CRMFileBlock Object 4-23, 8-57
properties 8-58
CRMGraphicBlock Object 4-24
methods 8-59
properties 8-68
CRMGridColBlock Object 4-23, 8-69
properties 8-70
sample ASP page 8-7
CRMListBlock Object 4-23, 8-72
methods 8-73
properties 8-75
sample ASP page 8-7

CRMMarqueeBlock Object 4-23, 8-77
  properties 8-77
CRMMessageBlock Object 4-23, 8-79
  properties 8-80
CRMOrgChartGraphicBlock Object 4-24
CRMOrgGraphicBlock Object 8-82
  methods 8-82
CRMPipelineGraphicBlock Object 8-82
  methods 8-83
  properties 8-85
CRMPipeLineGraphicBlock Object 4-24
CRMQuery Object 4-23, 8-85
  methods 8-86
  properties 8-87
crmrecord 9-8
CRMRecord Object 4-22, 8-89
  methods 8-89
  properties 8-92
crmrecordtype 9-8
CRMSelfService Object 4-22, 8-95
  methods 8-97
  properties 8-98
CRMTargetListField Object 8-99
  properties 8-99
CRMTargetListFields Object 8-103
  methods 8-104
CRMTargetLists Object 8-99
  creating and saving a Target List 8-102
  methods 8-100
  properties 8-100
  retrieving a target list 8-103
currency 9-9
Current component 6-3
CurrentUser 3-9
Custom Entities
  make availabe to reassign 5-24
Custom files 5-22
Custom page
  creating 5-5, 5-7
Custom queries 3-5
CustomActionFile 8-71
CustomIdField 8-71
Customization
  overview 2-6
Customization Basics 4-1
Customizing
  block 4-19
  classic dashboard 4-9
  interactive dashboard 4-13
  new entity 5-25

D

Database
  connecting to 5-5
  connecting to external 5-5
  creating new connection 5-3
  overview 2-7
Database customization
  introduction 5-1
Database Customization 5-1
Database level security 2-6
DatabaseName 8-87
DataField 8-99
Debug 8-17
Debugging
  .NET 11-4

Deduplication	
enabling	5-21
DefaultType	8-46
DefaultValue	8-47
Delay	7-17
delete	9-7
Delete(Index)	8-104
DeleteBlock(BlockName)	8-38
DeleteColumn	6-31
DeleteCustom_Caption	6-31
DeleteCustom_Captions	6-32
DeleteCustom_Field	6-32
DeleteCustom_Screen	6-32
DeleteCustom_ScreenObjects	6-32
DeleteEntry(EntryName)	8-56
DeleteGridCol(ColName)	8-73
DeleteRecord	8-92
example	5-17
DeliveryTime	8-14
Description	8-68, 8-101
Detached table level scripts	5-14
DirectoryPath	8-58
Dispatch Object	4-21
DisplayButton	8-41
DisplayForm	8-29, 8-80
Displaying	
errors	7-14
Drawing	
functions	7-14
DropTable	6-33
DropView	6-33

E

Editing	
list object	5-6
main menu tab group	4-25
Effect(Mode, Value)	8-61
Ellipse(X1,Y1,X2,Y2)	8-61
email	9-9
Email Object	4-22, 8-12
properties	8-13
EmailAddress	8-17
Enabling	
deduplication	5-21
EndSSSession	8-97
Entities	2-7
Entity	
adding report view	5-25
changing logo	5-25
creating new main	5-17
customizing	5-25
Entity level script	5-14
example	5-15
functions	5-12
introduction	5-12
EntryType	8-47
Eof	8-88, 8-93
ewarebase abstract	9-8
ewarebaselist	9-8
Exclude(ATargetID)	8-100
ExecSql()	8-86
Execute(Arg)	8-28, 8-74
Extending Sage CRM	
with .NET	11-1

Extensibility
  architecture 2-2
  module 2-2
Extension 8-11
External images 7-10

F

FAM 8-48
FastLogon 8-27
Field Level Scripting 3-10
FieldName 8-48
Fields 8-101
FieldValue 8-88
FileName 8-58
FileOpen 6-33
FindRecord 8-19
FinishLandingPage 6-29
FirstRecord() 8-89
FlipHoriz() 8-61
FlipVert() 8-61
Font 7-16
Font(Mode, Value) 8-62
FontColor(Color) 8-62
FontSize(Size) 8-63
FormAction 8-30
Frames
  adding 7-17
Functions
  drawing 7-14
  entity level scripts 5-12
  table level scripts 5-12
  web services 9-5

G

getallmetadata 9-7
GetBlock(BlockName) 8-19, 8-38
GetContextInfo(Context, FieldName) 3-7, 8-24
GetCustomEntityTopFrame 8-20
GetDLLDir 6-34
getdropdownvalues 9-6
GetEntry 8-57
GetField() 6-33
GetGridCol 8-74
GetInstallDir 6-34
getmetadata 9-6
GetPage() 8-20
GetTabs(TabGroup) 8-25
Getting Started 3-1
GetTrans(Family, Caption) 8-20
GetUniqueFileName(Path, FileName) 8-16
getversionstring 9-7
GradientFill 8-63
Graphic
  adding 7-10
  effects 7-14
Graphics
  examples 7-10
  formats 7-8
  overview 7-8
  performance 7-9
  special effects 7-17
Graphics and Charts 7-1
  introduction 7-1
GrayScale() 8-63

H

Header(Value) 8-13

Height 8-30

Help

  adding to custom pages 4-3

Hidden 8-49

HorizontalMaximum 8-77

HorizontalMinimum 8-78

hSpace 8-69

I

idbase abstract 9-8

IdField 8-93

Image

  clearing 7-14

  external 7-10

Image color

  changing 7-14

ImageHeight 8-69

ImageWidth 8-69

Include(ATargetID) 8-100

Init 8-97

InsertRecord

  example 5-16

Installation

  .NET troubleshooting 11-9

Installing

  .NET SDK 11-8

Installing a component 6-1, 6-7

Interactive dashboard

  customizing 4-13

Interactive Dashboard 4-13

  adding 3rd party gadget 4-16

  adding content block 4-15

IsHTML 8-13

Item 8-93, 8-104

ItemAsString 8-94

Items 8-10, 8-11

J

JumpEntity 8-49, 8-72

L

LabelX 8-35

LabelY 8-35

lead 9-9

Library

  .NET API Class 11-6

LibraryPath 8-12

LineTo(X,Y) 8-64

List

  creating 4-3

  displaying 4-4

  displaying using ASP page 4-4

  displaying using runblock 4-5

List block

  adding to classic dashboard 4-11

List object

  creating 5-5

  editing 5-6

ListBlock 3-5

Lists 4-3

LoadBMP(Filename) 8-63

LoadImage(text) 8-64

LoadJPG(Filename) 8-64
Log file
  component manager 6-8
Log(value) 8-16
logoff 9-5
logon 9-5
Logon(LogonId, Password) 8-26
LookUpFamily 8-50
Loops 7-17

M

mAddressFrom/mNameFrom 8-80
Mail Address Object 4-22
MailAddress Object 8-15
  properties 8-15
MailAdmin(Subject, Body) 8-16
Main menu
  create external link from 4-21
  creating button 4-20
  editing tab group 4-25
ManualChartEntry 8-34
MaxLength 8-50
mBody 8-80
Merging 7-16
mErrorMessage 8-81
Metadata 2-8, 5-23
Methods
  component manager 6-14
Mode 8-22, 8-33
Modifying
  system menus 4-20
Monochrome() 8-65
MoveTo(X,Y) 8-65
mSentOK 8-81
Msg 8-16
MsgHandler Object 4-22, 8-15
  methods 8-16
  properties 8-16
mShowCC/mShowBCC 8-81
mSubject 8-82
MultipleSelect 8-51
multiselectfield 9-8

N

Name 8-11, 8-15, 8-30, 8-101
New Workflow Properties
  sample ASP page 8-8
New(CRMTargetListField) 8-104
NewLine 8-31
newproduct 9-9
next 9-6
Next() 8-86
nextqueryrecord 9-7
NextRecord() 8-86, 8-90
NextRow() 6-33

O

Objects
  web services 9-5
OnChangeScript 3-10, 8-51
opportunity 9-9
opportunityitem 9-9
OrderBy 8-94
OrderByFields 8-101
orderitem 9-9
orders 9-9

Organizational chart
  example 7-6
OrgTree
  example 7-6
OrgTree(Mode, Value) 8-82

P

PadBottom 8-75
Param 6-34
Parameters
  advanced customization wizard 5-18
Parent 8-104
Pen 7-14
Pen(Mode, Value) 8-65
PenColor(Color) 8-65
PenWidth(Width) 8-66
person 9-9
phone 9-9
Pie chart effects 7-2
PieShape 8-66
Pipe_Summary 8-85
Pipeline
  example 7-11
PipelineStyle(Mode, Value) 8-84
PostInsertRecord
  example 5-16
Preview
  scripting customizations 6-4
Previous() 8-86
prevURL 8-76
pricing 9-9
pricinglist 9-9
Priority 8-13
productfamily 9-9
ProperCase 8-58
Properties 8-104

Q

query 9-5
queryentity 9-6
queryid 9-6
queryidnodate 9-6
queryrecord 9-7
QueryResultsToFile 6-34
Quick Reference Table 8-3
quoteitem 9-9
quotes 9-9

R

ReadOnly 8-52
Recipients 8-14
RecordCount 8-88, 8-94
recordfield 9-8
RecordID 8-95
Recording customizations 6-1, 6-2
RecordLock 8-90
Rectangle(X1,Y1,X2,Y2) 8-66
RefreshMetaData(Family) 8-21
RemoveLookup 8-44
Required 8-52
Resize(Width, Height) 8-66
Restricting
  access to button groups 4-9
  access to tab 4-26
Restrictor 8-54
Retrieve() 8-100

Rotate(Number) 8-67

RoundRect(X1,Y1,X2,Y2,X3,Y3) 8-67

RowsPerScreen 8-76

Runblock

- display a list using 4-5
- display a screen 4-7, 4-7

RunSQL 6-35

S

Sage CRM

- template 11-16

Sample ASP pages 8-7

Save() 8-100

Save(Name, Path) 8-10

SaveAs(Name, Path) 8-10

SaveAsGIF(text) 8-69

SaveAsGifs 8-67

SaveAsJPG(text) 8-67

SaveChanges() 8-91, 8-91

Saving

- component 6-7

Screen

- adding new data entry 5-7
- creating 4-5
- creating object 5-7
- display using ASP page 4-7
- display using runblock 4-7, 4-7
- displaying 4-6

Script parameters 6-13

Scripting

- client-side 3-7
- examples (component manager) 6-36
- in CRM interface 3-10
- multi-stage customizations 6-5
- workflows 6-6

Scripting customizations 6-1, 6-4

Scripts

- modifying component manager 6-11

scrmGetGadgetProperty 4-17

scrmGetSageCRMOwner 4-18

scrmPublishEvent 4-17

scrmRegisterEvent 4-17

scrmSaveGadgetProperties 4-18

scrmSetGadgetProperty 4-18

ScrollSpeed 8-78

SData 10-1

- authentication 10-3
- enabling 10-2
- introduction 10-1
- overview 10-1
- prerequisites 10-2
- urls 10-3

SearchAndReplaceCustomFile 6-35

SearchAndReplaceInDir 6-35

SearchAndReplaceInFile 6-35

Security 2-5

- application level 2-5
- database level 2-6
- server level 2-5

Selected 8-85

SelectSql 8-76

SelectSql() 8-87

Send() 8-12

SenderAddress 8-14

SenderName 8-14

Server level security 2-5

SetContext(EntityName, EntityID) 8-21

SetWorkflowInfo 8-92

ShowHeading 8-72

ShowLegend(true/false) 8-34

ShowNewWorkflowButtons 8-42

  sample ASP page 8-9

ShowSavedSearch 8-57

ShowSelectAsGif 8-72

ShowValidationErrors 8-31

ShowWorkflowButtons 8-42

  sample ASP page 8-8

Size 8-53

SQL 2-8, 8-89

SQLText=Text 8-35

Starting component manager 6-3

Stopping component manager 6-3

Style 7-15

Stylename(Style) 8-34

StyleSheet 8-78

Subject 8-13

System menus

  modifying 4-20

T

Tab

  actions 4-27

  creating invoices 5-6

  creating to display a list 5-4

  linking to ASP page 4-25

  restrict access to 4-26

Tab group

  creating new 4-24

Table

  create new connection 5-4

  creating 5-1

Table level script

  creating 5-14

  detached 5-14

  functions 5-12

  introduction 5-12

Table level scripting 3-12

TableExists 6-35

Tabs 4-24

TargetListID 8-100

TargetLists 8-27

Template

  creating a project 11-16

  CRM basic 11-16

  CRM entity 11-17

TextOut 8-68

TextOutCenter 8-68

Title 8-31

Translate 8-59

Triggers 2-8

Troubleshooting

  .NET installation 11-9

U

Uninstalling

  .NET SDK 11-10

uom 9-9

uomfamily 9-9

update 9-6

updaterecord 9-7

UpdateRecord
  example 5-15, 5-16
Url(Action) 8-26
User setup
  web services 9-2
users 9-9
Using
  .NET application extensions 11-2

V

Validate() 8-28
ValidateScript 3-10, 8-53
VerticalMaximum 8-78
VerticalMinimum 8-78
Viewing
  button groups 4-9
ViewName 8-32, 8-101
VisitorInfo 8-99
vSpace 8-69

W

Web architecture 2-1
Web Services 9-1
  Add Resource 9-14
  AlterColumnWidth 9-14
  configuration 9-3
  Create 9-14
  Delete 9-14
  functions 9-5
  Logon and Logoff 9-14
  Metadata 9-14
  objects 9-5
  QueryEntity 9-14
  QueryIdNoDate 9-14
  SelectionLists 9-14
  SID Grabber 9-14
  SID Key 9-15
  Update 9-15
  user setup 9-2
  Version 9-15
WhereClause 8-102
Width 7-15, 8-32
Workflow
  scripting 6-6
WorkflowTable 8-42
  sample ASP page 8-9
WSDL file 9-4

X

XLProp=text 8-35
Xprop=text 8-35

Y

Yprop=text 8-36

CRM Document Version Code: DEV-MAN-ENG-710-1.0