Sage CRM 2023 R2
Developer Guide

Updated: September 2023

Contents

About this guide

This guide is for Sage CRM implementers, developers, and system administrators. It provides information on how to customize and extend the functionality of Sage CRM by using the *Extensibility Module* (EM), also known as *CRM Blocks*.

This guide assumes that you are familiar with the following:

- Sage CRM System Administrator Guide or Help
- SQL Server databases, tables, views, data relationships, and normalization
- ASP, C#, HTML, JavaScript, JSON, Microsoft .NET Framework, SOAP, Web Services, and XML

Step-by-step instructions in this guide assume that your default theme in Sage CRM is **Contemporary**. If you are using a different theme, you may need to use slightly different steps to access the **Administration** area in Sage CRM.

> **Note:** This help refers to *Sage CRM* but your system might have a different brand name, such as *Sage 200 Sales and Marketing*. The system works in the same way regardless of its name. The functionality that's available to you depends on the modules that you're licensed to use.

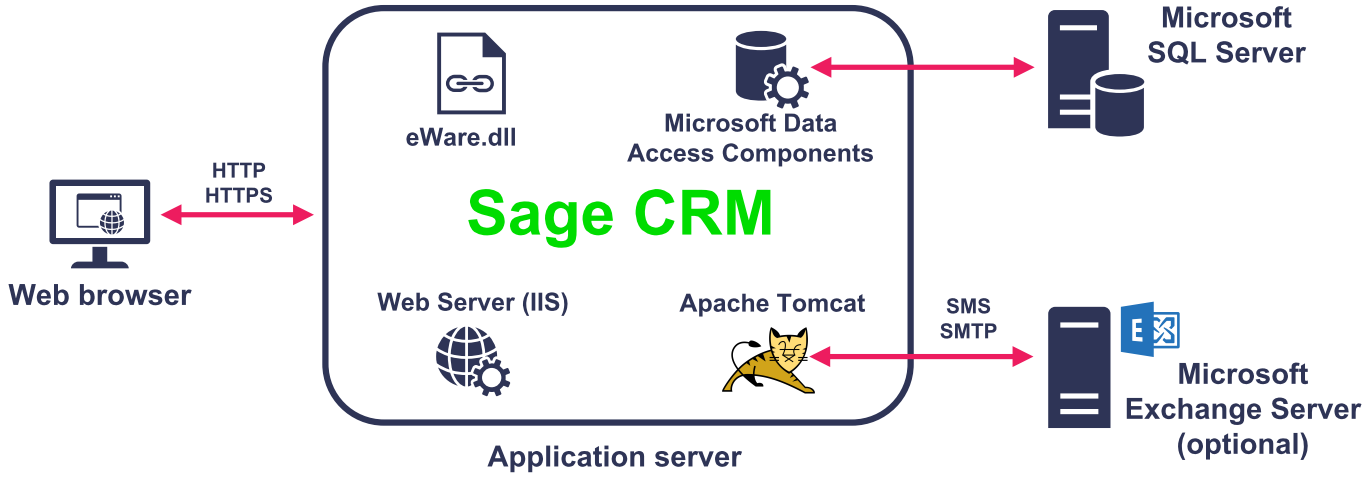
Getting started

- **Sage CRM architecture**
- **Creating an ASP page**
- **Adding an ASP page to Sage CRM**
- **Framesets**
- **Creating custom queries**
- **Understanding context**
- **Scripting in the Sage CRM interface**

Sage CRM architecture

- **Web**
- **Extensibility**
- **.NET API**
- **Security**
- **Database**
- **Customization**

Web



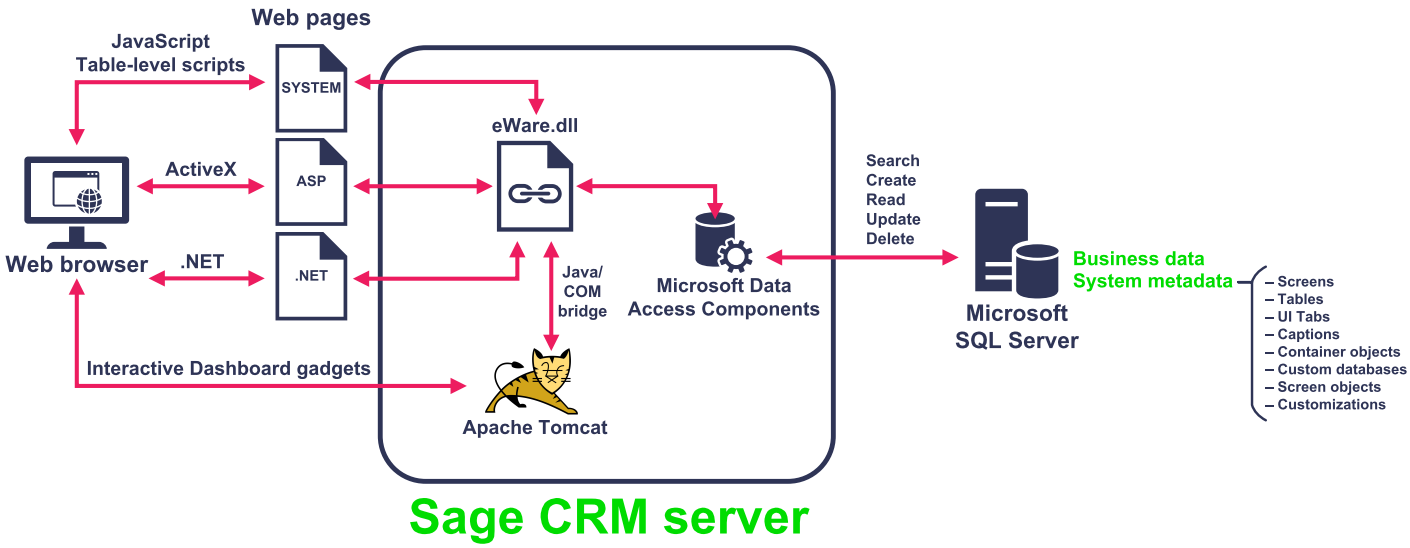
- **Web browser**. Sage CRM has a thin client configuration, therefore users can access and work with Sage CRM by using just a web browser. This can be any web browser supported by Sage CRM.
- **Application server**. The Sage CRM application server includes a number of components, which work together to coordinate the delivery of information and functionality to Sage CRM users. These components are used to implement user security, maintain user persistence, read and write information in the Sage CRM database, generate web pages from data, and process business rules and logic. The application server also runs the eWare.dll file, which is the Sage CRM dynamic link library. Web Server (IIS) communicates with Sage CRM via eWare.dll and Internet Server Application Programming Interface (ISAPI) technologies. eWare.dll and Web Server (IIS) must be on the same server.

- **Microsoft SQL Server**. This server hosts the Sage CRM database that stores corporate data and metadata, which define the Sage CRM screen configurations, system customization, security, and business rules.
- **Mail server**. You can connect the application server to this optional component to automate the sending of emails and SMS messages as part of the Sage CRM implementation. To customize the email functionality, you can use the CRMEmail object, and the CRMMsgHandler object and its child objects. You can also use the CRMMessageBlock object to send messages in SMS and email format.
- **Apache Tomcat Redirector**. The redirector in Sage CRM 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 if Tomcat is working. For more information, see the *Installation and Upgrade Help* on the **Sage CRM Help Center** and articles on the **Sage CRM Community**.

Extensibility

The Extensibility Module (EM) provides a range of powerful functions that allows you to customize and extend the existing CRM product. The functions are available through the CRM ActiveX object, which consists of CRM methods and properties. CRM object components have a variety of functions, which render HTML and display Screen and List objects previously defined in the CRM system.

Database connectivity options include searching, inserting, updating, and deleting data to and from CRM data, as well as data from external tables and databases.



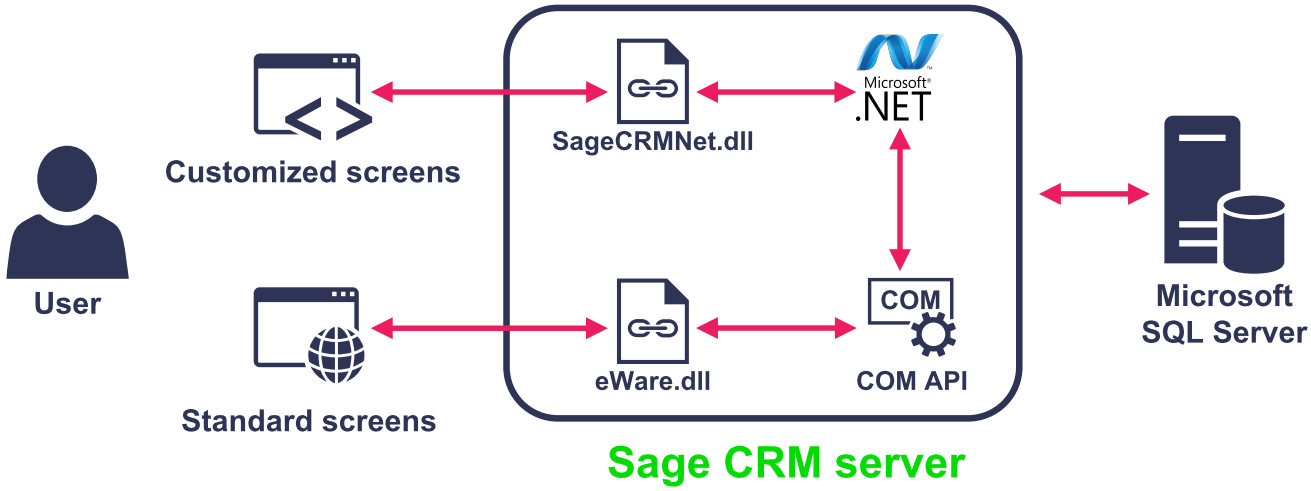
To see if the EM is included in your CRM installation, go to **<My Profile> | Administration | Customization | Company**. If the EM is included, **Blocks** and **TableScripts** tabs are displayed.

.NET API

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 (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.



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

The Sage CRM .NET API requires Microsoft .NET Framework 2.0. It provides a type library that exposes Sage CRM objects, properties, and methods. Through its core libraries, the Sage CRM .NET Component manages data access and web interface generation. Projects developed using the Sage CRM .NET Component are compiled into a DLL and called directly from within Sage CRM. By using Sage CRM metadata, Application Extensions constructed using the Sage CRM .NET API look, feel and perform like core system pages.

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

You can develop Sage CRM Application Extensions with any programming language that conforms with the .NET Framework 2.0, such as C#, VB.NET, or J#.

Advantages of using .NET API versus ASP

.NET API advantages	ASP drawbacks
• **Benefit from Microsoft Visual Studio features such as IntelliSense**. With the Sage CRM .NET component properly referenced by the application, Visual Studio treats Sage CRM objects like any other initialized C# objects. For example, if you're using a Sage CRM object and call one of its methods, IntelliSense provides you with a list of the object's properties and methods when you type the dot operator (.) after the object name. • **Access to .NET class library**. You can code in C# or Visual Basic .NET, which allows you to use the resources provided by the .NET class library. • **Improved security and protection of your source code**. .NET applications are compiled into binary DLLs before deployment, preventing access to your source code.	• **No access to Integrated Development Environment (IDE) features**. ASP pages are coded in simple text editors so you can't use IDE features, such as IntelliSense (which provides drop-down lists of available objects, properties, and methods), syntax checking, and debugging tools. • **Poor debugging**. Due to the absence of meaningful syntax checking and debugging tools in text editors, you can't ensure your ASP code is working as intended until you load your custom ASP page in Sage CRM. • **Only suitable when simple coding is required**. ASP simplicity makes it appropriate only to those solutions that require simple coding and rapid deployment. • **Does not comply to Object-oriented programming (OOP) principles**. Creating separate ASP pages is not conducive to OOP principles and associated practices, such as refactoring. Creating multifile projects can demand more advanced project tools. • **Source code is not protected**. Users can view the source code of your custom ASP pages by selecting the appropriate command from the web browser menu.

Security

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

- **Application-level security**
- **Server-level security**
- **Database-level security**

Application-level security

Every user is assigned a valid user name and password. Only the System Administrator can add or remove users. Within the system, each user can be assigned different levels of access security depending on their job role. The system knows when IIS uses HTTPS; when the client attaches documents to a form in the system, it sends it through encrypted sessions.

- **User Authentication/Password Setup**. A user requires a login ID and password to access the system. The user's password is encrypted in the system and in the database for maximum security.
- **Tab and Team Restrictions**. The System Administrator can restrict access to individual tabs within CRM, and the level of access that each user is allocated. The Administrator can assign users to teams, which further categorizes and restricts levels of access to individual tabs.
- **Security Profiles and Territories**. The System Administrator can set up Territory Profiles and Security Territories to manage security access rights across the organization. A Profile groups users according to access rights (View, Update, Insert, Delete). A Territory groups user rights by location or other criteria. For example, users in the Europe territory can view all Opportunities in the USA territory but can't update them. Administrators can set up complex inter-territory security rights and exception handling using Territory Policies. Profiles and Territories are set up from **<My Profile> | Administration | Users | Security**. For more information, see the *System Administrator Help*.

Server-level security

You can use any of the following methods to secure the Sage CRM server:

- **NT Challenge/Response**. Allows access to clients with a valid domain login.
- **Hyper Text Transfer Protocol Secure (HTTPS)**. Secures your data sessions with client users.
- **A firewall**. Restricts unauthorized access from outside the network and allows only authorized users through.

Database-level security

Sage CRM users don't have direct access to the database. The CRM DLL accesses the database 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, you can configure the CRM DLL to access the database with a user account that has limited permissions.

Database

Sage CRM entities

A Sage CRM entity represents a real world entity such as a Company, Person, or Sales Opportunity. The entity can have relationships with other entities. For example, a Company can have many People working for it, and Communications and Sales Opportunities can be associated with People.

Sage CRM entities consist of information from several tables linked in a logical way. For example, the Company entity consists 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, Sage CRM entities are several tables linked together in a logical business grouping.

Metadata

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

The system database contains Sage CRM metadata tables. Names of these tables have the **custom** prefix.

SQL and triggers

- You can use the CRM Query object's SQL method to include SQL statements in an ASP page.
- You can use SQL conditional clauses in **<My Profile> | Administration | Customization | <Entity> | Tabs**. For more information, see the *System Administrator Help*.
- You can use Sage CRM table and entity level scripts instead of SQL triggers.

Creating an ASP page

Sage CRM ASP pages use 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.

The following example code creates a custom ASP page which enables users to search for contacts in the Sage CRM database and view a list of search results:

```
<!-- #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 search results.
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);
%>
```

Below are the descriptions of building blocks used in this code.

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

This building block specifies the include file that instantiates and initializes the CRM object. The include file also references the Sage CRM CSS, defines constants, and checks for errors. Depending on the scripting language you use, you can specify one of the following include files in your code:

- **SAGECRM.JS**. Referenced in JavaScript-based ASP pages. This file sets the default language to JavaScript.
- **SAGECRMNOLANG.JS**. Doesn't 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 Sage CRM versions older than 5.6.
- **ACCPACCRM.JS**. For backward compatibility with Sage CRM versions older than 7.0.
- **ACCPACCRMNOLANG.JS**. For backward compatibility with Sage CRM versions older than 7.0.
- **ACCPACCRM.VBS**. For backward compatibility with Sage CRM versions older than 7.0.

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

In this building block, ASP delimiters `<% %>` tell the ISAPI.DLL that the contained ASP code executes on the server.

The **GetBlock(BlockName)** method initializes a child block that implements core CRM functionality.

The **GetBlock(BlockName)** method parameter value is `Container`, which indicates the **CRMContainerBlock object**. This object groups and correctly displays output from 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** is stored in the variable `SearchContainer`. The Container screen contains only default buttons. To make it useful, add some blocks.

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

The **GetBlock(BlockName)** method retrieves a block and its associated functionality. This code specifies `PersonSearchBox` which is an instance of the **CRMEntryGroupBlock object**. To edit the contents of `PersonSearchBox`, go to **<My Profile> | Administration | Customization | Person | Screens | Person Search Screen**. Other standard screens based on the **CRMEntryGroupBlock object** include `CompanySearchBox`, `PersonEntryBox`, and `CaseDetailBox`.

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

Add `SearchBlock`, an instance of `PersonSearchBox`, to the screen container.

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

This building block changes the attributes of the **Save** button. The Container object creates this by default.

```
1 // If button has been pressed then add the list block to show search results.
2 if (CRM.Mode == 2)
3 {
4     var resultsBlock = CRM.GetBlock('PersonGrid');
5     resultsBlock.ArgObj = SearchBlock;
6     SearchContainer.AddBlock(resultsBlock);
7 }
```

`CRM.Mode == 2` displays a search results grid when a form is submitted.

The **GetBlock(BlockName)** method returns a `PersonGrid` block. `PersonGrid` is an instance of the **CRMListBlock object**. To view the fields displayed in this list, go to **<My Profile> | Administration | Customization | Person | Lists | Person Grid**.

The returned `PersonGrid` block is stored in the `resultsBlock` variable.

The **CRMBlock object**'s property **ArgObj**, which is a base property implemented by all subclasses (such as the `PersonGrid` block), passes `SearchBlock` as a parameter for populating the list. This means that the list `resultsBlock` takes the search screen `SearchBlock` as a parameter, and the

contents of the list generated by `resultsBlock` are determined by the values of the fields on the search screen `SearchBlock`.

The **CRMContainerBlock object**'s **AddBlock(Block)** method adds the `resultsBlock` object.

Adding an ASP page to Sage CRM

When you've created an ASP page and want to make it accessible in Sage CRM, store the .asp file in the **CustomPages** folder, and then add the .asp file in Sage CRM by using the Sage CRM administration area.

The default location of the **CustomPages** folder on a Sage CRM server is as follows:

%ProgramFiles(x86)%\CRM\CRM\WWWRoot

This example creates a **Searchbox** tab in the context of a Company.

1. Copy the saved .asp file to the **CustomPages** folder.
2. Log on to Sage CRM as a system administrator.
3. Go to **<My Profile> | Administration | Customization | Company | Tabs**.
4. In the **Tab Group Name** column, click **Company**.
5. Under **Properties**, use the following options:
    - **Caption**. Enter a caption for your new tab. For example, *Searchbox*. When finished, click **Add**.
      Your new tab appears in the list under **Desktop HTML Tab Group Contents**. Use the up and down arrows to position your new tab.
    - **Actions**. Select **Customfile**.
    - **Custom File**. Enter the name of the custom .asp file you saved in the **CustomPages** folder earlier in this procedure. For example, *MySearchBox.asp*.
6. Click **Update** and then click **Save**.

To view the new tab you have just created, open an entity record details screen.

To improve the Sage CRM User Interface, framesets are no longer used in Sage CRM version 7.2 and later. ASP pages are rendered entirely within the main browser window, which ensures that the top content and left-hand menu of Sage CRM are always rendered correctly.

Framesets

Sage CRM previously used framesets to build the user interface. Framesets were removed in Sage CRM 7.2 and the sections are now rendered entirely within the same document. The top content area and the left hand main menu are rendered in DIV tags instead of individual frames.

There's also a change in how JavaScript is included in the user interface. In Sage CRM version 7.1 and earlier, most JavaScript passed into the user interface was embedded in the code. Eware.dll generated the JavaScript used to control the interface and merged it with HTML. In Sage CRM 7.2, JavaScript is no longer wrapped up in the eware.dll but instead is contained in .js files in the CRM or custom folders in WWWRoot. These files are referenced and deployed to the client.

There are new element IDs. If you search for a screen element with a specific ID and that ID has changed, the element won't be found and your script won't work. Or the ID might now refer to a different element resulting in your script attempting to manipulate the wrong element.

We strongly recommending that you use the new **Client-Side API** methods where possible to make future updates to Sage CRM more seamless.

Due to changes to page structure, the way JavaScript code is referenced within system pages including the implementation of a new **Client-Side API**, and new element IDs, you must carefully review any code that's designed to execute in the browser.

- These changes affect client-side code that references certain functions or objects, and interacts with frames such as TopContent or eWare_mid. Review how you've included JavaScript in classic Content blocks, or have used references to ASP pages and the COM API in a custom gadget.
- Several objects in Sage CRM version 7.1 are created in other frames. For example, WritetoFrame() and Arrays (userslist, usersdislist, targetlistslist). Change and test any code that references these objects.
- Check the metadata for screens and lists that contain client-side customizations. Run the following SQL statements in the database to find screens with defined custom content and onChange scripts:

  ```
  select * from Custom_ScreenObjects where Cobj_CustomContent is not null
  select * from Custom_Screens where SeaP_OnChangeScript is not null;
  ```

- The default basic CTI in Sage CRM version 7.1 used framesets. The CTI button frame and a frame for the CTI object no longer exist in the same way in Sage CRM version 7.2. The removal of the framesets effects any customization that depends on CTI interactaction with the screens. If you're using a third party CTI integration, consult your supplier about whether it has been updated for Sage CRM 7.2 before upgrading from an earlier Sage CRM install.

Creating custom queries

- **Creating a record set**
- **Formatting a list**
- **Manipulating record sets**

Creating a record set

The Sage CRM API allows easy access to the database to select and update data. The **CreateQueryObj(SQL, Database)** method returns a record set that can be viewed and manipulated in the same way as, for example, an ADO record set.

The following example uses the **CRMQuery object** to display 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.QueryEof)
{
    CRM.AddContent(Query.FieldValue("comp_name")+'<br>');
    Query.NextRecord();
}
Response.Write(CRM.GetPage());
%>
```

In this example, the **CreateQueryObj(SQL, Database)** method returns a **CRMQuery object** by specifying a valid SQL statement as a parameter. The default database is used, but you can specify another database by adding a second parameter to the **CreateQueryObj(SQL, Database)** method call.

You can expand the scope of your query by using relational database features such as joins and views.
To examine or copy system and custom views, go to**<My Profile> | Administration | Customization | <Entity> | Views**. Alternatively, scan the list of views in a database management tool such as SQL Server Enterprise Manager.

In this example, the returned query object is called `Query`. The **SelectSql()** method executes the `Select` query. You can use the **ExecSql()** method to run queries that don't return records, such as `Delete`, `Update`, and `Insert`.

In addition to encapsulating components to access and update the database, the **CRMQuery object** stores returned data.

This example uses 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 email address are retrieved using the **FieldValue** property.

The **AddContent(Content)** method builds up a block of HTML that's displayed when the output of the **GetPage()** method is written to the ASP page using the JavaScript `Response.Write` method.

Formatting a list

Use the **CRMListBlock object** to apply desired formatting to a filtered list. To initialize the **CRMListBlock object**, invoke the **GetBlock(BlockName)** method with the appropriate parameter.

The following example uses the **GetBlock(BlockName)** method to create a new list object:

```
<!-- #include file ="sagecrm.js"-->
<%
var NewList;
NewList = CRM.GetBlock("list");
NewList.SelectSql = "SELECT * FROM Company WHERE Comp_Deleted IS NULL AND Comp_Type =
'Prospect' AND Comp_Sector = 'Software'";
NewList.AddGridCol("Comp_Name");
CRM.AddContent(NewList.Execute());
Response.Write(CRM.GetPage());
%>
```

The `Select` query for retrieving a filtered list of company records is the same as the query used by **CRMQuery object**. However, the query string is passed to the **SelectSql** property of the **CRMListBlock object**.

The **GetBlock(BlockName)** method retrieves the **CRMListBlock object** stored in the `NewList` variable.

The **AddGridCol(ColName, Position, AllowOrderBy)** method specifies which columns to display. The example shows values relating to the company name and email address.

You can pass only field names that are returned by the SQL query.

You can 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(Arg)** method returns HTML to display the selected columns in a properly formatted list.

The **CRMListBlock 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 **AddContent(Content)** method. The **GetPage()** method is used in conjunction with `Response.Write` to display the output in the Sage CRM user interface.

Manipulating record sets

You can construct SQL strings and pass them to CRM objects to apply relational database concepts to the presentation and manipulation of CRM data.

If you're not familiar with SQL or you need a more abstract approach to handling data, use the **CRMRecord object** to create an updateable record object.

The following example calls the **CreateRecord(TableName)** method and specifies the company table as an argument:

```
<!-- #include file = "sagecrm.js"-->
<%
var Comp;
var Block;
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());
%>
```

Below are the descriptions of building blocks used in this code.

```
Comp.item('<FieldName>') = '<Value>';
```

Assigns new values to fields. This is similar to opening an ADO updateable recordset. For example, the following code samples are equivalent:

```
Comp.item ('comp_Name') = '4D Communications International';
```

```
INSERT INTO Company (Comp_Name) Values ('4D Communications');
```

You're not required to specify the item property because it's the default property for this object.

The following two lines of code are equivalent:

```
Comp.item('comp_Name') = '4D Communications International';
```

```
Comp('comp_Name') = '4D Communications International';
```

Understanding context

In Sage CRM development terms, context refers to information about the current situation of the user within the software interface. For example, a user who's looking at the Company summary screen is in the context of that company.

You can easily access contextual information, such as data from Company and Person tables, that relates directly to what the user is viewing. User table data about the current user is also available within the context framework. Use the **GetContextInfo(Entity, FieldName)** method to access information about the current context.

Scripting in the Sage CRM interface

Although server-side ASP offers full access to the CRM API, it's sometimes more convenient and quicker to handle tasks using code that runs on the client. You can include server-side and client-side scripts in the same ASP file.

Client-side scripts are processed by the browser, eliminating round-trips to the server. Client-side scripting in CRM ASP pages is handled in the normal way using JavaScript and the DOM.

Some Sage CRM scripting fields expect server-side code, while others accommodate client-side script.

- **Server-side code**. Use server-side code if you'll use objects from the CRMBlocks hierarchy. Scripts, such as SQL queries, that manipulate databases need server resources to access the specified records.
- **Client-side code**. Use client-side code for immediate responses to user actions, to access the DOM to navigate the screen interface, and to validate on screen information.
    - Use JavaScript, the DOM, the CurrentUser variable, and the ModuleCode variable to configure screens and create event handlers that respond to the current user's profile and modules available on the system.
    - Use of client-side validation saves time by checking information that's available on screen before the page is sent to the server for server-side validation. You can enter code directly through the interface so you can verify the connection between interface elements and event-handling code.
      For more information, see the *System Administrator Help*.

- For information about client-side classes and modules, see the **Client-Side API Help**.

Using field-level scripting

You can associate behaviors with individual fields using scripts that execute when a particular event occurs.

1. Log on to Sage CRM as a system administrator
2. Go to **<My Profile> | Administration | Customization | <Entity> | Screens | <ScreenName>**.
3. Select the field you want to customize from **Field**.
4. Enter code in **CreateScript**, **OnChangeScript**, or **ValidateScript**.
   The code is executed when an event affects the selected field.

Scripts in **CreateScript** run on the server and execute when the page is loaded into CRM.

Scripts in **OnChangeScript** run on the client and are executed when the JavaScript event OnChange occurs on the specified field.

Scripts in **ValidateScript** run on the server and are executed when the user clicks **Submit**. For more information, see **Validation scripts**.

Scripts in **CreateScript** and **ValidateScript** can access the CRM API without including files such as SAGECRM.JS.

CreateScript, **OnChangeScript**, and **ValidateScript** are also used for workflow and escalations. For more information, see the *System Administrator Help*.

You don't need to include statements or `<script>` tags. You can use the JavaScript `this` keyword to refer to the object that triggers the code.

Example 1

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

This example is a simple if statement for the comp_type field that disables another field if its value is *Partner*.

Example 2

```
if(this.value.toUpperCase() == this.value)
{
```

```
    window.alert(this.name + ' has been entered all uppercase');
}
```

In this example, the OnChange script alerts the user to change an entry before validation is triggered.

Validation scripts

When writing validation scripts that execute on the server, the following system variables can help you trap information and provide feedback.

- **Values( )**. Holds the inbound values of data coming into the system. It allows you to read any variable in the QueryString in CreateScripts. You can test for the dominant key and context.

  ```
  var x = Values("Key0");
  Valid = false;
  ErrorStr = x;
  ```

- **Valid**. Determines whether the ErrorStr value should be displayed on the screen. The default value is True. When set to False, it marks the current entry as invalid. Valid set to False has the following behavior in different parts of the system:
  - It displays an ErrorStr in create scripts and blocks the commitment of data in validate scripts.
  - It controls the display of workflow rule buttons in Primary, Transition, and Global workflow rules.
  - It causes a conditional rule to execute an alternative set of actions.
  - In Table Level and Entity Level scripts that update records in response to an action, it can set ErrorStr to the browser, or can block the entire transaction.
- **ErrorStr**. Returns the string in the error content bar at the top of the screen.

The following example validates a field value when a user is specifying details for an opportunity:

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

This script checks the value in the **oppo_type** field, and an action is taken if the opportunity type is **Mix**. This script only validates the value of the specified field, other fields are ignored.

If the **oppo_type** field value is invalid (`Valid = false`), an error message (`ErrorStr = 'This Mix type is temporarily not supported';`) is displayed in a red bar at the top of the screen.

This example tests onscreen information only. You can use other blocks accessible through the API to base validation on entity information not currently displayed onscreen. Also, the `CRMEntryBlock` object's properties such as `Required`, `ReadOnly`, `MaxLength`, `CreateScript`, `OnChangeScript`, and `ValidateScript` enable you to control and respond to values entered into a screen.

Using table-level scripting

You can use scripts to perform certain actions on individual fields when a record is inserted, updated, or deleted.

- Table-level scripts are executed when a record is inserted, updated, or deleted on a specified Sage CRM table.
- Entity-level scripts are executed when a Sage CRM entity is inserted, updated, or deleted.

You enter the scripts through the Sage CRM interface. For more information, see **Table- and entity-level scripts**.

Generally, you'll respond to triggers using server-side scripts that leverage the API. The triggers are: `InsertRecord()`, `PostInsertRecord()`, `UpdateRecord()`, and `DeleteRecord()`.

Example: Client-side validation

This example accesses the DOM of a CRM generated web page to capture and validate events.

The client-side JavaScript performs basic validation by checking if the last name field in the **PersonSearchBox** is empty before the search details are submitted to the server.

```
<script>

// Custom Content - personsearchbox add behaviour to the screen when it has finished loading.
crm.ready(function()

{
    // Hide the Search button.
    crm.hideButton("Search.gif");

    // Add onChange event to the pers_lastname field so that the Search button is shown when a value is entered in field.
    $("#pers_lastname").change(function ()
    {
        if (this.value != '') crm.showButton("Search.gif")
    });

})

</script>
```

For more information about the methods used in this example (such as `crm.ready`, `crm.hideButton`, and `crm.showButton`), see the **Sage CRM Client-side API Help**.

Example: Accessing user information

You can use client-side scripting to get information about the current user of the system. For that purpose, you can use the `CurrentUser` variable that is provided in the HTML code on each Sage CRM page.

This is available to the client through code contained in the include file, referenced as `<!--#include file ="sagecrm.js"-->`. 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` variable provides access to extensive information about the current user stored in the following columns in the Users table in the Sage 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 next example configures search options in a search screen according to the user's profile. All users in the Telemarketing department deal with customers in Europe, so the code automatically selects Europe from the Territories list when a member of the Telemarketing department accesses this search page.

```
<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 `CurrentUser` object ascertains the user's department. The information contained in `CurrentUser.user_department` is stored in the variable `channeldept`.

The `attachEvent` method calls the `Populate` function when the page is loaded (`"onload"`).

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

The subsequent `if` statement loops through list options to find Europe. The JavaScript method `indexOf` checks whether the current option text contains Europe. If it does, the `setIndex` variable is set and ensures that the Europe option is visible when the page is loaded.

Example: Getting information about installed modules

You can use the `ModuleCode` variable that is provided in the HTML code on each Sage CRM screen to get information about the modules installed on the system. The `ModuleCode` variable is defined in the **Eware.dll** file.

To get information about the installed modules, locate the `ModuleCode` variable in the HTML code, and then get its value.

The `ModuleCode` variable can take one of the following integer values:

- **1**. Indicates that the **Sales** module is installed.
- **2**. Indicates that the **Service** module is installed.
- **5**. Indicates that the **Sales** and **Marketing** modules are installed.
- **7**. Indicates that the **Sales**, **Marketing**, and **Service** modules are installed.

Example

```
<script>

if(ModuleCode & 1) alert('Sales module is installed');
if(ModuleCode & 2) alert('Service module is installed');
if(ModuleCode & 5) alert('Sales and Marketing modules are installed');
if(ModuleCode & 7) alert('Sales, Marketing, and Service modules are installed');

</script>
```

Gets the value of the ModuleCode variable and displays information about installed modules on the screen.

Customization

- **Using ASP pages**
- **Using customizable areas**
- **Transferring customizations to another Sage CRM instance**

For more information, see also **Using .NET API**.

Using ASP pages

To extend and modify system functionality, you can add a custom ASP page to CRM and display the page on a user-defined tab.

The Extensibility Module (EM) supports ASP, JavaScript, and Document Object Model (DOM) and provides a library of classes, methods, and properties.

The Block architecture serves as an SDK that allows you to use the CRM object, which is initialized by standard CRM include files referenced at the top of a custom file. The CRM object allows you to initialize further blocks that build up the screen interface, generate lists, manipulate database records, and modify scripts.

CRM blocks reside on the install server. Code using the CRM SDK typically runs on the server before the compiled page is dispatched to the client. ASP is used for server-side coding, as it's supported by IIS and provides six built-in objects that help you create dynamic web pages.

When a page has been downloaded from the server and displayed in the CRM system, you can use client-side JavaScript and the DOM to handle data and respond to user-generated events.

You can include server-side and client-side scripts in the same ASP file. Use server-side ASP code to access CRM Block functionality. Use client-side JavaScript for immediate responses to user actions and to access the DOM to navigate the screen interface.

Using customizable areas

To customize the following areas, use the system interface to enter scripts and change settings.

Customizable area	Without Extensibility Module	Additional customization with Extensibility Module
Fields	Create new fields and customize existing fields.	None
Screens	Customize existing screens in the Sage CRM database. Add and remove fields that were created in the system.	Add new screens.
Lists	Customize existing lists in the Sage CRM database.	Add new lists.
Tabs	Add new tabs to tab groups in the Sage 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 existing views.	None
Blocks	None	Add Blocks.
Table Scripts	None	Add Table and Entity Scripts.
Tables and Databases	None	• Connect to a new table. • Connect to a new database. • Create a new table.
Button Groups	None	Add new button groups.
Component Manager	Upload a component.	Record and script components.

Transferring customizations to another Sage CRM instance

You can transfer customizations made on one Sage CRM instance to another instance of Sage CRM. To transfer your customizations, complete the following steps:

- **Step 1: Save customizations in a component**
- **Step 2: Generate script files**
- **Step 3: Apply customizations to the target system**

You can transfer virtually any changes that you make in the **<My Profile> | Administration | Customization** area of Sage CRM. The table below lists the customizations that you can and cannot transfer between two instances of Sage CRM.

You can transfer	You cannot transfer
• 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 • Table script customizations • Translations including inline translation mode, field customization method, and translations list method • Reports including creation of new reports and modification of existing reports • Most workflow customizations • Button groups • Table and database connections • Interactive dashboards You can only record Interactive Dashboard customizations if the data sources, users, and channels are identical in the source and target Sage CRM systems. For example, if you script a dashboard gadget based on a saved search, the saved search must exist in the target Sage CRM system for the gadget to work on the dashboard.	• 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

Related ASP pages are transferred automatically only if they are directly referenced in your customization (for example, on a newly created tab). However, when an ASP page is updated, or when a file that's indirectly referenced is added (for example, an include file in an ASP page), you must manually copy these files to the component folder.

Step 1: Save customizations in a component

In this step, you configure the Component Manager, a feature of Sage CRM, to save the customizations you want to transfer to another Sage CRM instance in a component. When you save customizations in a component, they are marked in the Sage CRM database with the name of that component. Then, you can use the created component to generate a set of script files containing your customizations and use these files to apply your customizations to another Sage CRM instance.

There are two methods to save your customizations in a component:

- **Method 1: Record your future customizations**. Use this method if you haven't yet made the customizations you plan to transfer to another Sage CRM instance. This method involves a number of steps you need to complete before you start customizing Sage CRM.
- **Method 2: Retrieve existing customizations**. Use this method if you want to transfer customizations already applied to Sage CRM. With this method, you can configure criteria to retrieve specific customizations from the Sage CRM database and save them in a component. For example, you can retrieve customizations made by a specific Sage CRM user, within a certain date range, or both.

Method 1: Record your future customizations

1. Create a new component to record your future customizations:
    a. Go to **<My Profile> | Administration | Customization | Component Manager | Component Details**.
    b. Click **New**, and then use the following options:
        - **Component Name**. Enter the component name.
        - **Description**. Enter an informative description for the component.
        - **Set to Be Current Component**. Select this check box to set the new component as current. The Component Manager always uses the current component to record your customizations. You may be prompted to script the current component before setting your new component as current. If you do not script the current component, you may lose the customizations recorded previously.
          To script the current component, in the message box that opens, click **Cancel**.
2. Click **Save** to save your new component.
   Now you can make your customizations.
   When you are done, generate script files from your component as described in **Step 2: Generate script files**.

The component is set as current and listed under **Existing Components**. The current component automatically records all your customizations even if they span several days and you log in and out

of Sage CRM during that period. The screens you customize are labeled with the name of the current component that records your customizations. The current component records your customizations until you stop the Component Manager or set a different component as current.

You can stop and resume the recording as described in **Stopping and resuming recording** to record only what you need.

Stopping and resuming recording

You can stop and resume the recording to record only the customizations you need.

To stop the recording:

1. Go to **<My Profile> | Administration | Customization | Component Manager | Component Details**.
2. Click **Stop Component Manager**.
   This button is only available when the Component Manager is recording your customizations.

To resume the recording:

1. Go to **<My Profile> | Administration | Customization | Component Manager | Component Details**.
2. Click **Start Component Manager**.
   This button is only available when the Component Manager is stopped.

Changing the current component

All customizations you make are recorded against the current component. If necessary, you can change the current component.

1. Go to **<My Profile> | Administration | Customization | Component Manager | Component Details**.
2. Under **Existing Components**, click to select the component you want to set as current.
3. Click **Set To Be Current Component**.
4. If you are recording another component, you are prompted to script the current component before setting a new component as current.
   To script the current component, in the message box that opens, click **Cancel**.

Method 2: Retrieve existing customizations

You can configure criteria to retrieve existing customizations from the Sage CRM database and copy them to your new component. Then, you can use that component to generate script files and apply your customizations to another Sage CRM instance. For example, you can configure criteria to retrieve customizations created or updated by a particular user, within a certain date range, or

both. Optionally, you can search a particular existing component for the customizations that satisfy your criteria.

Use this method with caution, because the customizations you retrieve and save in your new component are removed from all the existing components where they were stored previously.

1. Go to **<My Profile> | Administration | Customization | Component Manager | Component Details**.
2. Click **Advanced Options**, and then use the following options to create a new component:
    - **Enter New Component Name**. Enter the component name.
    - **Description**. Enter an informative description for the component.
3. Under **Selection Criteria**, specify criteria to search for and retrieve the customizations you want to transfer to another Sage CRM instance.
   You can use the following options:
    - **Select And or Or to join the selection fields**. Select an operator to join your criteria.
      If you select **And**, all your criteria must be true for the customization to be retrieved and saved in your component.
      If you select **Or**, any your criterion must be true.
    - **Select to include Existing Component**. Optional. Select the existing component you want to search for the customizations you want to transfer to another Sage CRM instance.
    - **Created date**. Retrieves customizations that were created within the date range you specify. You can specify an absolute or relative date range.
    - **Updated date**.Retrieves customizations that were updated within the date range you specify. You can specify an absolute or relative date range.
    - **Created by**. Retrieves customizations created by the user you specify.
    - **Updated by**. Retrieves customizations updated by the user you specify.
4. Click **Mark Component**.
   As a result, the Component Manager searches for the customizations that satisfy your criteria. If such customizations are found, they are retrieved and saved in your new component.
   If these customizations were previously stored in any existing components, you are prompted to move them to your new component. To move customizations, click **Mark Component** once more.
   After you move the customizations to your new component, they are deleted from other components.

Now you can generate script files from your new component as described in **Step 2: Generate script files**.

If necessary, you can set your new component as current and record any additional future customizations. For more information, see **Changing the current component**.

Step 2: Generate script files

Once you have saved customizations in a component as described in **Step 1: Save customizations in a component**, use that component to produce a set of script files. You can then use these files to apply your customizations to the target Sage CRM system. The process of generating script files from a component is called **scripting**.

To generate script files from your component:

1. Go to **<My Profile> | Administration | Customization | Component Manager | Component Details**.
2. Under **Existing Components**, click to select the component that contains the customizations you want to transfer to another Sage CRM instance. You can view additional information about each component as described in **Viewing component details**.
3. Select one of the following formats for the output files:
   - **Sage CRM Script (.es) file**. Use this format to transfer your customizations to another instance of Sage CRM. This file stores your customizations as JavaScript. The human-readable comments provided in the file describe what has been customized.
     To use this format, leave the **Export as xml** check box cleared.
   - **Extensible Markup Language (.xml) file**. Use this format in scenarios that involve integration with other applications. Note that you cannot use this format to apply your customizations to another instance of Sage CRM.
     To use this format, select the **Export as xml** check box.

   To preview the customization script you are about to generate, click **Preview Script**. When you are finished, click **Continue**.
4. Click **Script Component**.
5. On the screen that opens, use the following options:
   - **File Name**. Enter the name for the output script file you want to generate. If you've already scripted the current component, enter a new file name.
   - **Description** (optional). Enter an informative description for the file. This optional information is saved in the corresponding Sage CRM Component (.ecf) file. For example, a description can remind you of customization changes you made in a particular implementation before you apply them to the target Sage CRM system.
6. Click **Script Component** to generate the script files.
   The screen that opens shows the names and locations of these files.
   Optionally, you can modify the generated script files as described in **Modifying script files**.
7. Add the generated script files to a .zip archive.
   Now you need to apply your customizations as described in **Step 3: Apply customizations to the target system**.

On a Sage CRM computer, you can find the generated script files in

%ProgramFiles(x86)%\Sage\<InstallPath>\<InstallName>\inf

The **inf** folder holds the Sage CRM Component (.ecf) file and the component folder named after the component. The component folder contains the Sage CRM Script (.es) file and any other corresponding files, for example, ASP pages. To transfer your customizations, make sure to add all these files to the .zip archive.

The Component Manager automatically copies files stored in the CustomPages folder of Sage CRM only if these files are included in a tab group with the customfile action. If your customization files include other files such as .asp, .dll, .txt, .js, or .inc, manually copy them into the component's CustomPages folder when the component scripting is finished.

On a Sage CRM server, you can find the **CustomPages** folder in the following location:

%ProgramFiles(x86)%\Sage\<InstallPath>\<InstallName>\WWWRoot

Viewing component details

For each component, you can view such information as component name, description, created date, who created the component, whether the component is set as current, and when the component was scripted last time. Optionally, you can change the component description.

1. Go to **<My Profile> | Administration | Customization | Component Manager | Component Details**.
2. Under **Existing Components**, click to select the component whose details you want to view.
3. Click **View Details**.
   The screen that opens shows the component details. You can optionally click **Change** to modify the component description.

Previewing script files

Before generating the customization script from a component, you can preview that script.

1. Go to **<My Profile> | Administration | Customization | Component Manager | Component Details**.
2. Under **Existing Components**, click to select the component for which you want to preview the customization script.
3. Click **Preview Script**.
   When you are finished, click **Continue** to return to the **Component Details** tab.

Modifying script files

You can modify the Sage CRM Component (.ecf) and Sage CRM Script (.es) files generated by the Component Manager:

On a Sage CRM server, you can find these files in:

%ProgramFiles(x86)%\Sage\<InstallPath>\<InstallName>\inf

For example, you can specify additional parameters in the .ecf file or modify JavaScript in the .es file to do the following:

- 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 the following views:
    - **IDatabase**. Returns the current Sage CRM database.
    - **ILocale**. Returns the installed locale. IWestern and IJapanese are two constants that can be returned.
    - **sViewText**. Provides a temporary storage buffer when scripting views.

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

Make sure you order your code correctly when modifying the Sage CRM Script (.es) file. For more information, see **Observing referential integrity**.

When modifying scripts, you can pass the following parameter types to the .es script file:

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

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>
```

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 install the component, the fields are displayed on the Parameter Info screen with the attributes you specified. To use the values entered, call the `Param()` method in the .es script file. For example, to retrieve the value in the **Enter Server Name** text box, call `Param(ServerName)` in the .es script file.

You can specify the following attributes for each parameter:

Attribute	Required	Description
Name	Yes	Specifies the field name. You can use this attribute with the Param function to return the value entered by the user on the Parameter Info screen.
Default	No	Specifies the default value for the parameter.
NewLine	No	Allows you to specify the line on which to place the parameter. This attribute can take one of the following values: • **False**. Specifies to keep the parameter on the same line as the previous one. • **True** (default). Specifies to place the parameter on a new line.
Rows	No	Specifies the number of rows that the parameter spans. The default value is 1.

Attribute	Required	Description
Cols	No	Specifies the number of columns that the parameter spans. The default value is 1.
Required	No	Specifies if the parameter is required. This attribute can take one of the following values: • **True**. Specifies that the user must enter a value for the parameter. • **False**. Specifies that the parameter is optional. Validation is done when the user clicks **Install Component**.
ReadOnly	No	Specifies if the parameter is read-only. This attribute can take one of the following values: • **True**. Specifies that the parameter is read-only. • **False**. Specifies that the parameter is writable.
Size	No	Specifies the maximum length of the field that contains the parameter value (in characters). The default value is 20. This attribute is only applicable to parameters of type `TEXT`.

Attribute	Required	Description
MaxLength	No	Specifies the maximum number of characters in the parameter value. The default value is 40.
Caption	No	Specifies the text label (caption) for the user interface field that shows the parameter value.
CaptionPos	No	Specifies the position of the text label (caption) relative to the field that shows the parameter value.
OnChange	No	Specifies JavaScript to run when a user changes the value in the field.
Attribute=Family	No	Specifies what caption family to use to get the drop-down list. This attribute is only applicable to parameters of type `SELECT`, this specifies .

Observing referential integrity

The architecture implemented in Sage CRM 7.0 and later facilitates the Interactive Dashboard. To allow the persistence of the Meta Data Model within this architecture, make sure you observe strict referential integrity within Sage CRM metadata tables.

This means you must correctly order the code in components in the Sage CRM Script (.es) file. For example, you can't add a view if the table the view is based on hasn't 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)

You cannot use the `AddCustom_Data` or `RunSQL` method to update the tables listed above, as the foreign keys are not set automatically. Use the table appropriate method instead. For example, to update Custom_Edits use `AddCustom_Edits`.

Setting how to handle script errors

By default, when you use the Sage CRM Script (.es) file to transfer your customizations, all script errors are handled in the following manner:

Whenever an error occurs, the script keeps running and applies all other changes specified in the .es file.

You can change this default behavior and configure your script to roll back all changes already made whenever an error occurs. To do so, add the `errorhandling=strict` line to the corresponding Sage CRM Component (.ecf) file.

Example

```
Description=My component
Directory=My component
Version=7.3
errorhandling=strict
```

Scripting multi-stage customizations

When using the Component Manager to record your customizations, you can create multiple sets of files, each containing specific changes. For example, you can use this method to record complex, multi-stage customizations and then apply them to the target Sage CRM system in stages.

1. Create a new component and set it as the current component.
2. Save it and begin making customization changes. When you want to script your changes, click **Script Component**.
3. Enter a name for the scripted component files in **File Name** and enter a description in **Description**.

4. Click **Script Component** and then click **Continue**.The component name you originally specified is still the current component on the Component Details page.
5. Make additional customization changes. These changes are recorded as part of the current component, in addition to changes you've already recorded and scripted.
6. Specify a new file name and click **Script Component**. The generated component files contain the changes you made in the first part of the customization and your recent changes. You can continue to script changes in this way as many times as you need to, but you must use a different, meaningful file name each time you script.
7. When you've fully completed the customization, perform a final scripting, keeping the current component name in **File Name**. You should always script the current component before beginning a new customization.
8. When you're installing the script to the second Sage CRM system, select **Apply All Changes** to install all changes in the final version of the component.

Code samples

This section provides examples of how to modify your scripts to do the following:

- **Enabling a component for multiple installs**
- **Changing a screen name**
- **Adding a message**
- **Copying an ASP page**
- **Replacing text in an ASP page**
- **Creating a new table**
- **Adding a new column**
- **Using the DataFile object**
- **Adding a new view**

Enabling a component for multiple installs

You can specify that a component can be used for multiple installs. This is useful if you install a component, make further customization changes, and then want to undo them. Rather than undoing the changes manually, you can reinstall the component on a clean Sage CRM install.

1. Open the .ecf file and add the following code:

```
multipleinstalls=y
```

2. Save the file.

Changing a screen name

This example changes a screen name from DemoScreen1 to Demo.

1. Open the .ecf file and enter the following:

```
Params:
Text Name=ScreenName,Caption=Type new screen name here,Required=True
```

2. Open the .es script file and change the following script.

   From:

```
AddCustom_ScreenObjects('DemoScreen1','Screen','Opportunity', 'Y','0','','','');
```

   To:

```
AddCustom_ScreenObjects(Param('ScreenName'),'Screen','Opportunity', 'Y','0','','','');
```

3. Save both files and install the component.
4. Enter *Demo* in **Screen Name** on the Parameter Info screen. For more information about adding fields to this screen, see **Modifying script files**.
5. Click **Continue**. When you install the component, DemoScreen1 is renamed to Demo and the record is added to the custom ScreenObjects table.

Adding a message

This example adds a message to the end of an installation.

1. Open the .ecf file and enter the following.

```
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 install the component.

4. Enter *Demo* in **Screen Name** on the Parameter Info screen. For more information about adding fields to this screen, see **Modifying script files**.
5. Click **Continue**. When you install the component, the following message is displayed at the end of the installation: **A new screen called Demo was installed into CRM**.

Copying an ASP page

You can 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 is copied from the Phase1 component directory to the following location in the current installation: \custompages\subfolder\edit.asp.

Replacing text in an ASP page

You can 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 the file **Edit.asp** are replaced with the word **Search**.

Creating 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 is added to Sage CRM. The following columns are automatically created for the table:
    - Dem_TableId
    - Dem_ System
    - Dem_ CreatedBy
    - Dem_ CreatedDate
    - Dem_ UpdateBy
    - Dem_ UpdateDate

- Dem_ TimeStamp
- Dem_ Deleted

Adding 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** is added to the DemoTable created in **Creating a new table**.

Using the DataFile object

This example uses 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.xlsx');
var i = 0;
while (!MyFile.EOF)
{
    i = 0;
    while (i < MyFile.FieldCount)
    {
        sValue = MyFile.GetField(i);
        //do something with value
        i++;
    }
    MyFile.NextRow();
}
```

Adding a new view

This component script adds a view. It uses the iDatabase variable. Open the .es script file and add the following script.

```
sViewText="CREATE VIEW vMyPhone AS SELECT";
if (iDatabase == IOracle)
{
    sViewText = sViewText + " Phon_CountryCode || N ' ' ||Phon_AreaCode || N ' ' || Phon_Number";
}
else
{
    sViewText = sViewText + " RTRIM(ISNULL(Phon_CountryCode, '')) + ' ';
    sViewText+= "RTRIM(ISNULL (Phon_AreaCode, '')) + ' ' + RTRIM(ISNULL(Phon_Number, ''))";
}
sViewText +=" AS Phon_FullNumber, Phone.* FROM Phone ";
sViewText += "LEFT JOIN Custom_Captions ON Phon_Type = Capt_Code WHERE";
```

```
sViewText += " Phon_Deleted IS NULL";
AddView("vMyPhone", "Phone", "This selects all of the phone numbers", sViewText, false, false,
false, false, false
```

Step 3: Apply customizations to the target system

In this step, you use the .zip file prepared in **Step 2: Generate script files** to apply your customizations to the target Sage CRM system. Note that your .zip file may contain script files for more than one component.

To apply your customizations, do the following:

- **Step 1: Upload component from the .zip file**
- **Step 2: Install component**

Step 1: Upload component from the .zip file

1. Log on to the target Sage CRM system as a system administrator.
2. Go to **<My Profile> | Administration | Customization | Component Manager | Components**.
3. Under **Add Component**, specify the .zip file you prepared in **Step 2: Generate script files**.
4. Click **Upload New Component**.

Step 2: Install component

1. Under **Available Components**, click to select the component you have just uploaded. You can click **View Details** to view additional information about the component.
2. Click **Install Component**.
    - If you didn't modify the script to include parameter information, ignore the message stating "No parameter information found in [component name]". For more information about adding parameter information, see **Modifying script files**.
    - If you included parameters, complete the fields on the Parameters, Step 1 of 2 screen. Click **Install Component** to continue installing the component. The Parameters, Step 1 of 2 page might appear without fields, but with information about the component you're installing.
3. Click **Preview Install** to view the script that's executed when the component is installed and a prediction of whether each step will be successful.
4. Select **Yes** or **No** in **Apply All Changes**. These options apply to changes made by previous components to the objects that are being changed by the current component.

- To install everything in the component and overwrite any changes from previous components, select **Yes**.
- To preserve changes from previous components, select **No**. Select **No** only when you've a specific reason for doing so as it may result in your component being only partially installed.

5. Click **Install**. Component Manager loads the new information, recreates the views, and reloads the metadata. When the Component Installed message appears, you can view the log file that has been generated. Otherwise click **Continue** to return to the Components tab.

When the component installation completes, you can view the component installation log to see the changes made or issues encountered during the installation.

Considerations for transferring workflows

When applying customizations to the target Sage CRM system, consider the following:

When...	**And you transfer customizations to target...**
Workflow with the same name exists both in source and target.	Workflow from source overwrites the workflow in target. **Exception:** Deleting workflow rules or states in source does not affect the corresponding rules or states in target.
Workflow only exists in source.	Workflow is created in target.
Workflow only exists in target.	Workflow is left intact.

Viewing component installation log

A log file is automatically generated when you install a component. The log file is saved as a Comma-separated Values (.csv) file.

To view the component installation log, do the following:

1. Wait until component installation completes, and then click **View Log File** to download the corresponding log file.
2. Open the downloaded .csv file.

Alternatively:

1. Go to **<My Profile> | Administration | System | Logging**.
2. From **Select log files**, select **Component Install Logs**.

3. In the **View Log** table column, click the icon to download the corresponding log file.
4. Open the downloaded .csv file.

The log file contains two columns and a row for each action attempted during the component manager installation. The first column contains one of the following values:

- **OK**. Indicates that the action completed successfully.
- **Overwrite**. Indicates that the action overwrote a previous change.
- **Fail**. Indicates that the action failed.

The second column contains the script command that was run.

Objects and blocks

Objects

Sage CRM object	Description
Dispatch	Controls all web requests and responses, finds the relevant UserSession, and sets the session keys. User requests are sent to the application's Web module, which creates an object to process the request, and output the relevant response. You can't access the Dispatch object because it's internal to Sage CRM.
CRM	Provides basic access to Sage CRM objects and functionality. Use the methods of this object to create new objects, get existing objects, and execute objects.
CRMBase	Provides functionality that's only applicable in the Sage CRM environment, such as the current company. 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	Used to create target lists also known as groups in Sage CRM 7.2 and later. To ensure that legacy code works with new installations, the term *target lists* is maintained in the API terminology.
CRM TargetListFields	A container for a list of TargetListField objects.
CRM TargetListField	Fields that are displayed on a target list.
CRMSelfService	Contains methods and properties that enable self service users to access relevant information from the Self Service web site.
MsgHandler	Used to customize scripts deployed by E-mail Management. It provides access to the Email Object.
Email	Used to customize scripts deployed by E-mail Management. It provides access to the email.
AddressList	Used to customize scripts deployed by E-mail Management. It provides access to To, CC, and BCC lists of addresses.

Sage CRM object	Description
Mail Address	Used to customize scripts deployed by E-mail Management. It provides access to individual addresses from the AddressList object.
AttachmentList	Used to customize scripts deployed by E-mail Management. It provides access to email attachments.
Attachment	Used to customize scripts deployed by E-mail Management. It provides access to individual email attachments from the AttachmentList object.
CRMRecord	Represents records in a table. An enumerator that returns all the specified fields in a table. Use the 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 is possible with the CRM Record object.
CRMBlock object	The base of all CRM blocks. This block determines the actual implementation of each CRMBlock method and property.
CRMContainerBlock object	Used to group other blocks on a screen. This block contains the standard Sage CRM buttons. You can configure workflow buttons on the screens where they'll be displayed. An example of a container block is a linked search panel and related list.
CRMEntryGroupBlock object	Used to group entries to create a screen. You can generate many types of entry group, such as a Company Search Box, a Person Entry Box, and a Case Detail Box. This block contains the standard Sage CRM buttons.
CRMListBlock object	Generates a custom list from columns in a Sage CRM table, or a table connected to Sage CRM through **<My Profile> | Administration | Advanced Customization | Tables And Databases**.
CRMEntryBlock object	Corresponds to a single field that's displayed or edited on screen. You can generate many entry types, such as text blocks, multi-select boxes, and currency input boxes. You typically add Entry blocks to EntryGroups or Containers. Use JavaScript scripts on these blocks to perform tasks when they load, change, or are validated.

Sage CRM object	Description
CRMGridColBlock object	Corresponds to a single column in the List block. Use the GridCol block to change properties on individual columns in a list.
CRMMarqueeBlock object	Adds scrolling text to a page. The content of the text is maintained through **<My Profile> | Administration | Customization | Translations**. 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 aren't part of Sage CRM and enables files to appear as if they are part of Sage CRM.
CRMMessageBlock object	Allows you to send messages in email and SMS format. Include this block in ASP pages to display a simple email form or to automate the message sent in response to a certain event.
CRMContentBlock object	A 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	Displays images through an ASP page. Use variables to represent live data from a database or incorporate details of the current user, such as user privileges or settings.
CRMChartGraphicBlock object	Displays a variety of charts which can be generated from data retrieved using SQL or added through an ASP page. It inherits all the functionality of the Graphics block.
CRMOrgChartGraphicBlock object	An implementation of the Graphics block used for organizational charting. These charts may depend on data retrieved through SQL or added through ASP. It inherits Graphics block functionality.
CRMPipeLineGraphicBlock object	Creates cross-section diagrams representing data from an ASP page or stored in a table. It inherits Graphics block functionality.

Blocks

About blocks

You need the Extensibility Module (EM) to create and customize blocks in **<My Profile> | Administration | Customization | <Entity> | Blocks**.

All lists, screens, and fields that you create are blocks within the system (ListBlock, EntryGroupBlock, and EntryBlock). New blocks must be based on an existing standard CRM block. Fo example, Container, EntryGroup, or Marquee. Associate all new blocks with the entity to which they relate.

You can create and display new blocks of data from external tables and databases to which Sage CRM is connected. For more information about customizing tables, see **Database**.

When you create a new block, you can reference and run it from Sage CRM and in ASP pages.

Referencing block names

Once you've determined the exact name of a block, you can call it from an ASP page. There are several types of block names.

Block name type	Description
Generic name of the block	All blocks are based on a CRM basic block type. To reference CRM generic block names, go to **<My Profile> | Administration | Customization | <Entity> | Blocks** and click **New**. The custom block names are listed in **Block Type**. 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 any custom screen (Entry block)	A standard installation includes a number of predefined custom screens that you can manipulate or copy. To reference predefined custom screen names, go to **<My Profile> | Administration | Customization | <Entity> | Screens**. The screen names are displayed in **Screen Caption**. Alternatively, reference the name from the Custom Tables in your database. For example, if you're using SQL Server, the block names are in the database in Enterprise Manager.
Name of any custom list (List block)	A standard installation includes a number of predefined custom lists that you can manipulate or copy. To reference predefined custom list names, go to **<My Profile> | Administration | Customization | <Entity> | Lists**. The list names are displayed in **List Name**. Alternatively, reference the name from the Custom Tables in your database.
Name of any block that you have created	You can create your own blocks from the system interface.

Creating a block

All new blocks you create in CRM must be based on Sage CRM custom blocks. You can then customize the properties of the block.

You can customize any lists or screens that you create 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.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <entity> | Blocks.**
3. Click **New**.
4. Enter the block name in **Block Name**. This is the name you use to reference the block in ASP pages.
5. Enter the block type in **Block Type**.
6. To copy an existing block, select it from **Copy Existing Block**.
7. Select the screen or list with which the new block is associated from **Use Group**.
8. Click **Save**. The new block appears in the list of available blocks for the entity.

To access the new block within an ASP page, use the **GetBlock(BlockName)** method. The object returned by the **GetBlock(BlockName)** includes the specified properties.

You can also create a block in an ASP page linked to CRM. For more information, see **Creating an ASP page**.

Customizing a block

You can change the properties of an existing block or specify the properties of a new block.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <entity> | Blocks**.
3. Click the block you want to customize. The available properties depend on the block type. For more information, see **ASP objects**.
4. Enter new properties or change the existing ones.
5. Click **Save**.

Displaying a block

You can display a new block on a new tab using runblock to directly run the block. You can run Content, Marquee, Message, and Chart blocks using runblock. You can also run any EntryGroupBlock based on a current entity screen using runblock.

Alternatively, you can display a new block on a new tab using customfile to link to an ASP page that references the block.

Lists

- **Creating a list**
- **Customizing a list**
- **Displaying a list using runblock**
- **Displaying a list using an ASP page**

Creating a list

You can create a new list in Sage CRM from columns in existing tables or external tables connected to CRM.

1. Log on to Sage CRM as a system administator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Lists**.
3. Click **New**.
4. Enter the list name in **Name**. This is also the block name that you use to reference the list in ASP pages.
5. Enter the table or view that contains the list columns in **Table or View to Select Fields From**.
6. Enter the name of the filter box used to search the list in **Filter Box Name**.
7. Click **Save**. The new list is displayed.

You can also create a list in an ASP page linked to CRM. For more information, see **Creating an ASP page**.

Customizing a list

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Lists**.
3. Click the list you want to customize.
4. Make your changes and click **Save**.

For more information about customizing lists, see the *System Administrator Help*.

Displaying a list using runblock

You can display a list directly from a new tab using the runblock tab action. 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 maintains the context for the current entity.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <entity> | Tabs**.
3. Click the tab group to which you want to add the tab.
4. Enter the tab name in **Caption**.
5. Select **runblock** from **Action**.
6. Enter the list name in **Block Name**.
7. Click **Add** and then **Save**.

To view the list, click the new tab.

Displaying a list using an ASP page

You can display a list created from an ASP file by linking to the file from a custom tab. This option lets you set the properties of the list before displaying it.

1. Create your .asp file and add it to Sage CRM. For more information, see **Creating an ASP page** and **Adding an ASP page to Sage CRM**.
2. Log on to Sage CRM as a system administrator.
3. Go to**<My Profile> | Administration | Customization | <Entity> | Tabs**.
4. Click the tab group to which you want to add the tab.

5. Use the following options:
    - **Caption**. Enter a name for the tab.
    - **Action**. Select **customfile**.
    - **Custom File**. Enter the name of your .asp file.
6. Click **Add** and then click **Save**.

To view the list, click the new tab.

Screens

- **Creating a screen**
- **Customizing a screen**
- **Displaying a screen using runblock and screen name**
- **Displaying a screen using runblock with a custom block**
- **Displaying a screen using an ASP page**

Creating a screen

You can create a new screen in Sage CRM from columns in existing tables or external tables connected to CRM.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Screens**.
3. Click **New**.
4. Select a screen type from **Screen Type**.
   If you select Search Screen, an additional field called **Associated List** is displayed on the New Screen Definition page which allows you to associate a list with the screen.
5. Enter the name of the screen in **Screen Name**. This is the name used to reference the screen in ASP pages.
6. Enter the actual name that's displayed on the screen in **Screen Caption.**
7. Enter the name of the view that contains the screen fields in **Associated View**.

8. If you're using fields from a foreign table, do the following:
    a. Enter the table name in **Foreign Table**.
    b. Specify the column that uniquely relates the foreign table to the Sage CRM table in **Foreign Table Column**.
    This column has a corresponding field on the Sage CRM table.
9. Click **Save**.

You can create a block from the new screen and use the block properties to customize the appearance of the screen.

You can also create a screen in an ASP page linked to CRM. For more information, see **Creating an ASP page**.

Customizing a screen

When you've created a screen, you must define screen fields.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <entity> | Screens**.
3. Click the screen you want to customize.
4. Add fields to the screen and click **Save**.

Displaying a screen using runblock and screen name

You can display a screen directly from a new tab using the runblock tab action. 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 maintains the context for the current entity.

For example, suppose you create a new entry screen for the Company table to edit extra data relating to companies. You can add this screen to the Company tab group, but it won't work on any other tab group.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Tabs**.
3. Click the tab group to which you want to add the tab.
4. Enter the tab name in **Caption**.
5. Select **runblock** from **Action**.
6. Enter the screen name in **Block Name**.
7. Click **Add** and then **Save**.

To view the screen, click the new tab.

Displaying a screen using runblock with a custom block

You can display a screen by creating a block from the screen, 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. The tab group to which you add the block must be the tab group for the entity on which the block is based.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Blocks** and click **New**.
3. Enter the block details, select the screen name from **Use Group**, and click **Save**.
   The block is displayed in the list of blocks for the entity.
4. Click the name of the block you want to customize.
5. Enter properties and click **Save**.
6. Go to **<My Profile> | Administration | Customization | <entity> | Tabs** and select the block context.
7. Click the tab group to which you want to add the tab.
8. Select **runblock** from **Action** and enter the block name in **Block Name**.
9. Click **Add** and then **Save**.

To view the screen, click the new tab.

Displaying a screen using an ASP page

You can display a screen created from an .asp file by linking to the file from a custom tab. This option lets you set the properties of the screen before displaying it.

1. Create the .asp file and save it in the Custom Pages folder of your Sage CRM installation directory.
2. Log on to Sage CRM as a system administrator.
3. Go to **<My Profile> | Administration | Customization | <Entity> | Tabs** and select the block context.

4. Click the tab group to which you want to add the tab.
5. Enter a name for the tab.
6. Select **customfile** from **Action**.
7. Enter the name of the ASP page in **Custom File**.
8. Click **Add** and then click **Save**.

To view the screen, click the new tab.

Buttons

- **Creating button groups**
- **Adding buttons to button groups**
- **Displaying button groups**
- **Restricting access to button groups**

Creating button groups

You can define button groups that appear on specific Sage CRM screens to access custom functionality. A button group acts as a placeholder for displaying buttons.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | Button Groups**.
3. Click **New**.
4. Enter the name of the new button group in **Name**.
5. From **Select Action**, select the Sage CRM screen on which you want the button group to appear.

   > **Note:** You can define only one button group per Sage CRM screen.

6. Click **Save**. The new button group is displayed on the **Button Groups** list.

Adding buttons to button groups

When you've created a button group, you can add buttons that are displayed on the relevant Sage CRM screen.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | Button Groups**.
3. Click the button group to which you want to add the buttons.
4. Enter the button name in **Caption**.
5. Select **Customfile** from **Action**.
6. Specify the associated ASP file in **Custom File**.
7. Select the button image from **Bitmap**.
8. Click **Add**. The new button is displayed under **Desktop HTML Button Group Contents**. Use the up and down arrows to position the button relative to other buttons you've created.
9. Click **Save**. The new button is displayed on the summary screen.

Displaying button groups

Buttons that you've added to a button group appear automatically on the relevant Sage CRM screen. The button group is displayed on the right-hand side of the page above the **Help** button.

Restricting access to button groups

When you create or edit a button group, you can use an SQL statement to limit access to individual buttons in the group.

Buttons in the button group adhere to a user’s existing security rights. For example, a user must have at least View rights to Cases to open a button group which displays a list of cases.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | Button Groups**. The Button Groups list is displayed.
3. Click the button group you want to change.
4. Select the button to which you want to limit access in the **Desktop HTML Button Group Contents** area.
5. Enter the limiting SQL statement in the SQL field. For example:

```
User_PrimaryChannelId = 3
```

6. Click **Update** and then click **Save**.

Classic Dashboard

Classic Dashboard is only available if you upgraded Sage CRM from a pre-7.2 version.

- **Customizing the Classic Dashboard**
- **Adding a List block to the Classic Dashboard**
- **Adding a block to the Interactive Dashboard using the Contents field**
- **Adding a Chart block to the Classic Dashboard**

Customizing the Classic Dashboard

You can create and customize content for the Classic Dashboard using List, Chart, and Content blocks.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Blocks**.
   A list of existing blocks for the entity is displayed.
3. Click the block that you want to customize. The Maintain Block definition page is displayed.
4. Enter information in the fields listed below and click **Save**.

Field	Description	Applies to
Width	Leave this field blank so the block fills the available space.	• List Block • Chart Block • Content Block
Height	Leave this field blank so the block fills the available space.	• List Block • Chart

Field	Description	Applies to
		Block • Content Block
Interactive/ Classic Dashboard Level	To display a block in the list of Available Content on the dashboard, select one of the following options: • **Available In Dashboards**. Includes the block in Available Content for all users. • **Dashboard - Info Manager**. Includes the block in Available Content for all users with security administration rights set to Info Manager or System Administration. • **Dashboard - Administration Only**. Includes the block in Available Content for all users with security administration rights set to System Administration.	• List Block • Chart Block • Content Block
Double Width Block	Select this option to spread the block over two of the three columns on the Classic Dashboard.	• List Block • Chart Block • Content Block
Long Block	Select this option to set a longer maximum length for the block.	• List Block • Chart Block • Content Block
Dashboard Conditional	Enter a filter for records. For example, the My Cases block contains the following condition 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	Custom content. For example, content from external web sites. When you enter HTML (or Javascript within `<script>` tags), the content is displayed on the Classic Dashboard.	Content Block

For an introduction to classic and interactive dashboards, see the *User Help*. For information on how to set up standard classic dashboards for users, see the *System Administrator Help*.

Adding a List block to the Classic Dashboard

You can add a List block to the Classic Dashboard Content page. For more information about this page, see the *User Help*.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Blocks**.
   A list of existing blocks for the entity is displayed.
3. Click **New** and then use the following options:
   - **Block Name**. Enter a name for your block.
   - **Block Type**. Select **List Block**.
   - **Copy Existing Block**. Select the existing block you want to copy. Alternatively, select a list from **Use Group**.
   - **Use Group**. Select the existing list you want to use.
4. To order items in your list, open the list definition and select the field that you want to sort on. Select **Yes** from **Default Order By**. Note that **Allow Order By** has no effect when the list is viewed on the Classic Dashboard.
5. Click **Save**. The new block is displayed in the list of existing blocks for the current entity.
6. Click the block you want to customize.
7. From **Classic Dashboard Level**, select **Available In Dashboards**.
8. Click **Save**.

When users click **Modify Dashboard** on the **Classic Dashboard** tab, the newly created block is displayed in **Available Content**.

Adding a Content block to the Classic Dashboard

You can add a Content block to the Classic Dashboard Content page. For more information about this page, see the *User Help*.

1. Create an ASP page to display the content for the Classic Dashboard and save it in the Custom Pages folder.
   If you create an ASP page that uses the **GetPage()** method to display content, and you want to suppress the tab group, pass the none parameter to the GetPage method.
   From Sage CRM version 7.2b and later, all ASP pages must use the **AddContent(Content)** and **GetPage()** methods to build the HTML for the page. This is to ensure the correct

rendering of the page structure and links including the left hand main menu, horizontal tabs and top content.

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

2. Log on to Sage CRM as a system administrator.
3. Go to **<My Profile> | Administration | Customization | <Entity> | Blocks**.
   A list of existing blocks for the entity is displayed.
4. Click **New** and then use the following options:
   - **Block Name**. Enter a name for your block.
   - **Block Type**. Select **List Block**.
5. Click **Save**. The new block is displayed in the list of existing blocks for the selected entity.
6. Click the block you want to customize.
7. From **Classic Dashboard Level**, select **Available In Dashboards**.
8. In **Contents**, add the following code and click **Save**:

```
<script>
window.attachEvent("onload",callPage("test.asp"))
function callPage(strPageName)
{
    var SID = "";
    var strPath = document.URL;
    var arrayApp = strPath.split("eware.dll");
    var arrayFullKeys = strPath.split("?");
    var arrayKeys = arrayFullKeys[1].split("&");
    for(var i=0;i<arrayKeys.length;i++)
    {
        var arrayValue = arrayKeys[i].split("=");
        if (arrayValue[0].toLowerCase()== "sid")
        {
            SID = arrayValue[1];
        }
    }
    var strNewPath = arrayApp[0] + "CustomPages/" + strPageName +"?SID="+SID+GetKeys()
    document.write("<IFRAME WIDTH=100% src='"+strNewPath+"' height=height=120 width=120
frameborder=0 scrolling = 'no'>");
    document.write("</IFRAME>");
}
</script>
```

The example above contains a JavaScript function called `callPage`. The ASP page name (tesp.asp) is passed as a parameter to the function and it builds the path including the correct session and context information that allows the ASP page to use CRM blocks.

When users click **Modify Dashboard** within the **Classic Dashboard** tab, the newly created block is displayed in **Available Content**.

Adding a Chart block to the Classic Dashboard

You can add a Chart block to the Classic Dashboard Content page. For more information about this page, see the *User Help*.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Blocks**. A list of existing blocks for the entity is displayed.
3. Click **New** and then use the following options:
    - **Block Name**. Enter a name for your block.
    - **Block Type**. Select **Chart Block**.
    - **Copy Existing Block**. To copy an existing block, select the block. Alternatively, leave this option blank.
4. Click **Save**.
   The new block is displayed in the list of existing blocks for the current entity.
5. Click the block that you want to customize.
6. From **Classic Dashboard Level**, select **Available In Dashboards**.
7. Click **Save**.

When users click **Modify Dashboard** on the **Classic Dashboard** tab, the newly created block is displayed in **Available Content**.

Interactive Dashboard

Customizing the Interactive Dashboard

You can create and customize content for the Interactive Dashboard using content blocks.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Blocks**.
   A list of existing blocks for the entity opens.
3. Click the block that you want to customize.
4. Enter information for the fields listed below and click **Save**.

Field	Description
Dashboard Level	To display a block in the list of Available Content on the dashboard, select one of the following options: • **Available In Dashboards**. Includes the block in Available Content for all users. • **Dashboard - Info Manager**. Includes the block in Available Content for all users with Security Administration rights set to Info Manager or System Administration. • **Dashboard - Administration Only**. The block is included in Available Content for all users with Security Administration rights set to System Administration.
Contents	Custom content. For example, content from external web sites. When you enter HTML (or JavaScript within `<script>` tags), the content is displayed on the Interactive Dashboard.

For an introduction to classic and interactive dashboards, see the *User Help*.

Adding a block to the Interactive Dashboard using the Contents field

You can make a Content block available on the Interactive Dashboard by pasting your HTML code into the **Contents** field on the Maintain Block Definition page.

1. Prepare your HTML code and copy it to the Clipboard.
2. Log on to Sage CRM as a system administrator.
3. Go to **<My Profile> | Administration | Customization | Secondary Entities | System | Blocks**.

4. Click **New** and then use the following options:
    - **Block Name**. Enter a name for your block.
    - **Block Type**. Select **Content Block**.
5. Click **Save**.
6. Click the new block and then from **Interactive/Classic Dashboard Level** select **Available in Dashboards**.
7. Paste the HTML code you prepared in step 1 into **Contents** and click **Save**.
8. To display the block as a larger sized block on the Classic Dashboard, select the **Double Width Block** and **Long Block** check boxes.
   For more information, see **Customizing the Classic Dashboard**.
9. To test the block, go to **My CRM | Dashboard** and create a new web site gadget on an existing or new dashboard.
10. Select the new content block from **Content Block** and complete the gadget wizard.
    The gadget appears on the dashboard, showing the content block that you created.

Displaying an ASP page in a gadget

You can display an ASP page in a web site gadget on the Interactive Dashboard.

1. Create an ASP page called **test.asp** and save it in the **Custom Pages** folder. It should contain the following code, where `<instancename>` is the Sage CRM instance name and `testNew.asp` is the ASP page to be displayed in the web site gadget:

```
<%@ CodePage=65001 Language=JavaScript%>
<%
function GetKeyValue(querystringname)
{
 var strPathFull = String(Request.ServerVariables("HTTP_REFERER"));
 var arrayPath = strPathFull.split("?");
 var strPath = arrayPath[1];
 var arrayKeys = strPath.split("&");
 for (var i=0;i<arrayKeys.length;i++)
 {
  var arrayValue = arrayKeys[i].split("=");
  if (arrayValue[0].toLowerCase()== querystringname.toLowerCase())
  {
   return arrayValue[1];
  }
 }
 return "";
}
Response.Redirect
("http://localhost/<instancename>/CustomPages/testNew.asp?SID="+GetKeyValue
("SID")+"&J=testNew.asp&T=User&PopupWin=Y")
```

```
%>
```

If you create an ASP page that uses the **GetPage()** method to display content, and you want to suppress the tab group, pass the none parameter to the **GetPage()** method.
In Sage CRM version 7.2b or later, all ASP pages must use the **AddContent(Content)** and **GetPage()** methods to build the HTML for the page. This is to ensure the correct rendering of the page structure and links including the left hand main menu, horizontal tabs, and top content.

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

2. Log on to Sage CRM as a system administrator.
3. Click **Dashboard and open the relevant dashboard| New Gadget**.
4. Click **New Gadget | Create Gadget**.
5. Click **Web Site** and enter *#crm_server#/CustomPages/test.asp* in **Web Address**.
6. Click **Next** and enter a name and description for the gadget.
7. Click **Finish**. The ASP page is displayed in the gadget.

Adding a third-party gadget to the Interactive Dashboard

You can make a third-party gadget available to Sage CRM users through the Interactive Dashboard.

Third-party gadgets can communicate with other gadgets in Sage CRM through the Event Channel, passing data in JSON format. Third-party gadgets are set up as web site gadgets, which link to a file in the **wwwroot\staticcontent** folder. You can use them in My CRM or Company context. Third-party gadgets can send or receive data from other gadgets, which means they can set the filter or be filtered by other gadgets.

Sage CRM exposes the following methods that allow you to create gadgets using JavaScript:

- **scrmGetSageCRMOwner method**
- **scrmRegisterEvent method**
- **scrmPublishEvent method**
- **scrmGetGadgetProperty method**
- **scrmSetGadgetProperty method**
- **scrmSaveGadgetProperties method**

For a custom gadget example, see **Code sample: Custom gadget** .

The following example demonstrates how to link a third-party gadget to a Company List and a Company Summary gadget:

1. Copy the third-party gadget file to one of the following location:

   %ProgramFiles(x86)%\Sage\CRM*<Installation Name>*\WWWRoot\StaticContent

   The default installation name is **CRM**.
2. Set up a List gadget and a Record Summary gadget that use company data.
3. Add a Web Site gadget which links to the third-party gadget file you have copied. For example: #crm_server#/StaticContent/*<Gadget HTML File Name>*
4. Click **Links** on the Web Site gadget to link it 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**. As a result, the Record Summary is populated with the company data.

scrmGetSageCRMOwner method

Finds the web site gadget that owns the current page.

Parameters

- **iframeDomElementId**. ID of the iframe element that's stored by the URL gadget.

Example

```
gadget = parent.scrmGetSageCrmOwner(window.frameElement.id);
```

scrmRegisterEvent method

Registers event classes the gadget posts and/or listens to.

Parameters

- **gadget**. Gadget that publishes/receives event; Must not be null. May be obtained by calling **scrmGetSageCRMOwner method** method.
- **entityId**. ID (custom_tables.bord_tableid) of entity that gadget publishes or 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**. Either PUBLISH (when gadget publishes information), LISTEN (when gadget responds for events in other gadgets), or BOTH. Must not be null or empty.

Example

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

scrmPublishEvent method

Publishes event.

Parameters

- **gadget**. Gadget that calls the method.
- **fieldName**. Field name that's published.
- **jsonData**. Data to publish, the first property must be entityRecordId and must contain the value of the fieldName field.

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);
```

scrmGetGadgetProperty method

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 method

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 method

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);
```

Code sample: Custom gadget

This example declares methods used to interact with the Interactive Dashboard eventing mechanism. This page can publish and listen to the Company entity.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Untitled Page</title>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        <!First, the page needs to declare methods used to interact with
        Interactive Dashboard eventing mehanism. !>
        <!In this example, the page declares that it may publish as well as listen to
        the Company entity. !>
        <script type="text/javascript">
            //This global variable stores the Web Site Gadget that displays the page
            //in an iframe; It will be used in two places:
            //1)when page declares that it wishes to use Interactive Dashboard and
            //2)when page publishes information to other gadgets;
            gadget = null;
            try{
                //Find the Web Site Gadget that owns the current page:
                //API description:
                //public static IFrameGadget scrmGetSageCrmOwner(String iframeDomElementId)
                //@param iframeDomElementId Id of the iframe element that is stored by the url
gadget;
                gadget = parent.scrmGetSageCrmOwner(window.frameElement.id);
                //Now that the gadget is located, the page needs to register event classes
                //it will post and/or listen to.
                //API description:
                //public static void scrmRegisterEvent(IFrameGadget gadget, String fieldName,
                //String entityId, String fieldType, String direction)
                //@param gadget Gadget that publishes/receives event; Must not be null.
                //May be obtained by calling scrmGetSageCrmOwner method;
                //@param entityId ID (custom_tables.bord_tableid) of entity that gadget
                //publishes/may listen to. May be null;
                //@param fieldName Name of field the gadget publishes/may listen to.
                //Must not be null or empty;
                //@param fieldType Type of field the gadget publishes/may listen to. May be null;
                //@param direction one of: "PUBLISH" (when gadget publishes information),
                //"LISTEN" (when gadget responds for events in other gadgets),
                //"BOTH"; Must not be null or empty;
                parent.scrmRegisterEvent(gadget, "Test Field", "5", null, "BOTH");
            }
            catch(err){
                //errors that comes from Interactive Dashboard API are written to
                //log file on the server automatically,
                //and are rethrown again up the stack - so developer can react for errors as well
                alert(err.description);
            }
       </script>
        <!As the page publishes information, it is handy to write one method that
        does it and call it later in from within body!>
        <script type="text/javascript">
            //This method builds message to be published based on form input,
            //and eventually - publishes the message;
            //Notice, that native gadgets require that the published data is in JSON format. The
first
            //field in JSON data must be called "entityRecordId" and its value must be row
identifier.
```

```
            function publishInteractiveDashboardEvent(){
                try{
                    message = '{"entityRecordId":"' + document.forms[0].elements
["companyId"].value + '"}';
                    //API description:
                    //public static void scrmPublishEvent(IFrameGadget gadget,
                    //String fieldName, String jsonData)
                    //@param gadget Gadget that calls the method
                    //@param fieldName field name that is being published,
                    //the "entityRecordId" value from jsonData is value of this field
                    //@param jsonData data to publish, the first property must be called
                    //"entityRecordId" and must contain value of field called fieldName
                    parent.scrmPublishEvent(gadget, "Test Field", message);
                }
                catch(err){
                //errors that comes from Interactive Dashboard API are written to log file on
                //the server automatically,
                //and are rethrown again up the stack - so developer can react for errors as well
                    alert(err.description);
                }
            }
        </script>
         <!To receive an event from othe gadget, event handler must be declared.!>
         <script type="text/javascript">
            //To receive an event from othe gadget, the following method must be declared.
            //This method is called whenever gadget to which current page is subscribed to
            //publishes an event.
            //NOTE: the gadget may publish the same event more than one time, so it is good
practice to
            //react to the first event only and ignore the others.
            //API description:
            //function onSageCrmEvent(publisherGadgetId, publisherFieldName, publisherFieldType,
            //publisherEntityId, publishedData)
            //publisherGadgetId Id of publisher gadget
            //publisherFieldName Name of field in publisher gadget; Always filled;
            //publisherFieldType Type of field in publisher gadget; May be null or empty;
            //publisherEntityId ID of entity that is being published; May be null or empty;
            //publishedData The data publisher gadget sends; typically JSON string where the
first
            //value is called "entityRecordId" and contains ID of record;
            lastEntityRecordId = null;
            function onSageCrmEvent(publisherGadgetId, publisherFieldName, publisherFieldType,
            publisherEntityId, publishedData){
                //first, convert the publishedData in JSON format to object
                var publishedObject = eval('(' + publishedData + ')');
                //now check if this event hasn't been already processed
                entityRecordId = publishedObject.entityRecordId;
                if(entityRecordId!=lastEntityRecordId){
                    //mark this event as processed
                    lastEntityRecordId = entityRecordId;
                    //do the processing
                    logIncomingEvent(publishedObject, publishedData);
                }
            }
        </script>
        <!To read or write custom propertyies from/to the server, call simple method from API>
        <script type="text/javascript">
            //This method reads property from that may have been stored on the server
            function readCustomProperty(){
                try{
                    //API description:
                    //public static void scrmGetGadgetProperty(IFrameGadget gadget, String
```

```
propertyName)
                    //@param gadget Gadget that calls the method
                    //@param propertyName name of the property to read
                    propValue = parent.scrmGetGadgetProperty(gadget,
                    document.forms[0].elements["propertyName"].value);
            //Property may be null, if was not yet written
            if(propValue==null){
              propValue = '';
            }
            document.forms[0].elements["propertyValue"].value = propValue;
                }
                catch(err){
                //errors that comes from Interactive Dashboard API are written to log file on
                //the server automatically,
                //and are rethrown again up the stack - so developer can react for errors as well
                    alert(err.description);
                }
            }
            //This method saves property on the server so it can be restored in the future
            function saveCustomProperty(){
                try{
             //First, you set ALL properties. In our case it is only one property,
                    //but you can set as many as you like.
                    //API description:
                    //public static void scrmSetGadgetProperty(IFrameGadget gadget, String
propertyName)
                    //@param gadget Gadget that calls the method
                    //@param propertyName name of the property to read
                    parent.scrmSetGadgetProperty(gadget, document.forms[0].elements
["propertyName"].
                    value, document.forms[0].elements["propertyValue"].value);
            //Now, when all properties are set,
                   //save them on the server in one http request;
            parent.scrmSaveGadgetProperties(gadget);

            alert('Property saved succesfully');
                }
                catch(err){
                //errors that comes from Interactive Dashboard API are written to log file on the
                //server automatically,
                //and are rethrown again up the stack - so developer can react for errors as well
                    alert(err.description);
                }
            }
      </script>
        <script type="text/jscript">
            //this is simple function used in example to log incoming events;
            function logIncomingEvent(publishedObject, publishedData){
                var table=document.getElementById('eventsLog');
                var row = table.insertRow(2);
                var cellRecordId=row.insertCell(0);
                var cellRawData=row.insertCell(1);
                cellRecordId.innerHTML="'" + publishedObject.entityRecordId + "'";
                cellRawData.innerHTML="'" + publishedData + "'";
            }
       </script>
</head>
<body>
    <form id="form1" runat="server">
    <div>
      Custom properties allow to store data between sessions on CRM server.<br/>
      Custom property test: <br/>
```

```
      <table>
      <tr> <td> Name:</td><td><input
type="text" name="propertyName" value="testProperty"/> </td></tr>
      <tr> <td> Value: </td><td> <input type="text" name="propertyValue" value=""/> </td> </tr>
      </table><br/>
      <input type="button" value="Save!" onclick="saveCustomProperty();"/><input type="button"
         value="Read!" onclick="readCustomProperty();"/><br/>
      <hr/>
      Gadgets may publish events top other gadgets:<br/>
            Company ID: <input type="text" name="companyId" value="18"/>
            <input type="button" value="Publish!" onclick="publishInteractiveDashboardEvent
();"/> <br/>
      <hr/>
      Gadgets may also receive events from other gadgets:<br/>
            <table id='eventsLog' border='1' width="100%" style="text-align:center">
                <tr>
                    <td colspan="2" align="center">
                        Incoming events:
                    </td>
                </tr>
                <tr>
                    <td align="center">
                        Record Id
                    </td>
                    <td align="center">
                        Raw data
                    </td>
                </tr>
                </table>

    </div>
    </form>
</body>
</html>
```

System menus

- **Modifying system menus**
- **Creating a new menu button**
- **Adding an external link to the main menu**

Modifying system menus

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

- Administration menu
- Main menu

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

For more information on System Menus, see the *System Administrator Help*.

To customize tab groups for system menus:

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | System Menus**.
3. In the **Tab Group Name** column, click the tab group that you want to modify.
4. Use the page that opens to add, remove, or update tabs as necessary.
5. When you are finished, click **Save**.

To customize tab groups for a Primary or Secondary entity:

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization**.
3. Click the Primary or Secondary entity.
4. Click the **Tabs** tab.
5. In the **Tab Group Name** column, click the tab group you want to modify.
6. Use the page that opens to add, remove, or update tabs as necessary.
7. When you are finished, click **Save**.

Creating a new menu button

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

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | System Menus**.
3. In the **Tab Group Name** column, click the tab group you want to modify:
    - To create a main menu button, click **MainMenu**.
    - To create an admin menu button, click an admin link. **Admin** is the Administration main menu and home page. **AdminUsers** is the Users home page.

4. On the page that opens, use the following options:
    - **Caption**. Enter the caption you want to assign to the new menu button.
    - **Action**. Select **customurl**, and then in the **Url name** text box type the URL of the page you want the new menu button to open. If the target page is a custom ASP page, select **customfile**, and then type the ASP file name in the **Custom File** text box.
    - **Bitmap**. Select the file that contains the icon you want to appear on the new menu button.
    - **New Window**. Select **Yes** if you want the new menu button to open the page in a new window. If you want to open the page in the current window, select **No** in this option.
      This option is only available when in **Action** you selected **customurl**.
5. Click **Add** and then click **Save** to create the new menu button.
   The new menu button is displayed on the top of the screen.

Adding an external link to the main menu

You can add a new **Home** button to the Menu and configure that button to open a web page of your choice.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | System | System Behavior**.
3. Click **Change**.
4. In **Home page URL**, enter the URL of the web page you want the **Home** button to open.
   For example, you can configure the **Home** button to open the web page of your company.
5. Click **Save**.
6. Log off and then log back on.
   The new **Home** button appears on the right-hand side of the Menu at the top of the Sage CRM page.

Tabs

- **Creating a new tab group**
- **Editing the main menu tab group**
- **Adding a tab that links to an ASP page**
- **Restricting access to a tab**

- **Tab properties**
- **Tab actions**

Creating a new tab group

When you link to a new table in the Sage 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. To display the new tab group, link it to a main menu button.

These steps create a new tab group from a newly connected table.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Tabs**.
3. Click **New**, enter a tab name in **Name**, and then click **Save**.
   The new tab group is displayed in the list of tab groups.
4. Click the tab group name.
5. Add new tabs to the tab group.
6. Click **Update** and then click **Save**.

To view the tab group, create a new main menu button and link it to an ASP page that calls the tab group. When the user clicks the new menu button, the tabs in the new tab group are displayed.

Editing the main menu tab group

You can edit tab groups in the Main Menu, My CRM menu, and Team CRM menu.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | System Menus**.
3. Click the tab group you want to edit.
4. Edit the tab group as necessary.
5. Click **Save**.

For more information on customizing tabs, see the *System Administrator Help*.

Adding a tab that links to an ASP page

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <entity> | Tabs**.
3. Click the tab group to which you want to add the new tab.
4. Use the following options:
    - **Caption**. Enter a name for the new tab.
    - **Action**. Select **customfile**.
    - **Custom File**. Enter the file name of the ASP page.
5. Click **Add** and then click **Save**.

The sample ASP page below sets the tab to display the invoice list for the current company. The ASP page calls a previously created Invoice List. For more information on creating lists, see **Creating a list**.

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

var ThisCompany;
var Invoices;

// Get the current company ID.
ThisCompany = CRM.GetContextInfo("Company","Comp_CompanyID");

// Call the list block that you previously created for invoices.
Invoices = CRM.GetBlock("Invoice_List");
Invoices.Title = "3rd Party Invoice History";

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

%>
```

To view the new tab, find a relevant entity record and click the new tab. For example, find a company and click the **Invoices** tab.

Restricting access to a tab

You can use an SQL statement to make a tab accessible exclusively to specific users, teams, or territories. Other users, teams, and territories won't have access to that tab. For example, if you're adding an **Invoices** tab to the company tab group, you can restrict tab access to users in the Direct Sales team.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Tabs**.
3. Click the tab group that contains the tab to which you want to restrict access.
4. In **SQL**, enter an SQL statement to restrict access to the tab, click **Add**, and then click **Save**.

If you want to restrict access to several tabs, avoid using the syntax `user_userid=<User ID>`, because the database is queried separately for each restricted tab.

Rather, use one of the following:

Syntax	Description
`U:<User IDs>`	Makes the tab accessible to the specified users only. **Example** `U:4,5` This example makes the tab accessible to the users whose IDs are 4 and 5.
`C:<Channel IDs>`	Makes the tab accessible to the specified teams only. **Example** `C:4,5` This example makes the tab accessible to the teams whose channel IDs are 4 and 5.
`T:<Territory IDs>`	Makes the tab accessible to the specified territories only. **Example** `T:-2147483640` This example makes the tab accessible to the territory whose ID is -2147483640.

Tab properties

Field	Description
Caption	Enter the tab name you want to appear on the Sage CRM user interface.
Action	Select a tab action to display various Sage CRM screens. For more information, see **Tab actions**.
Custom File	Enter the name of the custom ASP file you want to use. This field only shows up when in **Action** you select **customfile**. The ASP file you specify must be stored in the following folder: <Sage CRM Installation Folder>\<Install Name>**WWWRoot\CustomPages** If the target ASP file is stored in the root of that folder, you only need to enter the file name, for example, *MyCustomFile.asp*. If the target ASP file is stored in a subfolder, enter the subfolder name followed by the file name, for example, *MyAspFiles\MyCustomFile.asp*.
Url Name	Enter the URL you want to open. This field only shows up when in **Action** you select **curomurl**.
Block Name	Enter the name of the block you want to run. This field only shows up when in **Action** you select **runblock**.
Tab Group Name	Enter the tab group name. This field only shows up when in **Action** you select **runtabgroup**.
System Act	Select the system action you want to perform. This field only shows up when in **Action** you select **other**.
SQL	Enter an SQL statement to restrict access to the tab to specific users, teams, or territories. For more information, see **Restricting access to a tab**.
Bitmap	Specify the graphic file you want to use. For example, when you're creating a menu button linked to a custom page. To include your custom graphic file in the list, copy it to the the folder that stores graphic files for the current theme in Sage CRM. When the current theme is **Contemporary**, graphic files are stored in the

Field	Description
	following default folder: **%ProgramFiles (x86)%\CRM\CRM\WWWRoot\Themes\Img\Ergonomic\Icons**
New Window (only available when **Action** is set to **customurl**.	Select if you want to open the custom URL in a new or current window. • **Yes**. Opens the screen in a new window. • **No**. Opens the screen in the current window.

Tab actions

When you create a new tab, you can specify the tab action. Each action displays different Sage CRM screens including typical search screens and custom screens created using ASP pages. For information about system actions that are available if you don't have the Extensibility Module, see the *System Administrator Help*.

Action	Description
customfile	Displays custom ASP pages. For examples, see: • **Adding a tab that links to an ASP page** • **Modifying system menus**
customurl	Displays URLs. For examples, see: • **Adding an external link to the main menu** • **Modifying system menus**
runblock	Displays the following blocks: • The screen name of a screen based on the current entity • The list name of a list based on the current entity • Any EntryGroupBlock based on a screen of the current entity

Action	Description
	• Content blocks • Marquee blocks • Message blocks • Chart blocks
runtabgroup	Displays tab groups.
Other	Select a system action. For more information, see the *System Administrator Help*.

Adding Help to custom pages

You can add a help button and link a context-sensitive help topic to an ASP page. For more information on customizing help, see the *System Administrator Help*.

1. Use an HTML editor to create an HTML help file and save it to <Sage CRM Installation Folder><Install Name>\WWWRoot\HELP\EN\Main Menu\Content\User.

   For example: %ProgramFiles(x86)%\Sage\CRM\CRM\WWWRoot\HELP\EN\Main Menu\Content\User

   Here's an example of an HTML help file.

   ```
   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
   <html xmlns="http://www.w3.org/1999/xhtml">
   <head>
       <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
       <title>My Help File</title>
   </head>
   <body>
   <h2 class="pHeading1">This is the Heading for the Help Topic</h2>
   <p class="pBody">This is where the body of the help topic goes</p>
   <hr />
   <p style="font-size: 7pt" font-family-"arial">&copy; Copyright Sage Technologies. All
   rights reserved.</p>
   </body>
   </html>
   ```

2. Add a help button to your ASP page.
   The following code creates the button. In this example, the custom help file is called **FI_**

SearchingForCompany.htm.

```
//The new help file:
var HelpFile = "FI_SearchingForCompany.htm";

//This specifies the path to the help system and should be included here unchanged.
var strCustomHelpButton = CRM.Button("Help", "help.gif",
"javascript:window.open('/"+sInstallName+"/help/EN/Main Menu/Default_
CSH.htm#User/"+HelpFile+"',
'HELPWIN','scrollbars=yes,toolbar=no,menubar=no,resizable=yes,top=200,
width=600,height=400');");;::
re = /href/gi;
strCustomHelpButton = strCustomHelpButton.replace(re, "onclick");
myBlock.AddButton(strCustomHelpButton);
```

3. In Sage CRM, open the ASP page and then click the **Help** button.
   The online help opens in a new window and displays the new help page.
   Due to a limitation of this method, the new help page can't be included in the Sage CRM Help Table of Contents, Index, or Search.

We recommend that you create backup copies of your custom help pages and store them in a safe place.
Any changes you make to help files in the Sage CRM installation folder may be overwritten when you upgrade Sage CRM to a new version or install a Sage CRM patch.

Database

You can create new database connections and external table connections. A database connection is the registration of another database to which the Sage CRM server can connect directly or indirectly. You can select from predefined database types. When you've made the database connection, you can create a new table connection by referencing the new database connection or an existing database connection. You need the Extensibility Module in order to create these connections. Also, the database with which the connection is made must have a field that can uniquely identify the table, and there must be a matching field with this value on the related table in Sage CRM.

- **Creating a new database table**
- **Creating a new database connection**
- **Creating a new table connection**
- **Table- and entity-level scripts**
- **Database customization examples**
- **Getting a list of field types used in the system**

Creating a new database table

1. Go to **<My Profile> | Administration | Advanced Customization | Tables And Databases**.
2. Click **Create Table**.
3. Complete the following options and click **Save**:
    - **Table Name**. Enter a name for the new table. Make sure the name does not contain spaces.
    - **Table Caption**. Click in this text box to enter a table caption. By default, the table caption is identical to the value you entered in **Table Name**. You can edit the default table caption if necessary.
    - **ID Field Name**. Enter the name of the table field (column) with which you want to uniquely identify the table. Use the following format: *<ColumnPrefix>_<TableName>ID*. This field is required to use the table like a normal Sage CRM table with screens, lists, and so on.
    - **Column Prefix**. Enter a prefix for the columns in the table. A column prefix is usually three to four characters long. Do not include an underscore.
      Column prefix example: `newt`.
    - **Description Field**. Enter a description to use this table as a lookup. When you configure a selection entry type, the table is listed in **Existing Lookups**.
      Enter a description for tables with small amounts of rarely changing data only because the records are loaded into memory. The user must ensure that metadata is refreshed whenever changes are made to the table so that changes are reflected in the drop-down list.
      If the table contains a large (approximately 1,000+) number of records, Sage CRM may time out when loading.
    - **Company Id Field**. Enter name of the table field that holds identity values to link the new table to the Company entity. Use the following format: *<ColumnPrefix>_<FieldName>*.
    - **Person Id Field**. Enter the name of the table field that holds identity values to link the new table to the Person entity. Use the following format: *<ColumnPrefix>_<FieldName>*.
    - **User Id Field**. Enter the name of the table field that holds identity values to link the new table to the User entity. Use the following format: *<ColumnPrefix>_<FieldName>*.
    - **Workflow Id Field**. Enter the name of the table field that's used to identify workflows. Use the following format: *<ColumnPrefix>_<FieldName>*.

- **Top Level Entity**. To make the table a Primary entity, select **Yes**. Otherwise, select **No**.
- **Allow Web Service Access**. To allow Web Service access to the table, select **Yes**. Otherwise, select **No**.
- **Read-only SData**. To allow SData Provider to access the table, select **Yes**. Otherwise, select **No**. For more information, see **Using SData API**.

The following columns are automatically included in the new table. If you entered a value in **Company Id Field**, **Person Id Field**, or **User Id Field**, the corresponding ID field is created in the table.

Column	Description
Newt_NewTableId	Stores the unique ID for records.
Newt_CreatedBy	Stores the user who creates a new record.
Newt_CreatedDate	Stores the date when new records are created.
Newt_UpdatedBy	Stores the user who updates a record.
Newt_UpdatedDate	Stores the date when a record is created.
Newt_Timestamp	Stores the time when a record is created.
Newt_Deleted	Stores the date when a record is deleted.

Creating a new database connection

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | Tables And Databases**.
3. Click **New Database Connection**.
4. Complete the following options and click **Save**:
   - **Database Driver**. Select the type of the database to which you want to connect.
   - **Server Name (SQL Server Only)**. Enter the name of the computer that hosts the Microsoft SQL Server database to which you want to connect. This field is not required for other database types.
   - **Database Name**. Enter the name of the database to which you want to connect.
   - **Port Number**. Enter the port number on which you want to access the database.

- **Database Description**. Enter a friendly description with which you want to identify the database in the Sage CRM user interface.
- **User Name**. Enter the user name of the account under which you want to access the database.
- **Database Password**. Enter the password that matches the user name.

When Sage CRM connects to the database, it appears in **<My Profile> | Administration | Advanced Customization | Tables And Databases**.

Creating a new table connection

Note that you cannot create a connection to two or more external tables if all those tables contain a field with identical name.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | Tables And Databases**.
3. Click **New Table Connection**.
4. Complete the following options and click **Save**:
    - **Table Name**. Enter the name of the table to which you want to connect.
    - **Table Caption**. Enter a friendly table description you want to appear in the Sage CRM lists.
    - **Database**. Select the database that contains the table to which you want to connect.
    - **ID Field Name**. Enter the name of the table field (column) with which you want to uniquely identify the table.

To access the table, go to **<My Profile> | Administration | Customization | Secondary Entity**.Note that you cannot add views to an external table.

The table columns are displayed as fields in **<My Profile> | Administration | Customization | <external table> | Fields**. To add the fields to screen areas, such as lists and screens, go to **<My Profile> | Administration | Customization | <external table>**.

Table- and entity-level scripts

Table- and entity-level scripts are an alternative method of creating SQL triggers that can be performed in the Sage CRM system. The Extensibility Module is required to create and customize table- and entity-level scripts.

Table-level scripts are executed when a record is inserted, updated, or deleted on a specified Sage CRM table. Entity-level scripts are executed when a Sage CRM entity is inserted, updated, or deleted.

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

Compared to SQL triggers, table- and entity-level scripts provide the following benefits:

- **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 handle error messages. If an unhandled script error occurs, the system displays the script name, line number, and the error message.
- **Smoother upgrade process**. Before performing an upgrade, you have to disable all SQL triggers and then re-enable them afterwards. This is not the case with table- and entity-level scripts.

In this section:

Table-level scripts

Table-level scripts are executed when a record is inserted, updated, or deleted on a specified table. Table-level scripts enable you to reference CRM objects and 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.

Detached table-level scripts

Detached table-level scripts run within a predefined amount of time after a record is inserted, updated, or deleted on a specified table. This enables the system to store a queue of scripts, so users don't have 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. Users don't see errors as they happen; they must view the log file for any diagnostic errors.

Entity-level scripts

Entity-level scripts and entity-level with rollback scripts are executed when an entity is inserted, updated, or deleted.

- Use entity-level scripts when an 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.
- Use entity-level with rollback scripts when you want to stop an action if a script error occurs.

You can use entity-level scripts on the following entities: Company, Person, Email, and Address. The scripts are invoked from the following standard Sage CRM screens when you click the final **Save**.

Screen	EntityScript	Method
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()

Screen	EntityScript	Method
New Address	Address	InsertRecord()
Edit Address	Address	UpdateRecord()
Delete Address	Address	DeleteRecord()

For example, with an InsertRecord entity-level script attached to Company, each time you create a new Company, the `InsertRecord()` function is executed when all the normal Company updates are complete but before the final commit. Normal Company updates include inserts into many tables. For example, address, address_link, phone, email, person, person_link.

If an error occurs while executing an entity-level with rollback script, all changes are undone and the Company is not inserted. The error is displayed on the insert screen.

Creating a script

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | TableScripts**.
3. Click **New**.
4. Complete the following options and click **Save**:
    - **Name**. Enter a descriptive name with which you want to identify the script in the Sage CRM user interface.
    - **Windows User as Domain\User (Optional)**. Enter the Windows user account under which you want to run the script. Use the format *domain**user name*. If you leave this option blank, the script is run under the current user account.
    - **User Password**. Enter the password that matches the user account specified in **Windows User as Domain\User (Optional)**.
    - **Script Type**. Select the type of your script.
    - **View**. Only use this option when creating entity-level scripts. Enter the view from which fields are available in the script. This must be relevant to the current entity. If you leave this option blank, the default view for that entity is used.
    - **Order**. Specify the order of scripts if there's more than one script for a table.
    - **Logging Level**. Select the logging level to use when a user clicks **Show Log**:
        - **Off**. Disables logging.
        - **Low**. Writes low-level diagnostic information in the log table.

    - **Medium**. Writes medium-level diagnostic information in the log table.
    - **High**. Writes high-level diagnostic information in the log table.
- **On error, only display the default error message**. Select this check box to hide error details from users. You can enter an alternative error message in **Default Error Message**. If you select this check box but don't enter a default error message, no errors are shown to the user.
- **On error, retry the script after delay**. Select this check box to re-run the script after a defined amount of time when an error occurs. The script is automatically re-run until it completes without errors. This is useful for situations where an external resource is temporarily available. Every time the script is re-run, the amount of time until Sage CRM retries the script is increased.
- **Table level script**. Use this text box to enter your table- or entity-level script. Depending on the type of script you want to run, enter your script in the appropriate section. That is, `function InsertRecord()`, `function PostInsertRecord()`, `function UpdateRecord()`, or `function DeleteRecord()`.
- **Disabled**. Select this check box to disable the script. To enable the script, clear this check box.

Viewing script logs

You can view diagnostic information for a script that was run at least once. For example, this can be useful if you run **Detached table-level scripts** and want to check if the scripts completed successfully. Script logs can help you to determine the reason why the script failed.

To view script logs:

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | TableScripts**.
3. In the list, click the script name.
4. Click **Show Log**.
   Diagnostic information is displayed according to the logging level you set. You can change this each time the script is run. If you want to delete the diagnostic information, click **Clear Log**.

Disabling table-level scripts

When you create a table-level script for an entity in Sage CRM, the script is automatically enabled by default. If necessary, you can selectively disable table-level scripts for a particular entity.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | TableScripts**.
3. In the **Script Name** column, click the name of the script you want to disable.
4. Select the **Disabled** check box and click **Save**.

You can enable a disabled script at any time by clearing the **Disabled** check box.

Database customization examples

- **Creating a tab to display a list of invoices**
- **Displaying an invoice from a list**
- **Adding new data entry and maintenance screens**
- **Using UpdateRecord in an entity-level script**
- **Using InsertRecord in a table-level script**
- **Using PostInsertRecord in a table-level script**
- **Using UpdateRecord in a table-level script**
- **Using DeleteRecord in a table-level script**

Creating a tab to display a list of invoices

This example connects Sage CRM to a table called Invoices on a third-party database called External, and displays data from the table through Sage CRM. The Invoices table contains a field called `Customerid` that can uniquely identify companies within Sage CRM. The company table in Sage CRM has a corresponding field with this value.

You can work through the example using your own third-party database, or create a database using a tool such as SQL Enterprise Manager.

The External database and Invoices table are not supplied with the sample data in Sage CRM.

To create a tab that displays a list of invoices, complete the following steps:

- **Step 1: Connect to the External database**
- **Step 2: Connect the Invoices table**
- **Step 3: Create a List object for Invoices**

- **Step 4: Create an ASP page to display the Invoices list**
- **Step 5: Add a tab that opens the ASP page**

Step 1: Connect to the External database

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | Tables And Databases**.
3. Click **New Database Connection**.
4. Complete the database details fields. For more information, see **Creating a new database connection**.
5. Click **Save** to create the connection to the External database.

Step 2: Connect the Invoices table

1. Go to **<My Profile> | Administration | Advanced Customization | Tables And Databases**.
2. Click **New Table Connection**.
3. Complete the table details fields. For more information, see **Creating a new table connection**.
4. Click **Save** to connect to the Invoices table.
   The table is listed in **Administration | Customization | Secondary Entities**.

Step 3: Create a List object for Invoices

1. Go to **<My Profile> | Administration | Customization | Invoices | Lists**.
2. Click **New**.
3. Enter a list name and select the table on which it's based.
   You'll need to reference this list name from the ASP page.
4. Click **Save**.
5. Click the new list name and add the table columns you want to display in the list.
6. Click **Update** and then click **Save**.

Step 4: Create an ASP page to display the Invoices list

To display the Invoice list for the current company, you need to create a custom ASP page.

1. Create a new ASP page and save it as Invoices.asp.
2. In the main block of the ASP page, include statements to perform the following tasks:
   - Retrieve the current company ID and store it in a variable:

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

   - Create a block for the Invoice list and store it in a variable. You must reference the list created in **Step 3: Create a List object for Invoices**.

     ```
     Invoices = CRM.GetBlock("Invoice_List");
     ```

   - Display the list on the screen by executing the List block. Make sure you include a statement to show records for this company only:

     ```
     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-requisite: 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: Add a tab that opens the ASP page

1. Go to **<My Profile> | Administration | Customization | Company | Tabs**.
2. Click the **Company** tab group.

3. Use the following options:
    - **Caption**. Enter *Invoices*.
    - **Action**. Select **customfile**.
    - **Custom File**. Enter *Invoices.asp*.
4. Click **Add** and then click **Save**.

Now, when you open a Company record in Sage CRM and click the **Invoices** tab, a list of invoices for the Company record is displayed.

Displaying an invoice from a list

You can use Custom Jump action functionality to link from a list entry to a summary screen for the selected entry. This example displays an individual invoice from a list of invoices.

To display an individual invoice, complete the following steps:

- **Step 1: Edit the List object for the invoices table**
- **Step 2: Create a Screen object for invoice details**
- **Step 3: Create a custom page to display the invoice screen**

Step 1: Edit the List object for the invoices table

In this step, you add Custom Jump actions to the Invoices list.

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | Invoices | Lists**.
3. Click **Invoice List**.
4. For each field you want linked, from **Hyperlink To**, select **Custom Jump**.
5. In **Custom File**, enter the name of the ASP page that displays the Invoices screen (Invdetail.asp) in.
6. In **Custom ID Field**, enter the name of the Invoices table field that uniquely identifies each record. For example, *InvoiceID*.
7. Click **Save**.

Step 2: Create a Screen object for invoice details

1. Go to **<My Profile> | Administration | Customization | Invoices | Screens**.
2. Click **New** and select the fields to display on the invoice summary screen.

3. Click **Save**.

Step 3: Create a custom page to display the invoice screen

Use the script below to create a custom page to retrieve the screen you created for individual invoices.
The name of this page must be the same as the entry in **Custom File** in **Step 1: Edit the List object for the invoices table** , in this case, Invdetail.asp.

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

<%

var ThisInvoice;
var InvoiceDetailBlock;
var Record;
var Container;

// 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());

%>
```

To see the results, open a Company record in Sage CRM, and click the **Invoices** tab. A list of invoices for the Company record is displayed. The fields that are configured as links are formatted as hypertext.

To open the summary screen for the invoice, click an invoice number.

To return to the list, click **Continue**.

Adding new data entry and maintenance screens

This example illustrates how to add a new table to store customer-specific information in Sage CRM. You can edit or delete information in the table as required.

For example, you can use the new table to store company information needed after an opportunity has been closed, and before engineers start the implementation.

- **Step 1: Create a new table**
- **Step 2: Add the installed base tab**
- **Step 3: Create a List object for installed base**
- **Step 4: Create the installed base screen**
- **Step 5: Create one or multiple ASP pages to display records**
- **Step 6: Configure reporting on installed base records**

The steps in this example are applicable to Company, Person, Case, Lead, and Opportunity entities.

Step 1: Create a new table

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | Tables And Databases**.
3. Click **Create Table** to create a new database table called *InstallBase*.
4. Use the following options:
    - **ID Field Name**. Enter *inst_installbaseid*.
    - **Column Prefix**. Enter *inst*.
    - **Company ID Field**. Enter *inst_companyId*.
5. Click **Save**.

Step 2: Add the installed base tab

1. Go to **<My Profile> | Administration | Customization | Company | Tabs**.
2. Add a tab to the company tab group.
3. From **Action**, select **customfile**.
4. In **Custom File**, enter the ASP page name, *installbase.asp*.
5. Click **Update** and then click **Save**.

Step 3: Create a List object for installed base

1. Go to **<My Profile> | Administration | Customization | InstallBase | Lists**.
2. Click **New**.
3. Enter a list name and select the table on which it's based and click **Save**.
   Note the name of the list because you'll need to reference this name from the ASP page.
4. Click the list name and add the table columns to be displayed in the list.
5. To enable links between a list entry and the summary screen for the entry, click **inst_ companyid**.
6. Use the following options:
   - **Hyperlink To Field**. Select **Custom Jump**.
   - **Custom File**. Enter the name of the ASP file that displays an individual item.
   - **Custom ID Field**. Enter the name of the field in the new table that uniquely identifies each record.
7. Click **Update** and then click **Save**.

Step 4: Create the installed base screen

1. Go to **<My Profile> | Administration | Customization | InstallBase | Screens**.
2. Click **New**.
3. Customize the new screen to add the fields that you want to display.
   For more information about screen customization, see the *System Administrator Help*.

Step 5: Create one or multiple ASP pages to display records

Depending on how many records you want to display, click the corresponding link below and complete the provided steps:

- **Create an ASP page to display a single record**
- **Create ASP pages to display multiple records**

Create an ASP page to display a single record

This example creates an ASP page to display a single record in the new table if it doesn't already exist.

1. Create a custom ASP page to view and edit installed base records whose name matches the name specified in the tab group for Company. You can start with a sample Entry Group ASP

page and include statements in the main block of the ASP page. For more information, see **Creating an ASP page**.

2. Retrieve the identifier value for the current Company and store it in a variable.

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

3. Create an instance of the installed base screen and assign it to a variable.

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

4. Create a record for the installed base record and specify which record to display. See if the record already exists. If it doesn't, create a new record.

```
record = CRM.FindRecord("InstallBase","companyid="+CompanyId);

if (record.eof) {
  record = CRM.CreateRecord("InstallBase");
  record("CompanyId") = CompanyId;
  InstallBase.Title = "New Install Base Details";
}
```

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

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

6. 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 the Company. Thereafter, when the tab is selected, the record is shown for editing.

Here's the installbase.asp script.

```
<!-- #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());

%>
```

Create ASP pages to display multiple records

This example creates ASP pages to display multiple records in the new table for each customer. It creates two custom pages; one to view a list of records and another to jump to individual records.

1. Create a custom page whose name matches the name specified in the tab group for Company. For a sample Entry Group ASP page, see **Create an ASP page to display a single record** .
2. Include statements in the main block of the ASP page to perform the following tasks.
   - Retrieve the identifying value for the current Company and assign it to a variable.

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

   - Create a block for the Installed Base List and assign it to a variable. The list name is the name of the list created in **Step 3: Create a List object for installed base**.

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

   - Write the list to the screen by executing the List block, telling it to show records for this company only.

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

   - 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());
     ```

3. Create another custom page to jump to view/edit/delete individual records. The page name is the name of the page specified in **Custom Action File** in **Step 3: Create a List object for installed base**.
4. Include statements in the main block of the ASP page to perform the following tasks:
   - Retrieve the ID value of the record that's viewed from the Query string.

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

   - Create a block for the Installed Base screen and assign it to a variable.

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

   - Turn on the **Delete** button on the block to allow existing records to be deleted.

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

   - Turn on the **Continue** button to allow users to return to the list.

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

   - Check if this is an existing record or if a new record must be created. Create the Record object if required. If it's a new record, the exact Company ID must be set 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);
         }
     ```

5. Display the block, passing in the Record object. Note that the edit, delete, and add functionality is handled by the block internally.

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 the 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());

%>
```

Step 6: Configure reporting on installed base records

1. Do one of the following:
    - To create a view for the external table, go to **<My Profile> | Administration | Customization | Installed Base | Views**.
    - To create a new report category for Installed Base, go to **Main Menu | Reports | New Report Category**.
2. Create a report in the new category based on the Installed Base view.

To see the results, open a Company, and click the **Installed Base** tab. A list of installed base records for the Company is displayed. The fields that are configured as links are formatted as hypertext.

- To edit or delete the summary screen for the record, click **Record**.
- To add new records from the list screen, click **New**.

The Company ID on the Installed Base table is an integer value. To display the Company name (rather than the ID) in a report, set the **Entry Type** of the company ID field to **Search Select**, using the Company entity.

For more information about creating views, report categories, and reports, see the *System Administrator Help*.

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

Using UpdateRecord in an entity-level script

This example uses the UpdateRecord function in an entity-level script. The script is triggered when Company is updated. When company type changes from Prospect to Customer, a record is created in an external table called Invoices on a third-party database called External.

The External database and the Invoices table are not supplied with the sample data in Sage CRM.

1. Create a new script. For instructions, see **Creating a script**.
   When creating your script, do the following:
   - From **Script Type**, select **Entity Level**.
   - In **Table level script**, within the `function UpdateRecord` section, enter the following script:

```
function UpdateRecord()
{
    var sType = Comp_Type;
    var sOldType = _HIDDENComp_Type;
    if ((sType != null) && (sType.toLowerCase() == 'customer') && (sOldType != null) &&
(sOldType.toLowerCase() == 'prospect'))
    {
    // company type changed from Prospect to Customer
    // so create record in Invoices table
    // must work out what the next id is for the invoice table
    sql = 'DECLARE @returnkey integer '+ 'SELECT @returnkey = (select count(*) from
Invoices)';
    sql+= 'SELECT @returnkey as retkey';
    q = CRM.CreateQueryObj(sql,'external');
    q.SelectSql();
    NextInvoice = Number(q.FieldValue('retkey')) + 1;
    NewInvoice = CRM.CreateRecord('Invoices');
    NewInvoice.InvoiceId = NextInvoice;
    NewInvoice.customerid = CRM.GetContextInfo('Company', 'Comp_CompanyId');
    NewInvoice.description='New Customer Invoice';
    now = new Date();
    NewInvoice.InvoiceDate = (now.getMonth()+1) + '/'+ now.getDate() + '/'+ now.getYear
();
    NewInvoice.VAT = Number('120.00');
    NewInvoice.Amount = Number('800.00');
    NewInvoice.Currency = 'EUR';
    NewInvoice.SaveChanges();
    }
}
```

2. Click **Save**.

Using InsertRecord in a table-level script

This example uses the `InsertRecord` function in a table-level script. The script assigns new cases to the account manager of the selected company. You must disable Workflow for Cases before trying this example.

1. Create a new script. For instructions, see **Creating a script**.
   When creating your script, do the following:
   - From **Script Type**, select **Table Level**.
   - In **Table level script**, within the `function InsertRecord` section, enter the following script:

```
function InsertRecord()
{

// when case is created this sets the assigned user to be the account // manager of the
company selected
iPrimaryUserID = CRM.getContextInfo('company','comp_primaryuserid');

if (iPrimaryUserID > 0) {
    Values('case_assigneduserid') = iPrimaryUserID;
    }

}
```

2. Click **Save**.

Now when you create a new case for a Company but don't assign the case to a user, the case is automatically assigned to the company account manager when you save it.

Using PostInsertRecord in a table-level script

You can include the record ID generated by the `InsertRecord` function in the `PostInsertRecord` function of the same script. This example uses 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.

Use `PostInsertRecord` to insert or update other records using the ID of the record that has just been inserted. You can't update the current record in the `PostInsertRecord` function as it's already been saved. You can read values from the Values collection, however any changes to the Values collection won't take effect.

1. Open your table-level script.
   For more information on how to create a table-level script, see **Creating a script**.

2. Enter the following script in **Table Script** within the `function PostInsertRecord` section.

```
function PostInsertRecord()
{
    intid = CRM.getContextInfo('company','comp_companyid');
    var d=new Date();
    var CmLiRec,CommRec;
    var CompRec=CRM.FindRecord("Company","Comp_CompanyId="+intid);
    if (CompRec.Comp_PrimaryUserId +""!="undefined")
    {
    CommRec=CRM.CreateRecord("Communication");
    CommRec.Comm_Action='PhoneOut';
    CommRec.Comm_Type='Task';
    CommRec.Comm_Status='Pending';
    CommRec.Comm_note="Make follow up call to client to find out
    details of case and assure action";
    CommRec.Comm_Priority='Normal';
    CommRec.SaveChanges();
    CmLiRec=CRM.CreateRecord("Comm_Link");
    CmLiRec.CmLi_Comm_CommunicationId=CommRec.Comm_CommunicationId;
    CmLiRec.Cmli_Comm_CompanyId=intid;
    CmLiRec.Cmli_Comm_NotifyTime=d.getVarDate();
    CmLiRec.Cmli_Comm_UserId=CompRec.Comp_PrimaryUserId;
    CmLiRec.SaveChanges();
    }
}
```

3. Click **Save**.

Create a new case and note the user to whom it's assigned. Log on as that user and view the created task in the calendar.

Using UpdateRecord in a table-level script

This example uses the `UpdateRecord()` function in a table-level script. The script sets all opportunities associated with a company to **Stage Sale Closed**, when the user changes the status to **Archive**.

1. Go to **<My Profile> | Administration | Customization | Company | TableScripts**.
2. Click **New**, and then create a new table-level script.
   For more information about the options you need to complete, see **Creating a script**.
   When creating your script, do the following:
   - From **Script Type**, select **Table Level**.
   - In **Table Script**, within the `function UpdateRecord` section, enter the following script:

```
function UpdateRecord()
```

```
{
    // if company Status is changed to Archive then set all its companies
    // Opportunities to be Sale Closed
    if (Comp_Status == 'Archive')
    {
    // sql string that will do the update for this company
    sql = "UPDATE Opportunity SET Oppo_Stage='Sale Closed'
    WHERE Oppo_PrimaryCompanyId=";
    sql+=CRM.GetContextInfo('Company','Comp_CompanyId');
    UpdateQuery = CRM.CreateQueryObj(sql);
    UpdateQuery.ExecSql();
    }
}
```

3. Click **Save**.

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**.

Using DeleteRecord in a table-level script

This example uses the `DeleteRecord()` function in a table-level script. The script checks whether there are any outstanding leads associated with a person when a Person record is deleted. If there are any outstanding leads, a message is displayed.

1. Go to **<My Profile> | Administration | Customization | Person | TableScripts**.
2. Click **New**, and then create a new table-level script.
   For more information about the options you need to complete, see **Creating a script**.
   When creating your script, do the following:
   - From Script Type, select Table Level.
   - In **Table Script**, within the `function DeleteRecord` section, enter the following script:

```
function DeleteRecord()
{
    var ThisPersonId = CRM.GetContextInfo('person','pers_personid');
    var sCriteria = "lead_primarypersonid="+ThisPersonId+" and lead_deleted is null and
lead_status <> 'Opportunity'";
    var LeadRecord = CRM.FindRecord("Lead",sCriteria);
    if(!LeadRecord.eof)
    {
        CheckLocks=false; ErrorStr='There are outstanding leads';
        Valid = false;
    }
}
```

3. Click **Save**.

Open a Person record that has an associated Lead and then delete the Person record. The warning message is displayed.

Getting a list of field types used in the system

You can run the following SQL query against the Sage CRM database to get a list of all field types used in the system:

```
SELECT DISTINCT dbo.Custom_Edits.ColP_EntryType,
CONVERT(nVARchar(32), dbo.Custom_Captions.Capt_US) as "Caption"
FROM dbo.Custom_Captions RIGHT OUTER JOIN dbo.Custom_Edits
ON dbo.Custom_Captions.Capt_Code =
CONVERT(nVARchar(32), dbo.Custom_Edits.ColP_EntryType)
WHERE (dbo.Custom_Captions.Capt_Family = N'EntryType')
```

For more information about possible field types, see **EntryType**.

Graphics

In Sage CRM, you can implement graphics by using ASP pages. Graphics can be generated dynamically within Sage CRM. A graphic can be data aware and representative of data stored at the time it's generated. You can customize a graphic to vary depending on the user that displays it. Graphics can be generated with just a few commands, or customized at length.

For more information about the methods you can use to work with graphics in Sage CRM, see **CRMGraphicBlock methods**.

- **Considerations for using graphics**
- **Supported graphic file formats**
- **Using external images**
- **Changing image color**
- **Clearing an image**
- **Applying dithering, zooming, or transparency**
- **Setting color, line width, and style**
- **Filling solid shapes with color**
- **Changing current font**
- **Using animation**
- **Suppressing errors when processing an image**
- **Code samples**

Considerations for using graphics

- **Avoid large images and use GIF images where possible**.Operating system and hardware may have limitations when handling graphic files.
  To convert images to .gif files, you can use the **SaveAsGifs** method exposed by the **CRMGraphicBlock object**. For example:

```
<script language=javascript>

if (screen.colorDepth<=8)
{
    graphic.SaveAsGifs=true
}

</script>
```

- **Be conservative with animations**. Large animations can consume a lot of processor time on a client machine. Consider decompression of a 50-frame animated .gif file that is 250 x 250 pixels using 24-bit color. In this case, all 50 frames require roughly 9 MB of data if stored in memory simultaneously (250 * 250 * (24/8) * 50).

  If the .gif file is animated over three seconds, that's 3 MB per second of data, which requires a lot of processor power.
  When an animated .gif file is looping, the image is continually decoded from a copy of the animation stored on disk. Therefore, processor power is used instead of consuming large amounts of memory.

Supported graphic file formats

In Sage CRM, you can use the Graphic block to work with the following graphic file formats:

- **JPEG**. Default format for graphics and charts. 16 million colors (24 bit).
  High level of compression (small file size). Does not support animation and transparency.
- **GIF**. 256 colors (8 bit). High level of compression (small file size). Supports animation and transparency.
- **BMP**. Various color depths. No compression (large file size).

The characteristics of these formats can affect the image quality of your graphic or chart. If you don't require animation and transparency, use JPEG rather than GIF, as JPEG allows your image to contain a much greater color depth.

Fusion charts are rendered as JPEG in ASP and .NET, and they are rendered as HTML5 in the Classic and Interactive Dashboard.

To save graphics as JPEG or GIF, you can use the **SaveAsGifs** and **SaveAsJPG(text)** methods exposed by the **CRMGraphicBlock object**.

For example:

```
var graphic;
graphic = CRM.GetBlock('graphic');
graphic.SaveAsGifs = true;
```

Using external images

You can save images and load them from the server. Then, you can use the **CRMGraphicBlock object** generate part of the image from the loaded images. The **CRMGraphicBlock object** provides the **SaveAsGifs** and **SaveAsJPG(text)** methods that allow you to convert any loaded image to a JPEG or GIF image, as these are the standard types supported by most web browsers.

You can merge an external image onto a graphic. A color is passed as the transparent color for the external image. You can specify the position of the image with X and Y parameters. If you don't specify the position, it appears starting from 0,0 in the top left hand corner of your graphic.

Example 1

```
Effect('Merge','c:\\Person.ico');
```

Example 2

```
Effect('Merge','c:\\Person.ico,50,50');
```

Changing image color

You can change a particular color in an image.

The following example changes all instances of blue to red:

```
Effect('ChangeColor','Blue,Red');
```

Clearing an image

You can clear an image completely and wash it with a particular color. If you don't specify a color, the canvas is cleared as white.

Example

```
Effect('Clear','Blue');
```

Applying dithering, zooming, or transparency

You can use the **Effect(Mode, Value)** method provided by the **CRMGraphicBlock object** to apply a number of special effects to an image, including dithering, zooming, and transparency.

- **Dithering**. You can apply a dithering mode to an image to help improve its appearance, especially where color is limited. The available modes are:
  - Burkes
  - FloydSteinberg
  - JaJuNi
  - Sierra
  - SteveArche
  - Stucki

  The following example shows how to apply a dithering mode to an image:

  ```
  Effect('Dither','FloydSteinberg');
  ```

- **Zooming**. You can magnify an image using the Zoom parameter and a percentage of zoom required. By default, the area to be zoomed is the center of the image.
  Example:

```
Effect('Zoom','200');
```

- **Transparency**. Available only in GIF images. You can enable transparency to make any whiteness in an image become transparent.
  The following example shows how to enable transparency:

```
Effect('Transparency','true');
```

Setting color, line width, and style

You can use the **Pen(Mode, Value)** method to set the color, line width, and style of your drawings.

Setting a color

Use the following syntax to set the color:

```
Pen('Color','<ColorName>');
```

Where `<ColorName>` is the name of the color you want to use.

Example

```
Pen('Color','Blue');
```

Alternatively, you can use the `PenColor` method:

```
PenColor('Blue');
```

Setting line width

Use the following syntax to set the line width:

```
Pen('Width','<LineWidthValue>');
```

Where `<LineWidthValue>` is the thickness of the line.

Example

```
Pen('Width','3');
```

Alternatively, you can use the **`PenWidth`** method:

```
PenWidth('3')
```

Setting line style

Use the following syntax to set the line style:

```
Pen('Style','<StyleValue>');
```

Where `<StyleValue>` can take one of the following values:

- **Solid**. Specifies to use solid line.
- **Dash**. Specifies to use dashes.
- **Dot**. Specifies to use dots.
- **DashDot**. Specifies to use alternating dashes and dots.
- **DashDotDot**. Specifies to use a series of dash-dot-dot combinations.
- **Clear**. Specifies that no line is used. Use this value to omit the line around shapes that draw an outline using the current pen.

Examples

```
Pen('Style','Dot');
```

```
Pen('Style','Solid');
```

```
Pen('Style','Clear');
```

Filling solid shapes with color

You can use the **Brush(Mode, Value)** method exposed by the **CRMGraphicBlock object** to fill solid shapes, such as rectangles and ellipses, with a color or pattern. The pattern may be a predefined image loaded by using the **Brush(Mode, Value)** method.

Specifying a color

Use the following syntax to specify the color with which you want to fill a shape:

```
Brush('Color','<ColorName>');
```

Where `<ColorName>` is the name of the color you want to use.

Example

```
Brush('Color','Blue');
```

Loading an image

Use the following syntax to load an image and use it in all painting effects:

```
Brush('Load','<PathToImage>');
```

Where `<PathToImage>` is the full path to the image file you want to use. You can specify one of the following file types: .ico, .emf/..wmf, .bmp, .gif, or .jpg.

Example

```
Brush('Load','c:\\winnt\\winnt.bmp');
```

Specifying area to fill in

Use the following syntax to specify the area you want to fill in:

```
Brush('Fill','<Left>,<Top>,<Right>,<Bottom>');
```

Where `<Left>`, `<Top>`, `<Right>`, and `<Bottom>` define the rectangle area to be filled in.

Example

```
Brush('Fill','0,0,100,100');
```

Specifying a predefined style

Use the following syntax to choose a predefined style for filling in the specified area:

```
Brush('Style','<Value>');
```

Where `<Value>` can be one of the following:

- Bdiagonal
- Clear
- Cross
- DiagCross
- Fdiagonal
- Horizontal
- Solid
- Vertical

Example

```
Brush('Style','DiagCross');
```

Changing current font

You can change the current font used by the **TextOut** and **TextOutCenter** methods of the **CRMGraphicBlock object**. For example, you can select a font to use, determine the font size, specify the font color, and apply a rotation effect to the text output.

Selecting a font

Use the following syntax to select a font:

```
Font('Name','<FontName>');
```

Where `<FontName>` is the name of the font installed on the Sage CRM server.

Example

```
Font('Name','Times New Roman');
```

Setting font size

Use the following syntax to set the font size:

```
Font('Size','<FontSize>');
```

Where `<FontSize>` is the font size you want to use.

Example

```
Font('Size','24');
```

Alternatively, you can use the **FontSize** method:

```
FontSize('24');
```

Setting font color

Use the following syntax to set the font color:

```
Font('Color','<ColorName>');
```

Where `<ColorName>` is the name of the font color you want to use.

Example

```
Font('Color','Blue');
```

Alternatively, you can use the `FontColor` method:

```
FontColor('Blue');
```

Setting font style

Use the following syntax to set the desired font style:

```
Font('<Parameter>','<Value>');
```

In this syntax, use one of the following parameters:

- **Underline**. Toggles between underlined and normal font. This parameter can take one of the following values:
  - **True**. Underlines the current font.
  - **False**. Specifies to use normal font.
- **Italic**. Toggles between italic and normal font. This parameter can take one of the following values:
  - **True**. Italicizes the current font.
  - **False**. Specifies to use normal font.
- **Strikeout**. Toggles between striked out and normal font. This parameter can take one of the following values:
  - **True**. Adds a line through the middle of the current font.
  - **False**. Specifies to use normal font.

Examples

```
Font('Underline','True');
```

```
Font('Italic','False');
```

```
Font('Strikeout','False');
```

Rotating text output

Use the following syntax to rotate your text:

```
Font('Rotate','<Angle>');
```

Where `<Angle>` is the angle by which to rotate the text.

Example

```
Font('Rotate','45');
```

Using animation

You can use the **Animation(Mode, Value)** method exposed by the **CRMGraphicBlock object** to animate your graphics. The **Animation(Mode, Value)** method has the following syntax:

```
Animation('<Parameter>','<Value>');
```

In this syntax, you can use the following parameters:

- **Add**. Adds the next frame in a series. If no frames have been added previously, adds the first frame.
- **Delay**. Specifies delay for the animation. If this parameter is left blank, the default delay is used.
- **Loops**. Loops an animation a specified number of times or indefinitely. By default, an animation is shown one time. To repeat an animation indefinitely, set this parameter to 0.

Examples

```
Animation('Add','50');
```

```
Animation('Add','');
```

```
Animation('Delay','50');
```

```
Animation('Loop','0');
```

Suppressing errors when processing an image

By default, the **CRMGraphicBlock object** shows all errors encountered when processing an image.

You can suppress errors by using the `DisplayErrors` parameter of the **Effect(Mode, Value)**method exposed by the **CRMGraphicBlock object**, as shown in the example below:

```
Effect('DisplayErrors','false');
```

Code samples

- **Steps to add a progress bar**
- **Steps to add a pipeline to show Opportunities for a Company**
- **Implementing animation**

Steps to add a progress bar

This example adds a progress bar that illustrates Opportunity Certainty for the current opportunity. You can view the complete code in **Steps to add a progress bar**.

Step 1: Create a custom ASP file

Add the following code to the file:

1. Get the current Opportunity Certainty ID and store it in a variable:

```
var progress=CRM.GetContextInfo('opportunity','oppo_certainty');
```

2. Get a graphic block and store it in a variable:

```
var progressbar=CRM.GetBlock('graphic');
```

3. Define the dimensions and style of the progress bar.
   For example:

```
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);
}
```

4. Show the progress bar on the screen by executing the graphic block:

```
CRM.AddContent(progressbar.Execute());
Response.Write(CRM.GetPage());
```

Step 2: Add a new tab and link it to your ASP file

1. In Sage CRM, go to **<My Profile> | Administration | Customization**.
2. Click **Company**, and then click **Tabs**.
3. In the **Tab Group Name** column, click **Company**.
4. Under **Properties**, use the following options:
   - **Caption**. Enter the new tab name.
   - **Action**. Select **customfile**.
   - **Custom File**. Enter the name of the custom ASP file you created in **Step 1: Create a custom ASP file**.
5. Click **Update**, and then click **Save**.

To view the results, go to an Opportunity record and click the new tab you created.

Code sample: Adding a progress bar

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

<html>
<body>
<%

var progress=CRM.GetContextInfo('opportunity','oppo_certainty');
var 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>
```

Steps to add a pipeline to show Opportunities for a Company

You can graphically represent data over a chosen cross section by using a pipeline graphic. To add a pipeline graphic, you need to use methods exposed by the **CRMPipelineGraphicBlock object**.

When a user clicks a section of the pipeline graphic, the corresponding list is filtered to show entries related to that section. You can automatically display a pipeline graphic in the Opportunities, Cases, and Leads list screens in the context of My CRM, Company, People, Opportunity, Case, and Lead.

The example below displays the forecasted value of various stages of opportunities for a Company. Within the ASP, you must pass the pipeline the user's context and a field to uniquely identify the current company.

Step 1: Create a custom ASP file

Add the following code to the file:

1. Get the ID of the current Company and store it in a variable:

```
var CompId=CRM.GetContextInfo('company','comp_companyid');
```

2. Retrieve details about the opportunities from the Sage CRM database and store them in a variable as a string:

```
var SQLPipe='select sum(Oppo_Forecast) as a,' +'Oppo_Stage from vOpportunity ' +'where
(Oppo_PrimaryCompanyid='+CompId+') ' +'group by Oppo_Stage order by Oppo_Stage';
```

3. Convert the string stored in the SQLPipe variable into an object, store that object in another variable, and then execute the stored object:

```
var Querypipe=CRM.CreateQueryObj(SQLPipe);
Querypipe.SelectSQL();
```

4. Get the pipeline block and store it in a variable.

```
var pipe=CRM.GetBlock('pipeline');
```

5. Add pipeline entries and display the pipeline on the screen.
   For example:

```
while (!Querypipe.EOF)
{
    Label=Querypipe('Oppo_Stage');
    Value=Querypipe('a');
    pipe.AddPipeEntry(Label,parseFloat(Value),Value+"");
    Querypipe.Next();
}

pipe.Selected=2;

// The summary allows the addition of any desired text in HTML format 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());
```

   For the complete ASP page code sample, see **Code sample: Adding a pipeline**.

6. Save your ASP file to the **CustomPages** folder in the Sage CRM installation directory. The default location of the **CustomPages** folder is **%ProgramFiles (x86)%\Sage\CRM\CRM\WWWRoot**

Step 2: Add a new tab and link it to your ASP file

1. In Sage CRM, go to **<My Profile> | Administration | Customization**.
2. Click **Company**, and then click **Tabs**.
3. In the **Tab Group Name** column, click **Company**.
4. Under **Properties**, use the following options:
    - **Caption**. Enter the new tab name.
    - **Action**. Select **customfile**.
    - **Custom File**. Enter the name of the custom ASP file you created in **Steps to add a pipeline to show Opportunities for a Company**.
5. Click **Update**, and then click **Save**.

Code sample: Adding a pipeline

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

<%

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');
while (!Querypipe.EOF)
{
    Label=Querypipe('Oppo_Stage');
    Value=Querypipe('a');
    pipe.AddPipeEntry(Label,parseFloat(Value),Value+"");
    Querypipe.Next();
}

// 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 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());

%>
```

Implementing animation

This sample code implements animation.

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

var progress=70;
var 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')
    }
}

var container=CRM.GetBlock('container');
with(container)
{
    AddBlock(anim);
    DisplayButton(Button_Default)=false;
}

CRM.AddContent(container.Execute());
Response.Write(CRM.GetPage());

%>
```

Workflow

Changing workflow state

> **Warning:** Always back up your database before editing workflow tables.

You can change the stage of a record in workflow by editing the WorkFlowInstance table.

When an entity record is created in a workflow, the workflow ID on the record maps to an instance ID in the WorkFlowInstance table. The WorkFlowInstance table is linked to the WorkFlowState table and determines the current state and associated rules for the record.

To change the record in the WorkFlowInstance table to another state, do the following.

1. Look in the Workflow table to find the workflow ID.
2. Run a select from the WorkFlowInstance table for the relevant entity workflow. This returns the state ID for each stage of the workflow.
3. Update the state ID of the WorkFlowInstance record to be the new state. For example, if the

instance ID is 5224 and the state ID for the new state is 62, run the following:

```
UPDATE workflowinstance SET WkIn_CurrentStateId = 62 WHERE WkIn_instanceId = 5224
```

4. Refresh the UI to display the changes.

Tip: You can change workflow state using an ASP page. For more information, see **ASP page that changes workflow state**.

ASP page that changes workflow state

The following code is from a button on an ASP page that's called from a workflow rule associated with the communication entity. The button appears on the communication screen. When the user clicks the button, the communication is closed, the stage is changed, and the workflow state for the instance is updated.

The code shows how the workflow ID and the workflow instance are used to set the workflow state.

```
var strKeyID= "key6";
var Id = new String(Request.Querystring(strKeyID));
var intRecordId = 0;
if (Id.indexOf(",") > 0)
{
    var Idarr = Id.split(",");
    intRecordId = Idarr[0];
}
    else if (Id != "")
    {
        intRecordId = Id;
    }

//Retrieve Record
var myRecord = CRM.FindRecord("communication","comm_communicationid="+intRecordId);
myRecord.comm_status = 'Complete';
myRecord.SaveChanges();

////////////////////////////////////////
var myOppoRecord = CRM.FindRecord("opportunity","oppo_opportunityid="+CRM.GetContextInfo
("opportunity","oppo_opportunityid"));
myOppoRecord.oppo_stage = 'Qualified';
myOppoRecord.SaveChanges();

////////////////////////////////////////
// Please note that the wkin_currentstateid has to be checked with
// workflow definitions
var workflowinstanceRecord = CRM.FindRecord
("workflowinstance","wkin_instanceid="+
myOppoRecord.oppo_workflowid);
workflowinstanceRecord.wkin_currentstateid = 42;
workflowinstanceRecord.SaveChanges();
```

```
////////////////////////////////////
Response.Redirect(CRM.URL(260));
```

Moving records to another workflow

There are two ways to move records from one workflow to another. Both methods preserve tracking information held in the progress tables and displayed on the tracking screen. Changes made to the data such as completing tasks, sending mail merges, or sending emails are also preserved. You manage both methods using an ASP page called with a global or transition rule.

1. **Reset the workflow**. You can create a rule attached to the entry state that lets a user reset the workflow. This is useful if you've two valid workflows for an entity and a user progresses along the wrong workflow. In this case, the user must cancel the connection to the workflow and restart at the beginning of the alternative workflow. The code in the ASP page blanks out or nulls the record's workflowid field.

```
var intRecordId = CRM.GetContextInfo("case","case_caseid");
var myRecord = CRM.FindRecord("case","case_caseid="+intRecordId);
myRecord.case_workflowid = null;
myRecord.SaveChanges();
Response.Redirect(CRM.URL(281));
```

2. **Jumping to another workflow**. You can switch from one workflow directly into another workflow and join the new workflow at a particular stage. To do this, you must create a new workflowinstance record and link it to the workflowstate information and the record that must switch to the new workflow.

```
var intRecordId = CRM.GetContextInfo("case","case_caseid");
var myRecord = CRM.FindRecord("case","case_caseid="+intRecordId);
var wkinRecord = CRM.CreateRecord("workflowinstance");
wkinRecord.wkin_workflowid = 7; // This is the workflow id of the alternate case workflow;
wkinRecord.wkin_currententityid = 3; // this is the id of the case table found in the custom_table meta data table
wkinRecord.wkin_currentrecordid = intRecordId; // The ID of the Case record being workflowed;
wkinRecord.wkin_currentstateid = 27; // This is the id value of the state found in the workflowstate table.
wkinRecord.SaveChanges();
myRecord.case_workflowid = wkinRecord.wkin_instanceid; // This sets the Case as belonging to the new workflow.
myRecord.SaveChanges();
Response.Redirect(CRM.URL(281));
```

Identifying workflow context

To identify the general workflow context in create scripts, validate scripts and table level scripts, use the following code:

```
if (CRM.GetContextInfo("company","comp_companyid"))
{
    Valid = false;
    ErrorStr = "We are in company context";
}
```

However, a workflow can have different starting points because you can define multiple primary workflow rules. A primary workflow rule is displayed as a **New** button in the UI. You can control which **New** button is displayed by defining a JavaScript condition for the rule. In this case, you must know the primary workflow context to ensure you display the correct button.

To identify the primary workflow context, check the `Key0` value in the QueryString of the URL. In a workflow primary rule you can access this with the `Values()` collection. The following example checks the workflow context and if it is the primary context of the company, the rule is not displayed.

```
if (Values("key0")==1)
{
    // Then we are in company prime context.
}
```

```
if (Values("key0")==2)
{
    // Then we are in person prime context.
}
```

Note: `Key0` is also called the Dominant Key.

Identifying workflow transitions

Several transition rules can be attached to a workflow state. You can identify the available workflow steps using the ID of the record that you're progressing through the workflow.

For example, on an opportunity record whose ID is 11, use the following SQL to return the oppo_workflowid (1167):

```
SELECT oppo_workflowid FROM opportunity WHERE oppo_opportunityid =11
```

Then use the following to retrieve the workflowinstance record:

```
SELECT wkin_workflowid, wkin_currentstateid FROM workflowinstance WHERE wkin_instanceid = 1167
```

This returns the workflowid details and current state:

```
wkin_workflowid =2
wkin_currentstateid =10
```

Which you can use to find the next possible transitions:

```
SELECT wktr_nextstateid FROM workflowtransition WHERE wktr_workflowid = 2 and wktr_stateid = 10
```

Scripting escalation rules in a component

If you create an escalation rule outside workflow and want to add it to a component, you must create the component script code manually.

Tip: To include a workflow in a component, click **<My Profile> | Administration | Advanced Customization | Workflow | <Workflow>** and click **Preview List** to generate the component manager script.

The following component manager code creates a free standing escalation rule called testescalation.

```
// Code to add the name of the workflow rule
FamilyType='Tags';
Family='WorkflowRule';
Code='TestEscalation';
Captions['US']='TestEscalation';
AddCaption();

// Code to add the rule
var jRuleId10130 = AddCustom_Data('WorkflowRules',
'WkRl', 'WkRl_RuleId', 'WkRl_Entity,WkRl_Caption,
WkRl_RuleType,WkRl_Image,WkRl_Table,
```

```
WkRl_WhereClause,WkRl_Channel,WkRl_CustomFile,
WkRl_Order,WkRl_JavaScript,WkRl_Cloneable',"
"Opportunity","TestEscalation","Time",
"WorkflowDefault.gif","Opportunity","oppo_stage =
\x27Lead\x27 and oppo_assigneduserid=#U","","",,"","',','1,2,3');

RunSql('IF NOT EXISTS (SELECT WkRl_ActionGroupId
FROM WorkflowRules WHERE
WkRl_RuleId = ' + jRuleId10130 + ' AND
WkRl_ActionGroupId IS NOT NULL) BEGIN
UPDATE WorkflowRules SET WkRl_ActionGroupId = ' + jRuleId10130 + '
WHERE WkRl_RuleId = ' + jRuleId10130 + ' END');

RunSql('DELETE FROM WorkflowActions WHERE
WkAc_ActionId IN
(SELECT AcLi_ActionId FROM WorkflowActionLinks
WHERE AcLi_ActionGroupId =
(SELECT WkRl_ActionGroupId FROM WorkflowRules
WHERE WkRl_RuleId = ' + jRuleId10130 + ' ))');
RunSql('DELETE FROM WorkflowActionLinks
WHERE AcLi_ActionGroupId = (SELECT WkRl_ActionGroupId FROM WorkflowRules
WHERE WkRl_RuleId = ' + jRuleId10130 + ' )');

var jActionId = AddCustom_Data('WorkflowActions',
'WkAc','WkAc_ActionId', 'WkAc_Action,WkAc_Field,
WkAc_Value,WkAc_Attributes,WkAc_NewLine,
WkAc_RowSpan,WkAc_ColSpan,WkAc_Table,
WkAc_EmailTo,WkAc_EmailBCC,WkAc_EmailSubject,
WkAc_EmailBody,WkAc_Order,WkAc_Condition',"
"notify","","OppoTest","","","","","Opportunity","","","","","","",','');

var LinkId = AddCustom_Data('WorkflowActionLinks',
'AcLi','AcLi_ActionLinkId','AcLi_ActionGroupId,
AcLi_ActionId',-1+','+jActionId+',','');

RunSql('UPDATE WorkflowActionLinks SET
AcLi_ActionGroupId = (SELECT WkRl_ActionGroupId FROM
WorkflowRules WHERE WkRl_RuleId = ' + jRuleId10130 + ')
WHERE AcLi_ActionLinkId = ' + LinkId);
```

Activating workflow for secondary or custom entities

You can activate workflow for secondary and custom entities that are completely or partially managed by custom application extensions (ASP pages or .NET). The screens for existing system entities do not check for workflow. The following example activates workflow for a custom entity called Project.

1. Add a proj_workflowid column to the Project table in the database.
2. Set `datatype` to integer.

3. In the custom_tables metadata table, enter a value for Project bord_workflowidfield.

```
select * from custom_tables where bord_name='project'update custom_tablesset bord_
workflowidfield = 'proj_workflowid'where bord_name='project'
```

4. Refresh the system metadata. The Project table appears in the list of available tables when you create a new workflow rule.

Using ASP pages in workflow

Primary, transition, conditional, and global workflow rules can call ASP pages. Escalation rules cannot call ASP pages. Workflow rules using ASP pages are subject to the JavaScript condition which controls whether a workflow rule is available. For more information, see *Workflow* in the **System Administrator Help**.

You can use an ASP page in workflow to do the following:

- Control the creation of a custom entity. For more information, see **Workflow properties**.
- Save a new record to a workflow. For more information, see **SetWorkflowInfo (vWorkflowName, vWorkflowState)**.
- Carry out actions on more than one record.
- Carry out actions on an external application or database.
- Carry out a complex set of actions not in the standard set. For more information, see **Workflow properties**.

For code examples, see the following:

- **ASP page that changes workflow state**
- **Code example: ShowWorkflowButtons property**
- **Code example: WorkflowTable and ShowNewWorkflowButtons properties**

Creating workflow on an external table

Workflow works on tables inside Sage CRM only. To apply workflow to an external table, you must create a shadow table in Sage CRM. The data is retrieved from the external table in the ASP pages associated with the workflow rules. The table must have a workflowid field that links the record to the workflowinstance table.

For example, you can create the following:

- A foreign key on the internal Sage CRM table with a one-to-one relationship with the external table.
- A list screen to find and display a summary page for the external table.
- A **Start Workflow** button that calls an ASP page to insert a record into the internal table and attach it to the workflow.
- In addition to ASP pages, you can use conditional rules and global rules to perform actions. The JavaScript condition can look to the external table to determine whether workflow buttons should be displayed.
- Security policies on the shadow table that apply to the external table.

Using client side code in workflow

You can use client side code in a workflow. You can add a script to a workflow action.

The following example defines an onChange rule for the **Set Column Value** workflow action that's executed on the **Opportunity Certainty** field.

```
if (this.value >10)
{
    this.value = 10;
    crm.infoMessage('Maximum value of certainty at this stage is 10');
}
```

The following example is added to the **Display Message on Screen** workflow action:

```
<script>
// window.alert('hello world');
crm.infoMessage('hello world');
</script>
```

Charts

This section provides code samples illustrating how to create charts to graphically represent information in Sage CRM. Sage CRM includes a FusionCharts component that allows you to create animated and interactive widgets providing various information. You can also create organization charts.

- **About animated and interactive charts**
- **Creating an Opportunity certainty widget**
- **Creating an Opportunities and Cases widget**
- **Creating an organization chart**

About animated and interactive charts

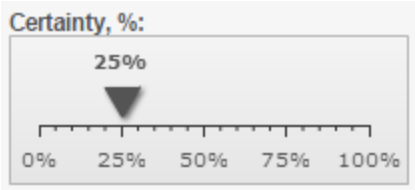
In this version of Sage CRM, animated and interactive charts are based on FusionCharts version 3.2. For more information, go to **www.fusioncharts.com**. In ASP and .NET, FusionCharts are rendered using JPG format.

To change the appearance of FusionCharts, you can use the following system parameters in the Custom_SysParams table located in the Sage CRM database:

- **ChartTimeoutSeconds**. Specifies the timeout (in seconds) before the conversion of FusionCharts to images starts. The resulting images are displayed on the Sage CRM user interface.
- **ReportsShowTextErrorInsteadOfImage**. Allows errors. Use this parameter in combination with the ChartUseFlash parameter. No data and no flash to be displayed as customized images in accordance with the file names below:
  - Theme/Images/no_adobeflash.jpg
  - Theme/Images/no_adobeflash-US.jpg
  - Theme/Images/no_data.jpg
  - Theme/Images/no_data.jpg-US.jpg
- **ChartOverruledWidthID**. Specifies overrule width for Interactive Dashboard chart width.
- **ChartOverruledHeightID**. Specifies overrule height for Interactive Dashboard chart height.
- **ChartOverruledWidthCD**. Specifies overrule width for Classic Dashboard chart width.
- **ChartOverruledHeightCD**. Specifies overrule height for Classic Dashboard chart height.

Creating an Opportunity certainty widget

This topic illustrates how to use FusionCharts to create a graphic widget that shows certainty for each Opportunity record in Sage CRM. The widget displays on the **Summary** tab for each Opportunity record and looks similar to the following:



To create this widget, do the following:

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | Opportunity**.
3. Click the **Screens** tab.
4. In the **Screen Name** column, click **OpportunityDetailBox**.
5. In the **Custom Content** text box, enter the following code, and then click **Save**.

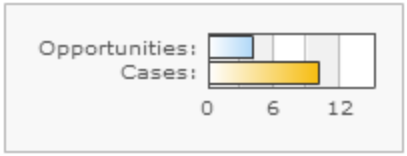
Note that you may need to replace the default Sage CRM install name used in the code below.

```
<script>

    crm.ready(function()
    {
        // Display a FustionCharts linear gauge in the Certainty field
        var strScriptPath = crm.installUrl() + "FusionCharts/FusionCharts.js";
        $.getScript(strScriptPath, function (data, textStatus, jqxhr)
        {
            // In the path to the HLinearGauge.swf file below, crm is the default Sage
CRM install name.
            // You may need to replace crm with the install name used in your
environment.
            var myChart = new FusionCharts("/crm/FusionCharts/HLinearGauge.swf",
"myChartId", "200", "75", "0");
            myChart.setJSONData(
                {
                    "chart":
                    {
                        "lowerlimit": "0",
                        "upperlimit": "100",
                        "palette": "1",
                        "numbersuffix": "%",
                        "chartrightmargin": "20"
                    },

                    "pointers":
                    {
                        "pointer":
                        [
                            {
                                "value": crm("oppo_certainty").value()
                            }
                        ]
                    }
                })

            myChart.render("_Dataoppo_certainty");
        });
    })

</script>
```

To view the created widget, go to an Opportunity record and open the **Summary** tab.

Creating an Opportunities and Cases widget

This topic illustrates how to use FusionCharts to create a graphic widget that shows active Opportunities and Cases for each Company record in Sage CRM. The widget displays on the **Summary** tab for each Company and looks similar to the following:



To create this widget, do the following:

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | Company**.
3. Click the **Screens** tab.
4. In the **Screen Name** column, click **CompanyBoxLong**.
5. In the **Custom Content** text box, enter the following code, and then click **Save**.
   Note that you may need to replace the default Sage CRM install name used in the code below.

```
<script>

// Define variable  to store data returned by Ajax requests
var myResults = {};
crm.ready(function ()
{
    // Function script
    var strScriptPath = crm.installUrl() + "FusionCharts/FusionCharts.js";
    $.getScript(strScriptPath, function (data, textStatus, jqxhr)
    {
        // Get Opportunities and Cases for each Company and store them in variables
        // Make sure to properly sequence Ajax requests, because chart can only be added
after requests return data
        var companyID = crm.getArg("key1");
        var strOppoWhereClause = "oppo_primarycompanyid eq " + companyID;
        var strCaseWhereClause = "case_primarycompanyid eq " + companyID;var successCase
= function (crmRecord)
        {
            myResults.CaseCount = crmRecord.totalResults;

            // Define text labels for Cases
            CaseData =
            {
                "label": "Cases:",
                "value": myResults.CaseCount,
                "tooltext": "Cases in progress" + myResults.CaseCount
            }

            // Add chart to the top of the page (TopContent)
            var myOutPut = "<table border=0><tbody><tr><td><div
id=chartContainer>FusionWidgets XT will load here!</div></td></tr></tbody></table>";
            var x = $("#EWARE_TOP").children();
            var y = $(x).children();var z = $(y).children();
            var a = $(z).children("td:last");
            a.after(myOutPut);

            // Define chart appearance.
            // In the path to the Bar2D.swf file below, crm is the default Sage
CRM install name.
            // You may need to replace crm with the install name used in your
environment.
            var myChart = new FusionCharts("/crm/FusionCharts/Bar2D.swf", "myChartId",
"200", "75", "0");
            myChart.setJSONData(
                {
                    "chart":
```

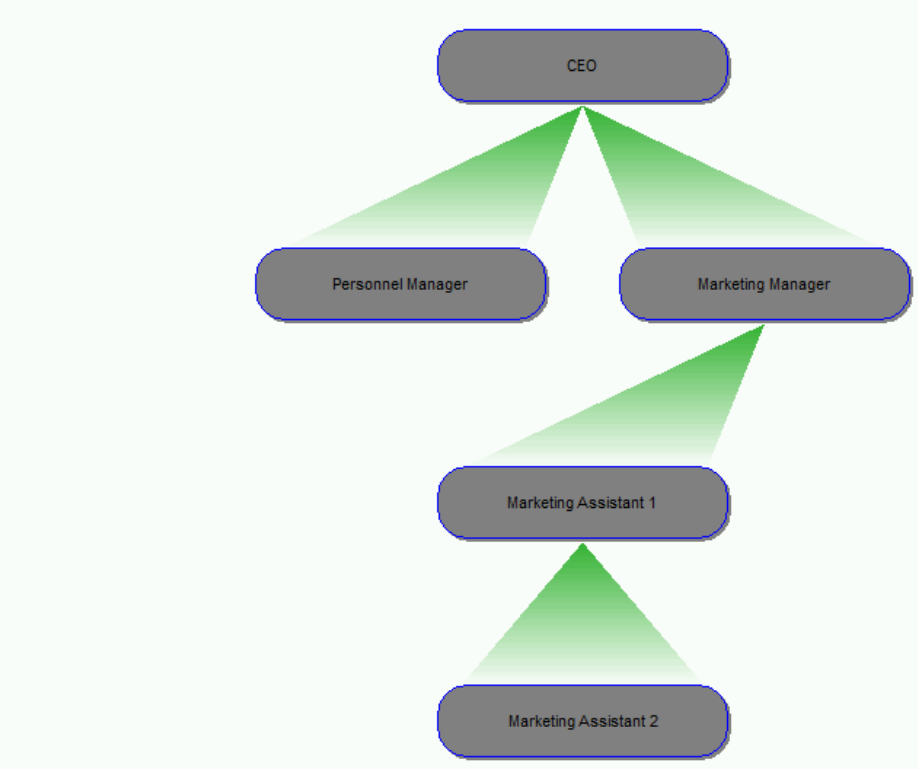
```
                    {
                        "bgcolor": "F1F1F1",
                        "showvalues": "0",
                        "canvasborderthickness": "1",
                    },

                    "data":
                    [
                        OppoData,CaseData
                    ]
                })

            myChart.render("chartContainer");
        }

        var successOppo = function (crmRecord)
        {
            myResults.OppoCount = crmRecord.totalResults;

            // Define text labels for Opportunities
            OppoData =
            {
                "label": "Opportunities:",
                "value": myResults.OppoCount,
                "tooltext": "Opportunities in progress" + myResults.OppoCount
            }

            // Get Cases for Company
            crm.sdata(
                {
                    entity: "cases",
                    where: strCaseWhereClause,
                    success: successCase
                });
        }

            // Get Opportunities for Company
            crm.sdata(
                {
                    entity: "opportunity",
                    where: strOppoWhereClause,
                    success: successOppo
                });
    })
})

</script>
```

To view the created widget, go to a Company record and open the **Summary** tab.

Creating an organization chart

This topic illustrates how to create an organization chart. You can display an organization chart on a new custom tab for each entity record you want. Below is an example organization chart:

My chart



The following example shows how to display this chart on a new tab for each Company:

1. Create a custom ASP file that contains the following code:

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

// Get the orgchart block and store it in the org variable.
var org;
org=CRM.GetBlock('orgchart');

// Add the chart title. If you omit the Title parameter, no title is added.
org.OrgTree("Title","My chart");

// Optionally, specify an icon for the top-level box in the chart.
// org.OrgTree("EntityIcon","C:\\FileName.bmp");

// Optionally, specify a custom background for each box in the chart.
// org.OrgTree("EntityImage","C:\\FileName.bmp");

// Add the top level of the chart (CEO).
org.OrgTree("Add",",CEO,false,");

// Add Marketing Manager.
org.OrgTree("Add","CEO,Marketing Manager,true,");

// Add Personnel Manager.
org.OrgTree("Add","CEO,Personnel Manager,true,");

// Add Marketing Assistant 1 and Marketing Assistant 2.
org.OrgTree("Add","Marketing Manager,Marketing Assistant 1,true,");
org.OrgTree("Add","Marketing Assistant 1, Marketing Assistant 2,true,");

// Define the style of connectors in the chart. You can use Arrow, Ray, or Line.
org.OrgTree("LineStyle","Ray");

// Define the style of boxes in the chart. You can use Square or Round.
org.OrgTree("BoxStyle","Round");

// Optionally, define the overall chart height. To use the default height, omit this
line.
// org.OrgTree("FullBoxHeight","150");

// Optionally, define the overall chart width. To use the default width, omit this line.
// org.OrgTree("FullBoxWidth","250");

// Optionally, define the height of each box in the chart. To use the default height,
omit this line.
// org.OrgTree("BoxHeight","50");

// Optionally, define the width of each box in the chart. To use the default width, omit
this line.
// org.OrgTree("BoxWidth","100");

// Specify if you want to use animation to display the chart.
// org.OrgTree("Animation","True");

// Specify if you want to display chart legend.
org.OrgTree("ShowLegend","False");
```

```
// Show your chart on the screen.
CRM.AddContent(org.Execute());
Response.Write(CRM.GetPage());

%>
</body>
</html>
```

2. Save the ASP file in the **CustomPages** folder in the Sage CRM installation directory. The default location of the **CustomPages** folder is %ProgramFiles (x86)%\Sage\CRM\CRM\WWWRoot\CustomPages.
3. Create a new tab and link it to your ASP file:
    a. Log on to Sage CRM as a system administrator.
    b. Go to **<My Profile> | Administration | Customization**.
    c. Click **Company**, and then click **Tabs**.
    d. In the **Tab Group Name** column, click **Company**.
    e. Under **Properties**, use the following options:
        - **Caption**. Enter the new tab name.
        - **Action**. Select **customfile**.
        - **Custom File**. Enter the name of the custom ASP file you created in step 1 of this procedure.
4. Click **Update**, and then click **Save**.

You can view the created organization chart on the new tab you created for each Company record.

APIs

- **Using Web Services API**
- **Using SData API**
- **Using .NET API**

Using Web Services API

About Web Services

Sage CRM Web Services API (Application Programming Interface) enables developers to manipulate records in Sage CRM with SOAP (Simple Object Access Protocol) over HTTP using XML (Extensible Markup Language).

Developers can use the Sage CRM Web Services API to do the following:

- Programmatically create, read, update, and delete entity records in the Sage CRM database. For example, Companies, People, Opportunities, Cases, Quotes, and Orders.
- Integrate third-party applications used within your organization with Sage CRM.

Sage CRM Web Services provide a standardized method for integrating web-based applications using the following components via an Internet protocol backbone:

- **XML**. Tags the data.
- **SOAP**. Transfers the data. For detailed information about SOAP, go to **http://www.w3.org/TR/SOAP**.
- **WSDL**. Describes the available services.

Web Services allow organizations to exchange data without in-depth knowledge of the IT systems behind the firewall. Web Services don't 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 Services 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 applications from different sources to communicate with each other without time-consuming custom coding. All communication is in XML, so you aren't limited to any one programming language.

To use the Sage CRM Web Services, complete the following steps:

1. Enable and configure Web Services. For more information, see:
    - **Prerequisites for using Web Services**
    - **Enabling Web Services for a user**
    - **Configuring Web Services**
2. View the WSDL file and prepare and submit your request to Web Services.
   For more information, see **Using the WSDL file**.
3. Receive and process the response from Web Services.

The SOAP Web Services API is only available in certain editions of Sage CRM. For more information, see the Sage CRM Editions Comparison Guide.

Prerequisites for using Web Services

To use Sage CRM Web Services, you must have the following installed on the Sage CRM server:

- A valid Sage CRM license key
- MSXML 4 Service Pack 2
  You can download this package at **http://www.microsoft.com/en-us/download/details.aspx?id=19662**

Sage CRM Web Services support all development environments that are compatible with SOAP 1.1. These environments include:

- Microsoft Visual Studio 2012 or later (C#, J#, VB.NET)
- Microsoft Visual C# 2010 Express Edition

Enabling Web Services for a user

To use the Sage CRM Web Services under a particular user account, you need to enable Web Services for that user account in Sage CRM.

Before enabling Web Services for a user, consider the following:

- Only one Web Services user can be logged on to Sage CRM at any given time. If a user tries to log on as another application, the user is prompted to log out first. However, it's possible to log on to the desktop or from a device with the same ID while a Web Service application is running.
- Web Services cannot work with Sage CRM fields to which the user account does not have access. For more information about using field security, see the *System Administrator Help*.

To enable Web Services for a user:

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Users | Users**.
3. Find and select the user account for which you want to enable Web Services.
4. Click **Change** .
5. Under **Allow Web Service Access**, select **True**.
6. Click **Save**.

Configuring Web Services

To configure Sage CRM Web Services, do the following:

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | System | Web Services**.
3. Click **Change** and then specify the Web Services configuration options described below.

Option	Description
Maximum number of records to return	Specifies the maximum number of records that Web Services return in a batch in response to a query. When this option is set to **0**, all records matching the query are returned in a single batch. This option is used in conjunction with the query and queryrecord methods. When a batch is returned, you're prompted to call the next batch, until all records matching the query are returned.
Maximum size of request	Specifies the maximum size of request (in characters) that can be sent to Web Services.
Make WSDL available to all	Specifies whether the Sage CRM Web Services WSDL file is accessible. Possible values: • **Yes**. Anyone can access the WSDL file.

Option	Description
	• **No**. Users and system administrators cannot access the WSDL file When configuring and testing Web Services, we recommend that you set this option to **Yes**. When you're done, set this option to **No** for better security. To view the WSDL file, in your web browser, enter the URL in the following format: http://<ComputerName>/<InstallName>/eware.dll/webservices/webservice.wsdl.
Enable Web Services	Specifies whether Web Services are enabled in Sage CRM. Possible values: • **Yes**(recommended). Enables Web Services. • **No**. Disables Web Services. To enable or disable Web Services for an individual entity, log on to Sage CRM as a system administrator, go to **<My Profile> | Administration | Customization | <Entity> | External Access**, and then use the **Allow Web Service Access** option.

Dropdown fields as strings in WSDL file	Specifies whether to display selection fields and their enumeration values in the WSDL file.
	
	Possible values:
	
	• **Yes** (recommended). Hides selection fields and their enumeration values in the WSDL file.
	• **No**. Displays selection fields and their enumeration values in the WSDL file.
	
	When this option is set to **No**, a selection field (comm_status in this case) displayed in the WSDL file looks similar to the following:
	
	`<simpleType name="comm_status">`
	`<restriction base="xsd:string">`
	`<enumeration value="Cancelled"/>`
	`<enumeration value="Complete"/>`
	`<enumeration value="Pending"/>`
	`<enumeration value="In Progress"/>`
	`</restriction>`
	`</simpleType>`
	
	When this option is set to **Yes**, information about selection fields is excluded from the WSDL file.

Option	Description
	Note: Enumerated values are returned in the default system language.
Send and return all dates and times in universal time	Specifies the format in which Sage CRM Web Services send and receive times and dates. Possible values: • **Yes**. Specifies to use Coordinated Universal Time (UTC). • **No**. Specifies to use the format set on the Sage CRM server. This option is important when migrating users from different time zones.
Accept web request from IP address	Specifies the unique IP address from which Web Services accept requests. You can only specify one IP address in this option. Leave this option blank to allow requests from all IP addresses.
Force web service logon	Specifies how to handle the client's attempt to log back on to Web Services when the connection is unexpectedly interrupted. Possible values: • **Yes** (recommended). Specifies that a new instance of the client is allowed to log on to Web Services following an interrupted connection. This automatically logs out the old instance of the client. • **No**. Specifies that a new instance of the client is blocked from logging on to Web Services. The old instance of the client remains logged on.

Required fields in quotes and orders

This topic lists the fields that must be populated when inserting or updating data in quotes and orders.

Line item type	Fields required for insert operations	Fields required for update operations

Simple	• orderquoteid
	• opportunityid
	• lineitemtype
	Can take one of these values:
	  • **i**. Simple.
	  • **f**. Free text.
	  • **c**. Comment.
	• productid
	• uomid
	• quantity
	• quotedprice
		• quantity
		• quotedprice
		• uomid

Free text	• orderquoteid
	• opportunityid
	• lineitemtype
	Can take one of these values:
	  • **i**. Simple.
	  • **f**. Free text.
	  • **c**. Comment.
	• description
	• quantity
	• quotedprice
		• description
		• quantity

Comment	• orderquoteid
	• opportunityid
	• lineitemtype
	Can take one of these values:
	  • **i**. Simple.
	  • **f**. Free text.
	  • **c**. Comment.
	• description
		• description

The following fields cannot be updated by the user:

- linetype
- orderquoteid

The following fields are populated automatically by Sage CRM and cannot be changed by the user:

- quotedpricetotal
- listprice
- discount
- discountsum

Using the WSDL file

Sage CRM provides a Web Services description language file called a WSDL file. The WSDL file describes the APIs that Sage CRM exposes, and the XML types that the APIs expect. The file also describes the server and location of specific services. When the client has read and parsed the WSDL file, it can call the APIs in the same way as any typical function call. Since data is passed and returned as XML, it can be easily interpreted and manipulated by the client.

To access the WSDL file, in your web browser, enter the URL in the following format:

http://<ComputerName>/<InstallName>/eware.dll/webservices/webservice.wsdl

where

- **<ComputerName>**. Is the name of the Sage CRM server.
- **<InstallName>**. Is the name of the Sage CRM installation folder.

If you're using Microsoft Visual Studio to create a client application, your Visual Studio project should contain a web reference to the WSDL file. When you add the reference in Visual Studio, the main pane lists the methods available from the Web Services.

If you name the service **CRMWebServices**, a new folder called **CRMWebServices** is added to your project, which contains the files webservice.discomap and webservice.wsdl. The Web Service proxy is created automatically. This is a C# version of the WSDL file that handles the dispatch of data in SOAP format to the Web Service.

Web Services methods

Methods are actions invoked from the client computer, such as adding, updating, or deleting information on the Sage CRM server. Web Services methods send synchronous requests that are

committed automatically. Once committed, Sage CRM Web Services handle the request and returns a response. The client application handles the response accordingly.

All inserts should typically be performed on an entity basis. However, to facilitate integration, you can update a company or person with address, phone, and email information. In many systems, a single contact record represents company, person, phone, email, and address information.

The following is a list of Web Services methods defined in the WSDL file:

Method	Description
logon	Logs the client on to the server and starts a session.
logoff	Logs off the client from the server and terminates the session.
query	Executes a query on a specified object based on a `where` clause, and returns a record or record set that satisfies the query. Returns results in batches. You specify batch size in the **Maximum number of records to return** option. For more information, see **Configuring Web Services**. Each batch is accompanied by a flag called `More`. If `More` is **True**, 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	Returns 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, and `More` is **False**.
queryentity	Returns a record if you supply an object and its ID. Example: `queryentity(company, 42)`
queryid	Returns an object of type `aisid`. For more information, see **Abstract objects**. If you query the database with a `where` clause, a date, IDs, and flags denoting whether the record was created, updated, or deleted since that date, are returned. This is useful for data synchronization.
queryidnodate	Returns an object of type `aisid`. For more information, see **Abstract objects**.

Method	Description
	To return specific objects, use a `where` clause.
getmetadata	Returns a list of Sage CRM field types to provide metadata about the requested table.
getdropdownvalues	Returns a list of drop-down fields for a specified table, and the values that Sage CRM expects for each field.
add	Adds records or lists of records to a specified object. Example: `add("company", NewCompany1, New Company2, New Company3)`
addresource	Adds a user as a resource. This user is not a fully enabled user. This method facilitates data migration.
update	Updates records or lists of records for a specified object.
altercolumnwidth	Resizes a column width to ensure compatibility with third-party databases, for example, ACT!.
delete	Deletes records or lists of records for a specified object. This method cannot delete records from the following tables, because they contain historical data: • newproduct • uomfamily • productfamily • pricing • pricinglist
addrecord	Adds records or lists of records to a specified object. This method has a different signature than the add method and uses the lists of fields in the `crmrecord` type.
queryrecord	Executes a query on a specified object based on a `where` clause, and returns a record or record set that satisfies the query. This method has a different signature than the `query` method and uses the lists of fields in the `crmrecord` type.
nextqueryrecord	Returns the next batch of records matching a `queryrecord`. Each batch

Method	Description
	is accompanied by a flag called `More`. While `More` is **True**, you can continue to call `Next` until all batches are returned. More is then **False**.
updaterecord	Updates records or lists of records for a specified object. This method has a different signature than the update method and uses the lists of fields in the `crmrecord` type.
getallmetadata	Returns a list of fields associated with all tables and some type information.
getversionstring	Returns the Sage CRM version number.

Web Services objects

Web Services objects are programmatic representations of main entities (such as companies and people), secondary entities (such as addresses and products), and any custom entities that you add. Because the WSDL is generated dynamically, any customizations made to the system are picked up each time the WSDL is refreshed at the client side.

Data is manipulated when the Web Services API interacts with object properties, which represent fields in the entities.

- **Abstract objects**
- **Standard objects**

Abstract objects

Object	Description
ewarebase abstract	An abstract declaration from which all other Sage CRM objects inherit.
idbase abstract	An abstract declaration from which all ID types inherit.
ewarebase list	A list of the abstract objects above.
crmrecord type	An enumeration that represents the types of a Sage CRM field (string, datetime, integer, or decimal). Use the crmrecordtype object with its associated add, update, and delete functions

Object	Description

	to query an entity and return a list of fields that you can iterate through.
	
	It allows you to specify which fields you want returned in your query, and to dynamically add fields to the Web Services entities. It removes the need for strongly typed objects in client applications. Follow code samples closely when performing these tasks.
	
	Example
	
	`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");`
	`}`
	
	This example specifies a field list and an entity name, a Where clause and an order by. If you enter an asterisk (*) or leave the field list blank, all the fields are returned.

crmrecord	Contains an entity name and a list of objects of type `recordfield` that represent one record in the Sage CRM database.
aisid	Contains the ID of the record, the created and updated date, and a flag to indicate whether the record was added, updated, or deleted since the token was passed to `queryid`.
multiselectfield	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. These values are translations.
recordfield	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` in 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	Description
company	Represents the Company entity in Sage CRM.
person	Represents the Person entity in Sage CRM.
lead	Represents the Lead entity in Sage CRM.

Object	Description
communication	Represents the Communication entity in Sage CRM.
opportunity	Represents the Opportunity entity in Sage CRM.
cases	Represents the Cases entity in Sage CRM.
users	Represents the Users entity in Sage CRM.
quotes	Represents the Quotes entity in Sage CRM.
orders	Represents the Orders entity in Sage CRM.
quoteitem	Represents the QuoteItems entity in Sage CRM.
orderitem	Represents the Order Items entity in Sage CRM.
opportunityitem	Represents the Opportunity Item entity in Sage CRM.
currency	Represents the Currency entity in Sage CRM.
address	Represents the Address entity in Sage CRM.
phone	Represents the Phone entity in Sage CRM.
email	Represents the Email entity in Sage CRM.
newproduct	Represents the New Product entity in Sage CRM.
uom	Represents the Units of Measure entity in Sage CRM.
uomfamily	Represents the UOM Family entity in Sage CRM.
pricing	Represents the Pricing entity in Sage CRM.
pricinglist	Represents the Pricing List entity in Sage CRM.
productfamily	Represents the Product Families entity in Sage CRM.

Web Services selection fields

This topic lists the selection (or drop-down) fields that Web Services support for Sage CRM entities.

Entity	Selection fields
Company	• comp_employees • comp_indcode • comp_mailrestriction • comp_revenue • comp_sector • comp_source • comp_status • comp_territory • comp_type
Person	• pers_gender • pers_salutation • pers_source • pers_status • pers_territory • pers_titlecode
Lead	• lead_decisiontimeframe • lead_priority • lead_rating • lead_source • lead_stage • lead_status
Communication	• comm_action • comm_hasattachments • comm_notifydelta • comm_outcome • comm_priority • comm_status • comm_type
Opportunity	• oppo_priority • oppo_product

Entity	Selection fields
	• oppo_scenario • oppo_source • oppo_stage • oppo_status • oppo_type
Case	• case_foundver • case_problemtype • case_productarea • case_solutiontype • case_source • case_stage • case_status • case_targetver
Address	• addr_country • prod_uomcategory
Product	• addr_country • prod_uomcategory

By default, the WSDL file displays selection fields and their possible values. To hide selection fields in the WSDL file, use the **Dropdown fields as strings in WSDL file** option in the Web Services configuration settings. For more information , see **Configuring Web Services**.

Getting possible values of a selection field

Use the `getdropdownvalues` C# method to get the list of possible values for a selection field in Sage CRM. The following example populates 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;
}
```

The next example displays the comp_sector selection field values in a comboSector combo-box, where the web service object is named `oWebService`.

```
LoadDropDowns("company", "sector", comboSector, oWebService);
```

Sample SOAP requests and XML

- **Logon request**
- **Authentication response**
- **Logoff request**
- **Delete request**
- **Update request**
- **Query request**
- **XML representing a company**

Logon request

This C# example logs on to Sage CRM.

```
// 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
  {
    HttpWebRequest request = (HttpWebRequest) WebRequest.Create
("http://cloud.sagecrm.com/myCustomerID/eware.dll/webservices/CRMwebservice.wsdl");
    HttpWebResponse response = (HttpWebResponse) request.GetResponse(); binding = new WebService
();
    SID = binding.logon("admin", ""); binding.SessionHeaderValue = new SessionHeader();
    //Persistent SID
    binding.SessionHeaderValue.sessionId = SID.sessionid;
    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 process.

```
<?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>
```

Authentication response

The following is a sample authentication response indicating that authentication against the endpoint was a success. The response format is JSON.

```
{
  "installName": "<myCustomerID>",
  "wsdlUrl":
"https://cloud.na.sagecrm.com/<CustomerID>/eware.dll/webservices/CRMwebservice.wsdl",
  "wsUrlConnection": "https://cloud.na.sagecrm.com/<CustomerID>/eware.dll/webservices/",
  "web2LeadUrl": "https://cloud.na.sagecrm.com/<CustomerID>/eware.dll/SubmitLead",
  "sDataUrl": "https://cloud.na.sagecrm.com/sdata/<CustomerID>j/sagecrm/",
  "edition": "professional",
  "domain": "cloud.na.sagecrm.com",
  "userId": 1,
  "teamId": 1,
  "found": true,
  "userAuthenticated": true,
  "userWebServicesEnabled": true,
  "userDisabled": false
}
```

Logoff request

This C# example logs off from Sage CRM.

```
//Log off
if (binding.logoff(binding.SessionHeaderValue.sessionId).success)
{
  Write("Logged off");
}
```

This XML example logs off from Sage CRM.

```
<?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>
```

Delete request

This C# example deletes a company whose ID is 1.

```
ewarebase[] idList = new ewarebase[1];
crmid aCompanyId = new crmid();
aCompanyId.crmid1 = 1; //1 is id of company to delete
idList[0] = aCompanyId;
deleteresult aResult = binding.delete("company", idList);
if (aResult.deletesuccess == true)
{
  Write("Number deleted successfully : " + aResult.numberdeleted);
}
```

This is the XML request that Web Services process.

```
<?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>
```

Update request

This C# example changes the name of the company whose ID is 66.

```
private static void UpdateACompany()
{
  String idString = "66";
  String newName = "newName";
  //can update a number of companies
  ewarebase[] companyList = new ewarebase[1];
  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 process.

```
<?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>
```

Query request

This example queries a company record whose ID is 66.

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

XML representing a company

This XML represents 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>
```

C# code samples

Sage CRM is supplied with a number of Microsoft Visual Studio 2005 C# solutions that contain code samples you can use when developing Web Services applications for Sage CRM.

On a Sage CRM server, you can find these C# solutions in the following location:
<Sage CRM installation folder>\WWWRoot\Examples\WebServices

By default, Sage CRM is installed to **%ProgramFiles(x86)%\Sage\CRM\CRM**.

In the specified location, you will find the **Sage CRM Web Services Sample Code.zip** file that includes the following folders:

- **CRM Add Resource**. Contains code that adds a resource (user) to Sage 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**. Contains code that changes the width of the comp_practicefield column in the Company table.
- **CRM Create**. Contains code that creates company, person, address, and phone records in Sage CRM.
- **CRM Delete**. Contains code that deletes a specified company from Sage CRM.
- **CRM LogOn and LogOff**. Contains code that logs on and logs off from Sage CRM. At logon, this code returns and displays a session ID.
- **CRM MetaData**. Contains code that gets information about Sage CRM tables. The user can select from five entities: Company, Person, Case, Opportunity, or Communication. When the user clicks **MetaData**, all table columns are displayed in the left-hand pane. The user can highlight any column to display information about the column. When the user clicks **AllMetaData**, the main Sage CRM tables are listed in a drop-down. The user can select a table to view all associated columns.
- **CRM QueryEntity**. Contains code that queries the Company table. The user enters a known company ID and clicks **Search** to display all people associated with the specified company.
- **CRM QueryIdNoDate**. Contains code that allows the user to enter a date. The user clicks **Query N/D** to display a list of company IDs where the update date is greater than or equal to the date entered.
- **CRM SelectionLists**. Contains code that works with selection lists. The user can select from seven tables: Company, Person, Opportunity, Case, Communication, Solution, or Library. When the user clicks **List**, the **Lists** drop-down list is populated with the selection fields in the table. When the user chooses a selection field and clicks **Lists Items**, the left-hand pane is populated with values from the selection field.
- **CRM SID Grabber**. Contains code that displays the current session ID and Sage CRM version number. At least one user must be logged on to Sage CRM.
- **CRM SID_Key**. Contains code that displays the current session ID on the form. At least one user must be logged on to Sage CRM.
- **CRM Update**. Contains code that updates the Source and Website fields in Sage CRM.
- **CRM Version**. Contains code that displays the current session ID on the form and Sage CRM version number in a pop-up box.

To run sample code, enter your Sage CRM user name and password, and click **Log On**. When you're finished working with code samples, click **Log Off**. Make sure that you specify the correct reference to the WSDL file. For more information, see Using the WSDL file.

Using SData API

- **About SData**
- **Prerequisites for using SData**
- **SData authentication**
- **Managing SData access**
- **HTTP request URL**
- **SData endpoints**

About SData

SData is a standard that can be used to read, write, update, and delete data between applications. It also provides 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 enables desktop, server, and web-based Sage applications to communicate with each other, third-party applications, and the web.

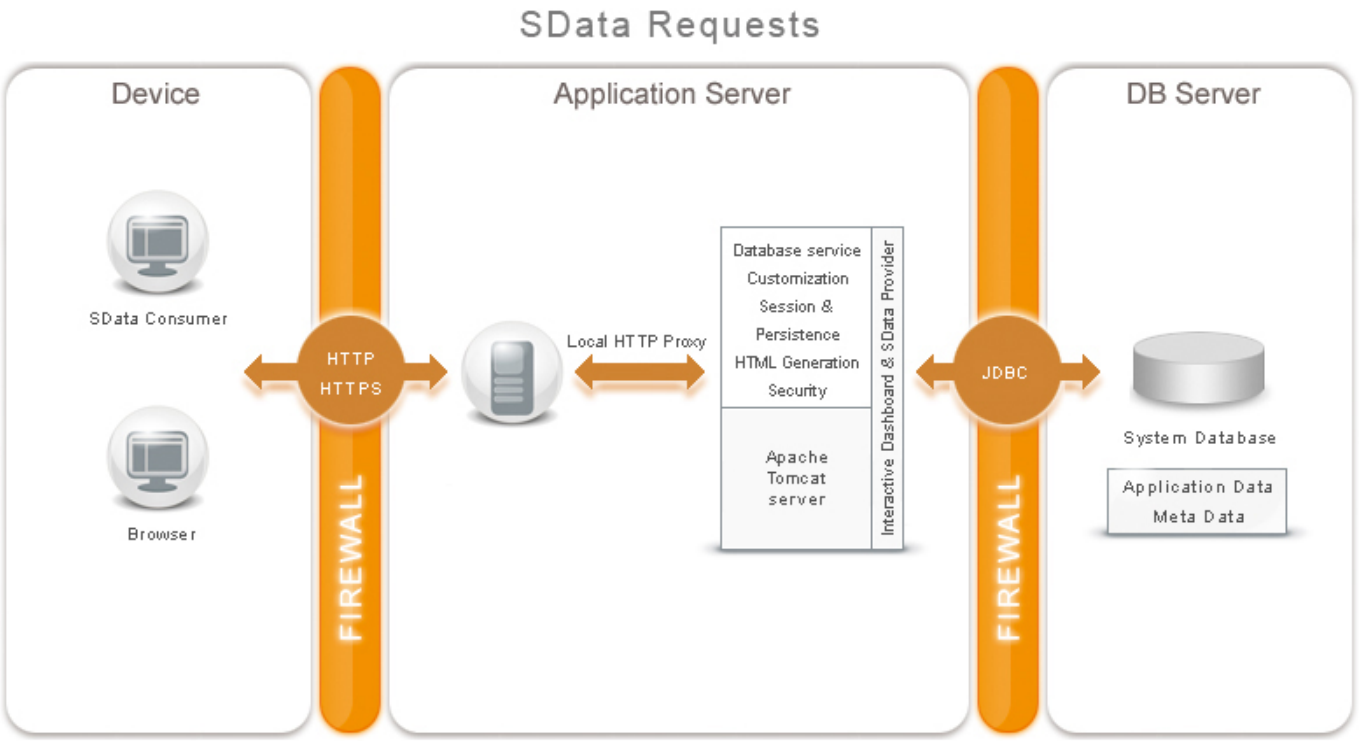
In Sage CRM, SData can only be used to read data by using the ATOM feed technology. SData is built on top of common industry standards including HTTP, XML, REST, and Atom/RSS. Any Sage CRM user can use SData. It doesn’t use up a user license so a user can be logged into Sage CRM and access data through a third-party, external application.

Each entity, custom table, and user view in Sage CRM can be exposed for read-only SData access. An XSD schema definition is available to provide information about which entities are exposed.

An SData request triggers an XML-based response. The response can include a single record or a collection of records. If the response size for an SData request exceeds 100 records, it defaults to 100 records. If a user tries to access SData from an external system and defines a page count of 200 records in the URL, the payload returns the records in blocks of 100 records per page. The same applies if a user defines a URL with no pagination.

For more information about SData, please refer to the following:

- *Sage CRM User Help* posted on the **Sage CRM Help Center**.
- **SData Specification** posted at **http://sage.github.io/SData-2.0**.



Prerequisites for using SData

To use SData, you must have the following installed on the Sage CRM server:

- A valid Sage CRM license key
- Apache Tomcat (installed as part of Sage CRM)

Apache Tomcat is a servlet container developed by the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a pure Java HTTP web server environment for Java code to run. Every request issued for the SData Provider is redirected to the Tomcat Server.

If you're working with Sage CRM and encounter a problem that requires a web server reset, you might need to reset both Microsoft Web Server (IIS) and Apache Tomcat. For more information, see the System Administrator Guide or Help.

SData authentication

All SData requests must include authentication data (user name and password) in their headers. SData access is subject to the same territory model as Sage CRM users. Profile security (CRUD

rights per entity) and access rights depend on the user type, for example, administrator or non-administrator.

Make sure that you use Base64 encoding to encrypt the user name and password supplied in the HTTP request header with X-Sage-Authorization line included, as shown below:

```
request.Headers.Add("X-Sage-Authorization", "Basic "+ Convert.ToBase64String
(Encoding.ASCII.GetBytes(this.userName + ":" + this.password)));
```

For better security, we recommend that you send authentication requests via HTTPS.

Note: You can use a session ID to authenticate calls to some SData endpoints. For details, see**Authentication: retrieve session ID for a user**.

Managing SData access

In the Sage CRM user interface, you can enable or disable read-only SData access to the following:

- **Entities**
- **Custom tables**
- **User views of entities**

Entities

Changing SData settings for an entity doesn't affect SData settings specified for the user views of the entity. You can also enable or disable read-only SData access when creating a new custom entity by using the Advanced Customization Wizard. For more information, see the *System Administrator Help*.

To enable or disable read-only SData access to an entity:

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | External Access**, where <Entity> is the entity for which you want to enable or disable read-only SData access.
3. Click **Change**.
4. Under **Read-only SData**, select one of the following:
    - **Yes**. Enables read-only SData access to the entity.
    - **No**. Disables read-only SData access to the entity.
5. Click **Save**.

Custom tables

To enable or disable read-only SData when creating a new custom table:

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Advanced Customization | Tables and Databases**.
3. Click **Create Table**.
4. On the page that opens, under **Read-only SData**, select one of the following:
    - **Yes**. Enables read-only SData access to the table being created.
    - **No**. Disables read-only SData access to the table being created.
5. Fill in other fields as required.
6. Click **Save** to create new table.

User views of entities

To enable or disable read-only SData access to the user view of an entity:

1. Log on to Sage CRM as a system administrator.
2. Go to **<My Profile> | Administration | Customization | <Entity> | Views.**
   where **<Entity>** is the entity for whose user view you want to enable or disable read-only SData access.
3. In the **View Name** column, click the user view for which you want to enable or disable SData access.
4. On the page that opens, click **Change**.
5. Do one of the following:
    - To enable SData access, select the **SData view** check box.
    - To disable SData access, clear the **SData view** check box.
6. Click **Save**.

Changing SData settings for an entity doesn't affect SData settings specified for the user views of the entity.

HTTP request URL

To access an entity, custom table, or user view via SData, you need to make an HTTP request in the format that is specific to your Sage CRM environment. The response is always XML.

Your HTTP request should have the following format:

```
http://<Server>/sdata/<InstallName>j/sagecrm/-/<Target>
```

Where:

- **<Server>**. Is the name of the computer on which Sage CRM is installed.
- **<InstallName>**. Is the name of the Sage CRM install (this is **crm** by default). Make sure to append **j** after the install name as shown in the HTTP request format above.
- **<Target>**. Is the name of the entity, custom table, or view you want to access.

The following table provides sample HTTP requests and their descriptions.

HTTP request	Description
`http://SageCrmServer/sdata/crmj/sagecrm/-/company/$schema`	Returns the company schema.
`http://SageCrmServer/sdata/crmj/sagecrm/-/company('43')`	Return the record for the company whose ID (comp_companyid field value) is 43.
`http://SageCrmServer/sdata/crmj/sagecrm/-/company?where=comp_companyid eq '43'`	
`http://SageCrmServer/sdata/crmj/sagecrm/-/company(comp_companyid eq '43')`	
`http://SageCrmServer/sdata/crmj/sagecrm/-/company?where=comp_name like 'A%' and comp_type eq 'customer%'`	Returns the company records where the company name begins with A and the company type is customer.
`http://SageCrmServer/sdata/crmj/sagecrm/-/company?where=comp_companyid eq '43'&include=person,address,phoneCollection,emailCollection`	Return the record for the company whose ID (comp_companyid field value) is 43. The response includes the name, address, phone, and email of the primary person associated with the company.
`http://CRMserver/sdata/crmj/sagecrm/-/company?where=comp_companyid eq '43'&include=Person`	
`http://CRMserver/sdata/crmj/sagecrm/-/vCompanySummary`	Returns all records from the vCompanySummary user view.
`http://CRMserver/sdata/crmj/sagecrm/-/person?where=concat(pers_firstname, pers_lastname) eq 'William Agnew'`	Returns the record for a person whose name is William Agnew.

HTTP request	Description
`http://CRMserver/sdata/crmj/sagecrm/-/quoteitems?where=quit_productid ge 3 and quit_productid le 10`	Returns quote records for products whose IDs fall between 3 and 10.
`http://CRMserver/sdata/crmj/sagecrm/-/quoteitems?where=quit_updateddate ge currenttimestamp()`	Returns quote records that were updated on the current date or earlier.
`http://CRMserver/sdata/crmj/sagecrm/-/company('43')&SID=61546204736053`	Authenticates the user by using session ID (SID) and returns the record for the company whose ID (comp_companyid field value) is 43. In Sage CRM version 7.1 and later, you can use session ID (SID) in the SData URL as an alternative authentication mechamism in SData requests.

You can also use fast SData requests for AJAX in Custom Content and OnChange scripts, for example:

```
<script>
function GetKeyValue(querystringname) {
  var strPath = window.location.search.substring(1);
  var arrayKeys = strPath.split("&");
  for (var i = 0; i < arrayKeys.length; i++) {
    var arrayValue = arrayKeys[i].split("=");
    if (arrayValue[0].toLowerCase() == querystringname.toLowerCase()) {
      return arrayValue[1];
    }
  }
  return "";
}
window.alert("start:" + GetKeyValue("SID"));
XmlHttp = new XMLHttpRequest();
var strURL = "http://richardsj-lt/sdata/crm71j/sagecrm/-/company?where=comp_companyid in ('43',
'45')&SID=" + GetKeyValue("SID");
XmlHttp.open('GET', strURL, false);
window.alert("open");
XmlHttp.send(null);
window.alert("send");
var strHtml = XmlHttp.responseText;
XmlHttp = null; // always clear the XmlHttp object when you are done to avoid memory leaks
window.alert("test:" + strHtml);
window.alert("end");
</script>
```

SData endpoints

This section provides information about some of the endpoints available via the SData API. To see how these endpoints work, use the provided Postman collection.

- **Postman collection**
- **Authentication: retrieve session ID for a user**
- **Search entities using Quick Find**
- **Retrieve date/time (calendar) preferences for a user**
- **Retrieve calendar translations for a user**
- **Retrieve metadata definitions of translations**
- **Retrieve metadata definitions of a screen or list**
- **Retrieve favourites for a user**
- **Retrieve active notifications for a user**
- **Retrieve notification options for a user**
- **Upload a file to a folder**
- **Delete all files uploaded in a session**

Postman collection

To see the SData API endpoints in action, you can use **Postman**, a multiplatform REST client with intuitive GUI for configuring HTTP requests, designing JSON payloads, and viewing HTTP responses.

We have prepared a Postman collection demonstrating how to use endpoints exposed through the SData API.

For steps on how to download and use the Postman collection, **go to the Sage CRM RESTful API documentation**, select the RESTful API reference for your Sage CRM version, and see the **Postman collection** section.

Prerequisites

You must have Sage CRM with demo data installed. This is required because the Postman collection uses the default Admin account that is installed with demo data.

Authentication: retrieve session ID for a user

You can use a session ID (SID) to authenticate your calls to the SData API and perform actions under the user account to which the SID belongs. To retrieve a session ID, submit your user name and password to Sage CRM as shown below.

Headers

Key	Value
`Content-Type`	`application/xml`

HTTP

```
POST http://{{server}}/crm/eware.dll/webservices/soap
```

URI parameters

Name	Description
`{{server}}`	Name or IP address of a Sage CRM server.

Request body

Tag	Description
<username></username>	Use this tag to submit the user name of the Sage CRM account whose SID you want to retrieve. **Example** `<username>Admin</username>`
<password></password>	Use this tag to submit the password of the Sage CRM account whose SID you want to retrieve.

Sample response

```
<?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>
        <logonresponsetype xmlns="http://tempuri.org/type">
            <result>
                <sessionid>125407476439516</sessionid>
            </result>
        </logonresponsetype>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Name	Description
<sessionid></sessionid>	Contains SID of the user account.

Search entities using Quick Find

Quick Find can search across single-line text, email address, and URL fields of the company, people, opportunity, lead, communication, order, quote, case, solution, library document, and custom entity records at once. Optionally, you can narrow your search down to a single entity.

For more information about configuring and using Quick Find, see *System Administrator Help* and *User Help*.

Authentication

Session ID. For details, see **Authentication: retrieve session ID for a user**.

HTTP

```
GET http://{{server}}/sdata/
{{install}}j/sagecrm2/$service/quickFind/getResults?query=eurolandia&SID={{sid}}&entity={{entity
name}}
```

URI parameters

Name	Description
{{server}}	Name or IP address of a Sage CRM server.
{{install}}	Sage CRM installation name. By default, this is `CRM`.
query	Key terms to search for. **Example** `query=eurolandia company invoice`
{{sid}}	Session ID of the user account under which to access Quick Find.
entity	*Optional*. The entity you want to search. When you omit this parameter, Sage CRM searches all supported standard and custom entities at once. This parameter can take one of the following values: • `Company` • `Person` • `Opportunity` • `Lead` • `Communication` • `Orders` • `Quotes` • `Cases` • `Solutions` • `Library` • Custom entitity name The parameter values are case sensitive, use them exactly as shown above. When specifying a custom entity name, enter it exactly as it appears in the Sage CRM user interface. **Example** `entity=Company`
{{entity name}}	Name of the entity you want to search.

Request body

None

Response

A JSON object containing the definitions of records that meet your search criteria.

Retrieve date/time (calendar) preferences for a user

Authentication

Session ID. For details, see **Authentication: retrieve session ID for a user**.

HTTP

```
GET http://{{server}}/sdata/{{install}}j/userdata?Action=getCalendarUserPreferences&SID={{sid}}
```

URI parameters

Name	Description
{{server}}	Name or IP address of a Sage CRM server.
{{install}}	Sage CRM installation name. By default, this is `CRM`.
{{sid}}	Session ID of the user account whose calendar preferences to retrieve.

Request body

None

Response

A JSON object containing date/time (calendar) preferences for the specified user.

Retrieve calendar translations for a user

Authentication

Session ID. For details, see **Authentication: retrieve session ID for a user**.

HTTP

```
GET http://{{server}}/sdata/{{install}}j/userdata?Action=getCalendarTranslations&SID={{sid}}
```

URI parameters

Name	Description
{{server}}	Name or IP address of a Sage CRM server.
{{install}}	Sage CRM installation name. By default, this is `CRM`.
{{sid}}	Session ID of the user account whose calendar translations to retrieve.

Request body

None

Response

A JSON object containing calendar translations in the language of the specified user.

Retrieve metadata definitions of translations

Authentication

Session ID. For details, see **Authentication: retrieve session ID for a user**.

Headers

Key	Value
`Content-Type`	`application/json`

HTTP

```
POST http://{{server}}/sdata/{{install}}j/metadata/-/$service/getTrans?SID={{sid}}
```

URI parameters

Name	Description
{{server}}	Name or IP address of a Sage CRM server.
{{install}}	Sage CRM installation name. By default, this is `CRM`.
{{sid}}	Session ID of the user account under which to access Sage CRM.

Request body

An array of caption family, caption code, and caption language values.

Name	Description
captFamily	Name of the caption family to which the caption belongs.
captCode	Code of the caption.
captLanguage	Language of the caption. Valid values: • DE • ES • FR • UK • US

Response

A JSON object containing metadata definitions of translations for the specified language and user account.

Retrieve metadata definitions of a screen or list

Authentication

Session ID. For details, see **Authentication: retrieve session ID for a user**.

HTTP

```
GET http://{{server}}/sdata/{{install}}j/metadata/-/$service/getCustomObject?name=CaseGrid&SID=
{{sid}}
```

URI parameters

Name	Description
{{server}}	Name or IP address of a Sage CRM server.
{{install}}	Sage CRM installation name. By default, this is `CRM`.
name	Name of the screen or list whose metadata definitions to retrieve.
{{sid}}	Session ID of the user account under which to access Sage CRM.

Request body

None

Response

A JSON object containing metadata definitions of each field included in the specified screen or list.

Retrieve favourites for a user

Authentication

Session ID. For details, see **Authentication: retrieve session ID for a user**.

HTTP

```
GET http://{{server}}/sdata/{{install}}j/userdata?Action=getFavourites&SID={{sid}}
```

URI parameters

Name	Description
{{server}}	Name or IP address of a Sage CRM server.
{{install}}	Sage CRM installation name. By default, this is `CRM`.
{{sid}}	Session ID of the user account whose favourites to retrieve.

Request body

None

Response

A JSON object containing an array of records added to favourites by the specified user.

Retrieve active notifications for a user

Authentication

Session ID. For details, see **Authentication: retrieve session ID for a user**.

HTTP

```
GET http://{{server}}/sdata/{{install}}j/userdata?Action=getNotifications&SID={{sid}}
```

URI parameters

Name	Description
{{server}}	Name or IP address of a Sage CRM server.
{{install}}	Sage CRM installation name. By default, this is `CRM`.
{{sid}}	Session ID of the user account whose notifications to retrieve.

Request body

None

Response

A JSON object containing an array of active notifications for the specified user.

Retrieve notification options for a user

Authentication

Session ID. For details, see **Authentication: retrieve session ID for a user**.

HTTP

```
GET http://{{server}}/sdata/{{install}}j/userdata?Action=getNotificationOptions&SID={{sid}}
```

URI parameters

Name	Description
{{server}}	Name or IP address of a Sage CRM server.
{{install}}	Sage CRM installation name. By default, this is `CRM`.
{{sid}}	Session ID of the user account whose notification options to retrieve.

Request body

None

Response

A JSON object containing an array of notification options for the specified user.

Upload a file to a folder

This endpoint enables you to upload a file to a folder on a Sage CRM server. You can upload only one file at a time.

Authentication

Session ID. For details, see **Authentication: retrieve session ID for a user**.

Headers

Key	Value
`Content-Type`	`multipart/form-data`

HTTP

```
POST http://{{server}}/sdata/{{install}}j/sagecrm/-/$service/fileUpload?SID={{sid}}
```

URI parameters

Name	Description
{{server}}	The name or IP address of a Sage CRM server.
{{install}}	Sage CRM installation name. By default, this is `CRM`.
{{sid}}	The session ID of the user account under which to upload files.

Request body

Key	Value
hiddenFileUpload	Required. Data of the files to upload.
hiddenUserFolder	Optional. The name of a subfolder in **<Sage CRM installation folder>\Library** to upload the specified file to. **Example value of hiddenUserFolder** `MyFolder` When you omit this parameter, the file is uploaded to **<Sage CRM installation folder>\Library\TEMP\<SID>**, where <SID> is the session ID you use to authenticate your call. By default, Sage CRM is installed to **%ProgramFiles(x86)%\Sage\CRM\CRM**.

Response

An XML object containing information about the uploaded file.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<root>
    <message></message>
    <file></file>
    <status></status>
```

```
</root>
```

Name	Description
<file></file>	Name of the uploaded file.
<status></status>	Upload status, such as `SUCCESS` or `FAILURE`.

Delete all files uploaded in a session

This endpoint enables you to delete all files uploaded to the following folder on a Sage CRM server:

<Sage CRM installation folder>\Library\TEMP\<SID>

Where <SID> is the ID of the session in which the files were uploaded.

Authentication

Session ID. For details, see **Authentication: retrieve session ID for a user**.

HTTP

```
DELETE http://{{server}}/sdata/{{install}}j/sagecrm/-/$service/fileDelete?SID={{sid}}
```

URI parameters

Name	Description
{{server}}	Name or IP address of a Sage CRM server.
{{install}}	Sage CRM installation name. By default, this is `CRM`.
{{sid}}	ID of the session in which the files to be deleted were uploaded.

Request body

None

Using .NET API

To view the .NET API documentation, **go to the Developer Portal**.

Reference

- **ASP objects**
- **Component Manager methods**

ASP objects

Object	Description
AddressList object	Provides access to the lists of recipients (To, Cc, and Bcc).
Attachment object	Provides access to message attachments.
AttachmentList object	Provides access to email attachments.
CRM object	Provides access to Sage CRM objects and functionality.
CRMBase object	Sets up the context information for the current view and displays the tabs for that view.
CRMBlock object	Is the base for all CRM blocks.
CRMChartGraphicBlock object	Use this object to draw charts and graphs.
CRMContainerBlock object	Use this object to group blocks on a screen.
CRMContentBlock object	Is a simple block that takes a text string and displays it on the page.
CRMEntryBlock object	Corresponds to a single field that's displayed or edited onscreen.
CRMEntryGroupBlock object	Corresponds to a screen in CRM.
CRMFileBlock object	Provides access to external files that are not part of the system.
CRMGraphicBlock object	Use this block to display images through an ASP page.
CRMGridColBlock object	Use this object to set the properties of an individual column within a list.
CRMListBlock object	Use this object to create and display lists.
CRMMarqueeBlock object	Use this object to add scrolling text to a page.
CRMMessageBlock object	Use this object to send messages in SMS and email format.
CRMOrgGraphicBlock	Use this object to create organizational charts.

Object	Description
object	
CRMPipelineGraphicBlock object	Use this object to create cross-sectional diagrams that represent data from an ASP page or table.
CRMQuery object	Use this object to enter and execute SQL statements against a known system database.
CRMRecord object	Represents records in a table.
CRMSelfService object	Provides access to the CRM database and to many CRM object methods, from outside the CRM application.
CRMTargetListField object	Use this objects to define fields to be included in a target list.
CRMTargetListFields object	Use this object to set up a list of **CRMTargetListField** objects.
CRMTargetLists object	Use this object to create and save a target list in conjunction with the **CRMTargetListField object** and **CRMTargetListFields object**.
Email object	Use this object to customize scripts deployed by E-mail Management. The Email object provides access to an email through its properties and methods.
MailAddress object	Use this object to customize scripts deployed by E-mail Management. The MailAddress object provides access to an individual address from the **AddressList object**.
MsgHandler object	Use this object customize scripts deployed by E-mail Management. It provides basic access to the **Email object** and functionality for the system.

AddressList object

Provides access to the lists of recipients (To, Cc, and Bcc). Allows you to customize scripts deployed by E-mail Management (see the System Administrator Guide or Help).

The AddressList object is returned by the **Email object properties** (**Recipients**, **CC**, and **BCC**).

- **AddressList methods**
- **AddressList properties**

AddressList methods

AddAddress(Address, Name). Adds an address to the list of recipients in an email.

AddAddress(Address, Name)

Adds an address to the list of recipients in an email.

Parameters

- **Address**. Specifies an email address to add.
- **Name**. Specifies the recipient name associated with the email address.

Examples

```
email.Recipients.AddAddress("john.doe@mydomain.com", "John Doe");
```

Adds the email address of John Doe to the To list.

AddressList properties

- **Items**. Returns email addresses and names in the form of a **MailAddress object**.
- **Count**. Returns the number of email addresses in the list.

Items

Returns email addresses and recipient names in the form of a **MailAddress object**.

Parameters

Integer. Specifies the index position of the email address to return.

Property value

Addresses and recipient names in the form of a **MailAddress object**.

Examples

```
var emailAddress;
emailAddress = email.Recipients.Items(1).Address;1
```

Returns the email address located at index position 1 in the To list and stores the address in the emailAddress variable.

Count

Returns the number of email addresses in the list.

Property value

Integer

Examples

```
if (email.Recipients.Count==1)
{
    // Insert code to perform action here.
}
```

Specifies to perform some action if the number of email addresses in the To list equals 1.

Attachment object

Provides access to message attachments. Allows you to customize scripts deployed by E-mail Management (see the System Administrator Guide or Help).

The Attachment object is returned by the **Attachments** property of the **Email object properties**. Use the **Items** property of the **AttachmentList object** to access the Attachment object.

- **Attachment methods**
- **Attachment properties**

Attachment methods

- **Save(Name, Path)**. Saves the attachment to a specified folder.
- **SaveAs(Name, Path)**. Saves the attachment as a file with the specified name.

Save(Name, Path)

Saves the attachment to a specified folder.

Parameters

- **Name**. Specifies the name of the file in which to save the attachment. The parameter is passed by reference and may be returned with a different value than the value sent to it.
- **Path**. Specifies the full path to the folder in which to save the attachment.

Return value

Boolean:

- **True**. Indicates that the save operation succeeded.
- **False**. Indicates that the save operation failed.

Examples

```
AttItem.Save("MyAttachment.txt", "C:\Attachments");
```

Saves the attachment stored in the AttItem variable to the MyAttachment.txt file located in the Attachments folder.

SaveAs(Name, Path)

Saves the attachment as a file with the specified name.

Parameters

- **Name**. Specifies the name of the file in which to save the attachment.
- **Path**. Specifies the full path to the location of the file.

Return value

Boolean:

- **True**. Indicates that the save operation succeeded.
- **False**. Indicates that the save operation failed.

Examples

```
AttItem.SaveAs("MyAttachment.txt", "C:\Attachments");
```

Saves the attachment stored in the AttItem variable to the MyAttachment.txt file located in the Attachments folder.

Attachment properties

- **Name**. Gets or sets the file name of the attachment.
- **Extension**. Gets the file name extension of the attachment.

Name

Gets or sets the file name of the attachment.

Property value

String (read/write)

Examples

```
var newname;
newname = AttItem.Name + "MyAttachment" + AttItem.Extension;
```

Adds the string *MyAttachment* as a suffix to the original file name of the attachment stored in the AttItem variable. Keeps the file name extension of the attachment. Stores the new file name of the attachment in the newname variable.

Extension

Gets the file name extension of the attachment.

Property value

String (read-only)

Examples

```
var newname;
newname = AttItem.Name + "MyAttachment" + AttItem.Extension;
```

Adds the string *MyAttachment* as a suffix to the original file name of the attachment stored in the `AttItem` variable. Keeps the file name extension of the attachment. Stores the new file name of the attachment in the newname variable.

AttachmentList object

Provides access to email attachments. Allows you to customize scripts deployed by E-mail Management (see the *System Administrator Help*).

- **AttachmentList properties**

AttachmentList properties

- **Count**. Returns the number of attachments.
- **Items**. Returns an attachment in the form of an **Attachment object**.
- **LibraryPath**. Specifies the path to the Sage CRM library.

Count

Returns the number of attachments.

Property value

Integer (read-only)

Examples

```
for (i = 0; i < Attachments.Count; i++)
{
    // Insert code to perform action here.
}
```

Specifies to perform some action on all attachments.

Items

Returns an attachment in the form of an **Attachment object**.

Property value

Integer. The index of the attachment.

Examples

```
for (i = 0; i < Attachments.Count; i++)
{
    AttItem = Attachments.Items(i);
    //  Insert code to perform action here.
}
```

Stores each attachment in the `AttItem` variable, and then performs the specified action on the attachment.

LibraryPath

Specifies the path to the Sage CRM library. This is the **Library** folder in the Sage CRM installation directory. The default location of the **Library** folder is %ProgramFiles%\Sage\CRM\CRM\Library.

Property value

String

Examples

```
var libdir = Attachments.LibraryPath + "\\" + CompanyQuery("comp_name");
```

CRM object

Provides access to Sage CRM objects and functionality. Exposes methods that allow you to create new objects, get existing objects, and execute objects.

Child objects: **CRMBase object** and **CRMSelfService object**.

In some legacy code (for example, in the include file), the CRM object may be instantiated as *eWare*. Existing code that refers to eWare blocks or the eWare object still works, and CRM and eWare are interchangeable (except in the context of CRMSelfService).

- **CRM methods**
- **CRM properties**

CRM methods

- **AddContent(Content)**. Adds the value in its parameter to the page in memory and returns that value when the **GetPage()** method is called.
- **CreateQueryObj(SQL, Database)**. Creates a new query object from the system database or an external database to which Sage CRM is connected.
- **CreateRecord(TableName)**. Creates a new record object in a specified database table.
- **FindRecord(TableName, QueryString)**. Finds and retrieves a record object from a specified database table.
- **Form(ElementName, Options)**. Retrieves the values of form elements posted to the HTTP request body. Allows you to filter the values to be returned.
- **GetBlock(BlockName)**. Retrieves a Sage CRM block.
- **GetCustomEntityTopFrame(Entity)**. Retrieves the top content (the icon, caption, and description) for a custom entity.
- **GetPage()**. Returns the page contents that have been previously added by the **AddContent** method.
- **GetTrans(Family, Caption)**. Returns the translation for a caption in the specified caption family, based on user's current language.
- **QueryString(ParameterName, Options)**. Retrieves the values of the parameters in the HTTP query string. Allows you to filter the values to be returned.
- **RefreshMetaData(Family)**. Updates the Sage CRM internal cache with new information.
- **SetContext(EntityName, EntityID)**. Updates the recent list for the specified custom entity.

AddContent(Content)

Adds the value in its parameter to the page in memory and returns that value when the **GetPage()** method is called. You can use this value in scripts to pass something that is only available on the server side, such as the current user's email address.

In Sage CRM version 7.2b and later, all ASP pages must use the **AddContent()** and **GetPage()** methods to build the HTML for the page. This is required to ensure the correct rendering of the page structure and links, including the left-hand main menu, horizontal tabs, and top content.

Parameters

Content. Specifies the string value returned by the **Execute(Arg)** method.

Examples

```
var MyList;
MyList = CRM.GetBlock('PersonGrid');
CRM.AddContent(MyList.Execute("pers_lastname like 'B%'"));
Response.Write(CRM.GetPage());
```

```
CRM.AddContent('<' + 'script>var defaultemailaddress=\'' + CRM.UserOption('defaultemailaddress')
+ '\'<' + '/script>');
```

CreateQueryObj(SQL, Database)

Creates a new query object from the system database or an external database to which Sage CRM is connected.

This method can return different data than the **FindRecord(TableName, QueryString)** method for the following reasons.
The CreateQueryObject method creates a new connection, uses a transaction that is different from the one used for update, and thus can only work with data that is actually in the database. The commit takes place after table-level scripts are run.
The **FindRecord(TableName, QueryString)** method, however, uses the dispatch connection, so it's the same transaction as the one used by the update. Therefore, this method can work with uncommitted data.

Parameters

- **SQL**. Specifies a valid SQL string.
- **Database**. Specifies the database to use. When this parameter is omitted, the system database is used by default.

Examples

```
var Query;
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());
```

Creates a query object from the company view by using the system database.

CreateRecord(TableName)

Creates a new record object in a specified database table. To save the record object, you must call the **SaveChanges()** method. The **SaveChanges()** method is automatically called by the **Execute (Arg)** method of most blocks when the mode is set to Save.

Parameters

TableName. Specifies the name of the table where the record is to be created. This can be the Sage CRM system database or an external database to which Sage CRM is connected.

Examples

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

<%
var Comp;
var block;
Comp = CRM.CreateRecord('company');
Comp.item('comp_Name') = 'My company';
Comp.SaveChanges();
block = CRM.GetBlock("companygrid");
CRM.AddContent(block.Execute(''));
Response.Write(CRM.GetPage());
%>
```

Creates a new company record.

FindRecord(TableName, QueryString)

Finds and retrieves a record object from a specified database table.

Parameters

- **TableName**. Specifies the database table from which you want to retrieve a record object.
- **QueryString**. Specifies the SQL `WHERE` clause that identifies the record object.

Example

```
var ThisPersonId;
var PersonBlock;
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());
```

Displays a record summary for the current person.

Form(ElementName, Options)

Retrieves the values of form elements posted to the HTTP request body. Allows you to filter the values to be returned. Provides an additional level of security in Sage CRM customizations.

Parameters

- **ElementName**. Specifies the name of the form element to retrieve.
- **Options**. Allows you to filter the form element values to retrieve. Can take one or several of the following values:
  - **IntegersOnly**. Returns only integers from the form element value. This value is useful if you want to get a record ID.
  - **WordOnly**. Returns only the following characters from the form element value: A-Z, a-z, 0-9, underscore (_). This value is useful if you're expecting a single value to be returned.
  - **SQLInjection**. Applies a SQL injection filter to protect against SQL injection attacks. If an injection attempt is identified, a blank value is returned.
  - **XSS**. Applies a cross-site scripting filter to protect against XSS attacks. Returns only content that passes the filter. This value is useful if you expect plain text to be returned.

  When using several **Options** values, use a comma as a separator.
  When the **Options** parameter is omitted, **SQLInjection** and **XSS** are used by default.

Return value

String

Examples

```
var MyElement = CRM.Form('MyElement', 'SQLInjection, XSS')
```

Applies both the SQL injection and cross-site scripting filters. This line of code performs the same action as `var MyElement = CRM.Form('MyElement')`

GetBlock(BlockName)

Retrieves a block. Use this method to call child blocks of the CRM object. For more information, see **Blocks**.

Parameters

BlockName. Specifies the name of the block, existing screen or list in Sage CRM to retrieve.

Examples

```
var Marquee;
Marquee = CRM.GetBlock("marquee");
```

Retrieves a new marquee block and stores it in the Marquee variable.

```
var Search;
var List;
var Container;
Search = CRM.GetBlock("CompanySearchBox");
List = CRM.GetBlock("CompanyGrid");
Container = CRM.GetBlock("Container");
Container.AddBlock(Search);
Container.AddBlock(List);
CRM.AddContent(Container.Execute());
Response.Write(CRM.GetPage());
```

Gets a Container, Search (CompanySearchBox), and List (CompanyGrid) block. Adds the search screen and list to the container and displays the container. You can configure the **Execute** method to use the search results as the argument for the list.

GetCustomEntityTopFrame(Entity)

Retrieves the top content (the icon, caption, and description) for a custom entity.

Parameters

Entity. Specifies the name of the entity for which to retrieve the top content.

Examples

```
CRM.GetCustomEntityTopFrame("Lease");
```

Retrieves the top content for the custom entity named Lease.

GetPage()

Returns the page contents that have been previously added by the `AddContent` method. The contents are returned in the format specified by the current device.

In Sage CRM version 7.2b and later, all ASP pages must use the **AddContent()** and **GetPage()** methods to build the HTML for the page. This is required to ensure the correct rendering of the page structure and links, including the left-hand main menu, horizontal tabs, and top content.

Parameters

TabGroupName. Optional. Specifies the tab group name that includes the tabs to be passed to the method. In this case, the method shows the passed tabs instead of the current default tabs.

Examples

```
<!-- #include file = "sagecrm.js"-->
<%
var Comp;
var block;
Comp = CRM.CreateRecord('company');
Comp.item('comp_Name') = 'My company';
Comp.SaveChanges();
block = CRM.GetBlock("companygrid");
CRM.AddContent(block.Execute(''));
Response.Write(CRM.GetPage());
%>
```

Creates a new Company record.

GetTrans(Family, Caption)

Returns the translation for a caption in the specified caption family, based on user's current language.

Self-Service users can set the language using the `VisitorInfo` method, for example:

```
CRM.VisitorInfo("Visi_Language")='DE';
```

Parameters

- **Family**. Specifies the name of the caption family to which the caption belongs. To view available caption families and caption types, go to **<My Profile> | Administration | Customization | Translations**.
- **Caption**. Specifies the name of the caption.

Examples

```
CRM.AddContent(CRM.GetTrans('GenCaptions','Screen'));
Response.Write(CRM.GetPage());
```

Displays the word Screen in the user's current language, provided that the translation in that language has already been added to Sage CRM translations.

QueryString(ParameterName, Options)

Retrieves the values of the parameters in the HTTP query string. Allows you to filter the values to be returned. Provides an additional level of security in Sage CRM customizations.

Parameters

- **ParameterName**. Specifies the name of the parameter to retrieve.
- **Options**. Allows you to filter the parameter values to retrieve. Can take one or several of the following values:
  - **IntegersOnly**. Returns only integers from the parameter value. This value is useful if you want to get a record ID.
  - **WordOnly**. Returns only the following characters from the parameter value: A-Z, a-z, 0-9, underscore (_). This value is useful if you're expecting a single value to be returned.
  - **SQLInjection**. Applies a SQL injection filter to protect against SQL injection attacks. If an injection attempt is identified, a blank value is returned.
  - **XSS**. Applies a cross-site scripting filter to protect against XSS attacks. Returns only content that passes the filter. This value is useful if you expect plain text to be returned.

  When using several **Options** values, use a comma as a separator.
  When the **Options** parameter is omitted, **SQLInjection** and **XSS** are used by default.

Return value

String

Examples

```
var myParam = CRM.QueryString('MyParameter', 'SQLInjection, XSS')
```

Applies both the SQL injection and cross-site scripting filters. This line of code performs the same action as `var myParam = CRM.QueryString('MyParameter')`

RefreshMetaData(Family)

Updates the Sage CRM internal cache with new information. For example. when the custom_ captions table was edited or new records were added.

Parameters

Family. Specifies the caption family to update (capt_family).

Examples

```
var NewCaption;
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");
```

Adds a new entry named Open to the Comp_Status caption family.

The **CreateRecord(TableName)** method adds a new record to the Custom_Captions table. Then, data is entered into the relevant table fields and the **SaveChanges()** method saves the changes to the database. The RefreshMetaData(Family) method refreshes the Sage CRM cache.

SetContext(EntityName, EntityID)

Updates the recent list for the specified custom entity.

Parameters

- **EntityName**. Specifies the name of the custom entity
- **EntityId**. Specifies the custom entity ID as it appears in custom_tables.

Examples

```
CRM.SetContext(EntityName,Id);
```

CRM properties

Mode. Sets a mode for a block.

Mode

Sets a mode for a block. The values this property can take are defined in the include file.

Take care when changing modes: you may unintentionally lock down a screen if the conditions you specify are not precise.

Property values

- **0** or **View**. Sets the mode to View.
- **1** or **Edit**. Sets the mode to Edit.
- **2** or **Save**. Sets the mode to Save.

Examples

```
if (CRM.Mode < Edit) {
  CRM.Mode = Edit;
}
```

Changes the mode to Edit if the current mode is View.

```
CRM.Mode = Edit; if (error != "") {
  CRM.Mode = Save;
  CRM.AddContent(error);
  Response.Write(CRM.GetPage());
}
```

Changes the mode to Save if an error has occurred. Also displays the error.

CRMBase object

Sets up the context information for the current view and displays the tabs for that view. This object contains all Sage CRM custom metadata.

- **CRMBase methods**
- **CRMBase properties**

CRMBase methods

- **Button**. Returns the text, image, and link for a Sage CRM button.
- **GetContextInfo(Entity, FieldName)**. Returns the named field from a table based on the current context.
- **GetTabs(TabGroup)**. Returns the specified group of tabs.
- **Logon(LogonId, Password)**. Allows you to log on to Sage CRM from a command prompt.
- **Url(Action)**. Transforms a URL to the format required by Sage CRM.
- **ConvertValue(Avalue, AfromCurr, AToCurr)**. Converts a value from one currency to another.

Button

Returns the text, image, and link for a Sage CRM button. These buttons are typically the generic buttons that appear on screens and containers. For example, you can use this method to add buttons that open web sites or ASP pages.

Parameters

- **Caption**. Specifies the caption for the button. The caption is translated based on the user's language, provided that a matching translation exists for the caption.
- **ImageName**. Specifies the image to display on the button. The image must be stored in the Img folder located in the Sage CRM installation directory.
- **URL**. Specifies the URL to link the button to. This can be a web address or a custom page located in the CustomPages folder in the Sage CRM installation directory.
- **PermissionsEntity**. Specifies the entity name. Allows you to add the button based on a users security profile for an entity.
- **PermissionsType**. Specifies permissions to apply. This can be VIEW, EDIT, DELETE, or INSERT, depending on the action the button performs. Allows you to add the button based

on a users security profile for an entity.

- **Target**. Sets the TARGET property of the button's anchor.

Examples

```
CRM.AddContent(CRM.Button("My button","MyImage.gif", CRM.Url("MyPage.asp")));
Response.Write(CRM.GetPage());
```

Displays a button named **My button**. This button contains an image stored in the MyImage.gif file and is linked to the MyPage.asp file.

GetContextInfo(Entity, FieldName)

Returns the named field from a table based on the current context. Use this method to return the RecordID for a table which you can use to build an SQL query. For example, to create charts.

Parameters

- **Entity**. Specifies the entity from which to return the field. This can be: Person, Company, Opportunity, Lead, Case, Solution, Channels (teams), Campaigns, Waves, Wave Items, SelectedUser (applicable when viewing the My CRM list), User (currently logged on user).
- **FieldName**. Specifies the name of the field to be returned.

Examples

```
GetContextInfo("case", "case_description");
```

Returns description for the case that the user is currently viewing.

```
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());
```

Displays cases that are currently in progress for the current company context.

This example obtains the unique company ID. Company ID is used in a SQL statement, which selects all cases that match the company's ID and have an In Progress status.

The **GetBlock(BlockName)** method returns a **CRMListBlock object** of type CaseList, which is stored in the CaseListBlock variable .

The SELECT statement used to extract the required cases is passed to the CaseListBlock's Execute function. The populated CaseListBlock is then passed as an argument to the **AddContent(Content)** method to store the page in memory.

Response.Write outputs the generated HTML to the screen.

GetTabs(TabGroup)

Returns the specified group of tabs. You can use this method to specify a new tab group that you want displayed from a menu option.

Only invoke this method if you are using an old include file (for example, CRM.js). Include this method at the top of your ASP file and after the CRM.js include file. This method is not needed if you use the SAGECRM.js or ACCPACCRM.js include file.

Parameters

TabGroup. Specifies the name of the group of tabs to return.

Examples

```
<% CRM.GetTabs("NewTabGroupName") %>
```

Adds a new menu button that displays a new tab group. The button links to an ASP that includes the GetTabs method.

```
<% CRM.GetTabs() %>
```

Displays tabs for the current entity.

```
<% Response.Write(CRM.GetTabs()) %>
```

Displays tabs for the current entity.

Logon(LogonId, Password)

Allows you to log on to Sage CRM from a command prompt.

To use this method, make sure to set the **External Logon Allowed** option for the relevant user to true in **<My Profile> | Administration | Users | Users**.

Do not use this method in ASP or .NET. When this method is used, no metadata is available.

Parameters

- **LogonId**. Specifies the user name to log on with.
- **Password**. Specifies the password that matches the user name.

Return value

- **<Blank string>**. Indicates that logon has succeeded.
- **<Error code>**. Indicates that logon has failed.

Examples

```
var CRM = new ActiveXObject('CRM.< Installation Folder>');
CRM.Logon("<user name>", "<password>");
```

Allows you to start using the Sage CRM object under a specified user account. Place this code in an external JavaScript page. You can copy and paste your encrypted password from the system database. It is a good practice to create a dedicated user account for external access to the system with limited access rights.

Url(Action)

Transforms a URL to the format required by Sage CRM. You can use the returned URL to create a link in Sage CRM.

Parameters

Action. Specifies a URL, ASP file, or .NET DLL file and method. If it's 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.

Examples

```
CRM.AddContent(CRM.Button("Chart","Cancel.gif",CRM.Url("system/InvChart.asp")));
Response.Write(CRM.GetPage());
```

Displays a button that links to an ASP page.

```
<a href='*<%=CRM.Url("http://www.mydomain.com")%>'>Click here to open the web site</a>
```

Creates an anchor that links to the specified web site.

```
myContainer.AddButton(CRM.Button(“Add”,”new.gif”,CRM.Url(“QuickLook.dll-RunQuickLook”)));
```

Creates a link that references the RunQuickLook base method of a .NET DLL(QuickLook.dll).

ConvertValue(Avalue, AfromCurr, AToCurr)

Converts a value from one currency to another.

Parameters

- **AValue**. Specifies the value to be converted.
- **AFromCurr**. Specifies the identifier of the currency to convert from. The currency must exist in the Curr_CurrencyID field of the Currency table, otherwise an error is returned.
- **AToCurr**. Specifies the identifier of the currency to convert to. The currency must exist in the Curr_CurrencyID field of the Currency table, otherwise an error is returned.

Return value

String containing the converted value formatted to the number of decimals specified for the currency in the AToCurr parameter.

Examples

```
var iValue;
var iFromCurr;
var iToCurr;
iValue = 50, 000;
iFromCurr = 1;
// Where 1 is the identifier of Euro.
iToCurr = 2;
// Where 2 is the identifier of British Pound.
CRM.AddContent("British Pound: " + CRM.ConvertValue(iValue, iFromCurr, iToCurr));
Response.Write(CRM.GetPage());
```

Converts 50,000 from Euro to British Pound.

CRMBase properties

- **FastLogon**. Disables the loading of metadata cache when a user logs on externally.
- **TargetLists**. Retrieves a **CRMTargetLists object**.

FastLogon

Disables the loading of metadata cache when a user logs on externally. Use this property with the **Logon(LogonId, Password)** method.

Property values

- **1**(default). Off.
- **2**. Low.
- **3**. High.

Examples

```
var CRM = new ActiveXObject('CRM.<CRMinstalldir>');
CRM.FastLogon = 1;
CRM.Logon("administrator", "P@ssw0rd");
```

Disables the loading of metadata cache and logs on the administrator account.

TargetLists

Retrieves a **CRMTargetLists object**. For examples, see **CRMTargetLists object**.

CRMBlock object

Is the base for all CRM blocks. The specific block type called by the CRM object determines the actual implementation of each CRMBlock methods and properties.

Preceding code:

```
var block = CRM.GetBlock("myblock");
```

- **CRMBlock methods**
- **CRMBlock properties**

CRMBlock methods

- **Execute(Arg)**. Executes the Sage CRM block and returns the display contents of the block.
- **Validate()**. Validates data entries.

Execute(Arg)

Executes the Sage CRM block and returns the display contents of the block.

Parameters

Arg (optional). Specifies any value that relates to the block type.

Example

```
var list = CRM.GetBlock("personlist");
CRM.AddContent(list.Execute());
Response.Write(CRM.GetPage());
```

Executes the personlist block.

Validate()

Validates data entries. For example, this method checks to see if a required field is filled in. This method is normally used in if statements.

Examples

```
if ((CRM.Mode==Save) && (!block.Validate()))
    {
        error="<font color=red><b>Please correct the highlighted entries</b></font>";
        CRM.Mode=Edit;
    }
```

Validates the relevant fields and displays an error message if validation has failed.

CRMBlock properties

- **ArgObj**. Provides an alternative way to pass parameters to a block that is a container.
- **CheckLocks**. Specifies whether to check if a record is in use before allowing the record to be edited.
- **DisplayForm**. Specifies whether the block wraps itself in a form element.
- **FormAction**. Sets the action that the form takes.
- **Height**. Sets the block position in pixels or percent from the top of the screen.
- **Name**. Sets or gets the name of the current block.
- **NewLine**. Specifies whether the block appears on a new line.
- **ShowValidationErrors**. Sets whether to display validation errors when a user incorrectly enters information in an entry box.
- **Title**. Sets the block title.
- **ViewName**. Specifies the name of the view on which the list is based.
- **Width**. Sets the block width in pixels or as a percentage of screen.
- **Mode**. Sets a mode for the ASP page.

ArgObj

Provides an alternative way to pass parameters to a block that is a container.

Examples

```
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);
```

Passes the SearchBlock result as the argument to the list block and displays the relevant search or results list.

CheckLocks

Specifies whether to check if a record is in use before allowing the record to be edited. If the record is in use, displays an error message. Only applicable on EntryGroup or Container blocks.

Property values

- **true** (default). Specifies to check if a record is in use.
- **false**. Disables the check.

The property values are case-sensitive.

Examples

```
Block = CRM.GetBlock("companyboxlong");
Block.CheckLocks = false;
```

Specifies not to check if record is in use.

DisplayForm

Specifies whether the block wraps itself in a form element.

Property values

- **true** (default). Specifies that the block wraps itself.
- **false**. Specifies that the block does not wrap itself and data is not saved in the form.

Examples

```
block = CRM.GetBlock("companyboxlong");
block.DisplayForm = true;
CRM.AddContent(block.execute());
Response.Write(CRM.GetPage());
```

Wraps the companyboxlong block in a form element, which means it follows the normal save/change steps (such as performing validation tasks).

FormAction

Sets the action that the form takes. You can use this property only if the **DisplayForm** property is set to true. By default, the form action is blank, which causes a submit to return to the same page.

Property parameters

None

Examples

```
block.FormAction = "mypage.asp";
```

Height

Sets the block position in pixels or percent from the top of the screen. The value of this property determines how far the block appears from the top of the screen.

Property parameters

None

Examples

```
var block = CRM.GetBlock("companyboxlong");
block.Height="150";
CRM.AddContent(block.Execute());
Response.Write(CRM.GetPage());
```

Sets the block to be 150 pixels from the top of the screen.

Name

Sets or gets the name of the current block.

Property parameters

Name. String. Specifies the new name for the block instance.

Examples

```
var block=CRM.GetBlock("entry");
block.Name="My New Block";
CRM.AddContent(block.Name);
Response.Write(CRM.GetPage());
```

Sets the block and returns the block name.

NewLine

Specifies whether the block appears on a new line. This property is only applicable to **CRMEntryBlock object** or **CRMEntryGroupBlock object** in a container.

This is the same as selecting **New Line** in the **Position** property from **<My Profile> | Administration | Customization | <Entity> | Blocks**.

Property values

- **true** (default). Specifies that the block appears on a new line.
- **false**. Specifies that the block doesn't appear on a new line.

Examples

```
var group = CRM.Getblock("container");
var block = CRM.GetBlock("companyboxlong");
group.AddBlock(block);
var block2 = CRM.GetBlock("personboxshort");
block2.NewLine=false;group.AddBlock(block2);
CRM.AddContent(Group.Execute());
Response.Write(CRM.GetPage());
```

Inserts two **Entry** blocks into a container side-by-side.

ShowValidationErrors

Sets whether to display validation errors when a user incorrectly enters information in an entry box.

Property values

- **true** (default). Specifies to display validation errors.
- **false**. Specifies not to display validation errors.

Examples

```
var MyBlock;
MyBlock = CRM.GetBlock("entrygroup");
MyBlock.ShowValidationErrors = false;
```

Specifies not to display validation errors.

Title

Sets the block title.

This is the same as setting the **Title** field value from **<My Profile> | Administration | Customization | <Entity> | Blocks**.

Examples

```
var block = CRM.GetBlock("companyboxlong");
block.Title="My Block";
CRM.AddContent(block.Execute());
Response.Write(CRM.GetPage());
```

Sets the company summary block title.

ViewName

Specifies the name of the view on which the list is based. Only use this property with **CRMListBlock object**.

Examples

```
var NewList = CRM.GetBlock("PersonList");
NewList.ViewName="vListCommunication";
NewList.AddGridCol("Pers_FullName");
CRM.AddContent(NewList.Execute(""));
Response.Write(CRM.GetPage());
```

Creates a new list based on the vListCommunication view.

Width

Sets the block width in pixels or as a percentage of screen.

Examples

```
var block = CRM.GetBlock("companyboxlong");
block.Width="40%";
CRM.AddContent(block.Execute());
Response.Write(CRM.GetPage());
```

Sets the width of the company summary box to 40%.

Mode

Sets a mode for the ASP page. This allows you to control what happens to certain blocks when they are executed.

Property values

- **0**. Sets the mode to View.
- **1**. Sets the mode to Edit.
- **2**. Sets the mode to Save.
- **3**. Sets the mode to PreDelete.
- **4**. Sets the mode to PostDelete.

Constants are declared for these values in the Sage CRM include files.

Examples

```
var Record = CRM.CreateRecord("Case");
var EntryGroup = CRM.GetBlock("CaseDetailBox");
if (CRM.Mode == 0)
    {
        CRM.Mode = 1;
    }
CRM.AddContent(EntryGroup.Execute(Record));
Response.Write(CRM.GetPage());
```

If the CRM.**Mode** of the **EntryGroup** block is set to View (0), this example changes the mode to Edit (1) and displays the block in that mode.

CRMChartGraphicBlock object

Use this object to draw charts and graphs.

The ASP page author controls the type of chart and associated parameters. In addition, the ASP author chooses the database query to use for the chart or graph. The CRMChartGraphicBlock object inherits all GraphicBlock capabilities and adds the ability to generate a variety of charts.

Syntax to initiate the CRMChartGraphicBlock object:

```
ChartGraph = CRM.GetBlock('chart');
```

- **CRMChartGraphicBlock methods**
- **CRMChartGraphicBlock properties**

CRMChartGraphicBlock methods

- **BackGradient(Visible, StartColor, EndColor)**. Applies a gradient to the background of a chart.
- **ChartTitle(text)**. Sets a title for the chart.
- **ManualChartEntry(Value, MakeNull=true/false)**. Creates a chart where the data is not contained in a Sage CRM table.
- **ShowLegend(true/false)**. Determines whether to show or hide the legend for the chart.
- **Style(Stylename)**. Sets the style of the chart.

BackGradient(Visible, StartColor, EndColor)

Applies a gradient to the background of a chart.

Parameters

- **Visible**. Specifies whether the background gradient is visible.
  Can take one of the following values:
  - **True**
  - **False**
- **StartColor**. Specifies the name of start color to be used for the background gradient. This parameter accepts a WideString value.
- **EndColor**. Specifies the name of end color to be used for the background gradient. This parameter accepts a WideString value.

Examples

```
ChartGraph.BackGradient(true,'Blue','White');
```

Sets a blue gradient that fades to white.

ChartTitle(text)

Sets a title for the chart. If no chart title is set, the title is removed allowing more room for the chart.

Parameters

Text. Sets a title for the chart. This parameter accepts a WideString value.

Examples

```
ChartGraph.ChartTitle('Case Priority');
```

Sets the chart title to "Case Priority".

ManualChartEntry(Value, MakeNull=true/false)

Creates a chart where the data is not contained in a Sage 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).

Parameters

- **Value**. Sets a value for a chart entry. This parameter accepts a WideString value.
- **MakeNull**. Defines whether the corresponding chart entry is blank. Accepts one of the following values:
    - **True**
    - **False**

Examples

```
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)

Determines whether to show or hide the legend for the chart.

Parameters

Boolean. Sets whether or not to show the chart legend. Can take one of the following values:

- **True**. Specifies to show the legend.
- **False**. Specifies to hide the legend.

Examples

```
ChartGraph.ShowLegend(true);
```

Shows the chart legend.

Style(Stylename)

Sets the style of the chart.

Parameters

Stylename. Specifies the name of the chart style to be used. This parameter accepts one of the following text values:

- **Bar** (default). 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

Examples

```
ChartGraph.Style('Pie');
```

Sets the chart style to Pie.

CRMChartGraphicBlock properties

- **LabelX**. Sets the text label for the X-axis (horizontal) of a chart.
- **LabelY**. Sets the text label for the Y-axis (vertical) of a chart.
- **SQLText=Text**. Uses a SQL query to retrieve and assign values to the **LabelX**, **LabelY**, and **XLProp=text** parameters from the specified database table. This only occurs if no values were set for the mentioned parameters.
- **XLProp=text**. Specifies values to be displayed on the X-axis.
- **Xprop=text**. Sets the field name to be used along the X-axis.
- **Yprop=text**. Sets the field name to be used along the Y-axis.

LabelX

Sets the text label for the X-axis (horizontal) of a chart.

Values

Text. Specifies the text label for the X-axis. This parameter accepts a WideString value.

Examples

```
ChartGraph.LabelX = 'Date';
```

Sets the text label of the X-axis to "Date".

LabelY

Sets the text label for the Y-axis (vertical) of a chart.

Values

Text. Specifies the text label for the Y-axis. This parameter accepts a WideString value.

Examples

```
ChartGraph.LabelY = 'Certainty, %';
```

Sets the text label of the Y-axis to "Certainty, %".

SQLText=Text

Uses a SQL query to retrieve and assign values to the **LabelX**, **LabelY**, and **XLProp=text** parameters from the specified database table. This only occurs if no values were set for the mentioned parameters.

SQL query goes through the database table and uses the first field values that satisfy the specified criteria as values for the parameters.

for the chart if X,Y, or XL labels haven't been set. Sage CRM navigates through the fields in the table as defined in the SQL query and uses the first fields it finds and is able to use.

Values

Text. Specifies the SQL query to be used for retrieving and assigning values for the parameters. This must be a WideString value.

Examples

```
Chart = CRM.GetBlock('chart'):
Chart.SQLText = 'Select * from OpportunityProgress Where '+
'Oppo_OpportunityId='+OppId;
```

XLProp=text

Specifies values to be displayed on the X-axis.

Values

Text. Specifies the field name that stores the text values to be displayed on the X-axis. Accepts a WideString value.

Examples

```
ChartGraph.XLProp = 'Fld_Date';
```

Uses values stored in the Fld_Date field as values for the X-axis.

Xprop=text

Sets the field name to be used along the X-axis.

Values

Text. Specifies the field name. This must be a WideString value.

Example

```
ChartGraph.Xprop = 'Fld_Date';
```

Yprop=text

Sets the field name to be used along the Y-axis.

Values

Text. Specifies the field name. This must be a WideString value.

Example

```
ChartGraph.Yprop = 'Fld_Date';
```

CRMContainerBlock object

Use the CRMContainerBlock object to group blocks on a screen. It acts as a wrapper for other blocks. You can nest CRMContainerBlock inside other containers. An example of a container block is a linked search panel and related list.

A container block has standard Sage CRM buttons and any number of extra buttons. If any blocks with buttons are included, the buttons are shown only once on the container block and then applied to all the internal blocks. The standard CRM buttons are:

- **Change** or **Save**. Displayed as **Change** when the screen is in View mode and **Save** when the screen is Edit mode. This button is shown by default.
- **Delete** or **Confirm Delete**. Displayed as **Delete** when the screen is in View mode and **Confirm Delete** when the screen is in Confirm Delete mode. This button is not shown by default.
- **Continue** or **Cancel**. Displayed as **Continue** when the screen is in View mode and **Cancel** when the screen is in Edit or Confirm Delete mode. This button is not shown by default.

The Execute function on a block takes only one argument. When CRMContainerBlock is executed, it passes its argument to all its item blocks as they are executed. If item blocks in a container block require different arguments for their Execute functions, set the ArgObj property on each item block and don't pass any argument to the container.

Syntax to create a container with two blocks:

```
// Create a container.
Container = CRM.GetBlock("container");
// Get two screens.
Screen1 = CRM.GetBlock("Screen1");
Screen2 = CRM.GetBlock("Screen2");
// Add the screens 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());
```

- **CRMContainerBlock methods**
- **CRMContainerBlock properties**
- **Code example: ShowWorkflowButtons property**
- **Code example: WorkflowTable and ShowNewWorkflowButtons properties**

CRMContainerBlock methods

- **AddBlock(Block)**. Adds a block to a container.
- **AddButton(ButtonString)**. Adds a button to the Container block.
- **DeleteBlock(BlockName)**. Deletes the block from the container.
- **GetBlock(BlockName)**. Returns a pointer to the specified block object that exists within the container.

AddBlock(Block)

Adds a block to a container.

Parameters

Block. Specifies a reference to the block to be added. This can be any block previously retrieved by using the **GetBlock(BlockName)** method.

Examples

```
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());
```

Creates a container block and adds two blocks to it.

AddButton(ButtonString)

Adds a button to the Container block.

The button string must be HTML code. This HTML code is added after the other buttons are drawn. The easiest way to get the HTML for the button is to use the **Button** method.

Parameters

ButtonString. Specifies HTML code to render the button. This should be a link within a <table> tag in the ASP page. This parameter accepts a string value.

Examples

```
R = CRM.FindRecord('Company','Comp_CompanyId=1');
Holder = CRM.GetBlock('companyboxlong');
Holder.AddButton(CRM.Button("Try This","new.gif",CRM.Url("AnotherPage.asp")));
CRM.AddContent(Holder.Execute(R));
Response.Write(CRM.GetPage());
```

Adds a button named **Try This** to the company summary block.

DeleteBlock(BlockName)

Deletes the block from the container.

Parameters

BlockName. Specifies the name of the block to be deleted. This parameter accepts a string value.

Examples

```
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());
```

Deletes the AddressBoxShort block for non-administrators.

The AddressBoxShort block is one of the blocks in the CompanySummaryBlock container that has been set up in the **<My Profile> | Administration | Customization | Company | Blocks** area of Sage CRM.

GetBlock(BlockName)

Returns a pointer to the specified block object that exists within the container.

Parameters

BlockName. Specifies the name of the block to return. This parameter accepts a string value.

Examples

```
MyCustomContainer = CRM.GetBlock("MyCustomContainer");
R = CRM.FindRecord('Company','Comp_CompanyId=30');
MyE = MyCustomContainer.GetBlock("CompanyBoxShort");
CRM.AddContent(MyE.Execute(R));
Response.Write(CRM.GetPage());
```

Displays the CompanyBoxShort block in the MyCustomContainer container block. This container block has been set up in the **<My Profile> | Administration | Customization | Company | Blocks** area of Sage CRM.

CRMContainerBlock properties

- **ButtonAlignment**. Adjusts the alignment of the buttons on the screen.
- **ButtonImage**. Specifies the image file that contains an icon to be displayed on the standard buttons.
- **ButtonLocation**. Sets the location of the buttons in the container.
- **ButtonTitle**. Overrides the default text labels on the standard buttons.
- **DisplayButton**. Shows or hides the standard buttons.
- **Workflow properties**. Use Workflow properties to include the same button types in an ASP page for any table in the system database, including new custom tables that you've added for a customer.

ButtonAlignment

Adjusts the alignment of the buttons on the screen.

Values

Possible values of this property are defined in the **SageCrmNoLang.js** include file and by default are as follows:

- **0**. Bottom.
- **1**. Left.
- **2**. Right.
- **3**. Top.

The values this parameter can take depend on the value set in the **ButtonLocation** parameter.

If the **ButtonLocation** parameter is set to **Top** or **Bottom**, the ButtonAlignment parameter can only be set to Left (**1**) or Right (**2**).

If the **ButtonLocation** parameter is set to **Left** or **Right**, the ButtonAlignment parameter can only be set to Top (**3**) or Bottom (**0**).

Example

```
Container = CRM.GetBlock("container");
Container.ButtonLocation = Top;
Container.ButtonAlignment = 1;
CRM.AddContent(Container.Execute());
Response.Write(CRM.GetPage());
```

Aligns the buttons to the left of the screen.

ButtonImage

Specifies the image file that contains an icon to be displayed on the standard **Change** or **Save** button. Use this property to override the default image file for the buttons.

Values

The image file you specify must be stored in the following location:

<Sage CRM installation folder>\WWWRoot\Img\Buttons

If the image file you want to use is stored in a different location, specify the full path to the file.

Example

```
Container = CRM.GetBlock("container");
Container.DisplayButton(Button_Default) = true;
Container.ButtonTitle = "My Button Title";
CRM.AddContent(Container.Execute());
Response.Write(CRM.GetPage());
```

Sets the image and text to be displayed on the standard button.

ButtonLocation

Sets the location of the buttons in the container.

Values

Possible values of this property are defined in the **SageCrmNoLang.js** include file and by default are as follows:

- **Bottom**
- **Left**
- **Right** (default)
- **Top**

If this property is set to **Top** or **Bottom**, the buttons are shown in a horizontal line. Otherwise, they are shown in a vertical line.

Examples

```
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());
```

Displays the standard buttons at the top of the container.

ButtonTitle

Overrides the default text labels on the standard buttons, such as **Change** and **Save** (Button_ Default).

Value

Text. Specifies the text label to be displayed on the buttons.

Example

```
Container = CRM.GetBlock("container");
Container.DisplayButton(Button_Default) = true;
Container.ButtonTitle = "My Button";
CRM.AddContent(Container.Execute());
Response.Write(CRM.GetPage());
```

Sets the text label displayed on the standard buttons to **My Button**.

DisplayButton

Shows or hides the standard buttons.

Parameters

- **(Button_Default)**. Specifies whether to show or hide the **Change** and **Save** buttons. Can take one of the following values:
  - **true**. Shows the buttons.
  - **false**. Hides the buttons.
- **(Button_Delete)**. Specifies whether to show or hide the **Delete** button. Can take one of the following values:
  - **true**. Shows the button.
  - **false**. Hides the button.
- **(Button_Continue)**. Specifies whether to show or hide the **Continue** button. Can take one of the following values:
  - **true**. Shows the button.
  - **false**. Hides the button.

The standard buttons are defined in the **SageCrmNoLang.js** include file.

Examples

```
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());
```

Shows all standard buttons (**Change**, **Save**, **Delete**, and **Continue**).

Workflow properties

Use Workflow properties to include the same button types in an ASP page for any table in the system database, including new custom tables that you've added for a customer. The Container block has three properties that enable Workflow functionality:

- **WorkflowTable**. Sets the table for the **ShowNewWorkflowButtons** property to use.
- **ShowWorkflowButtons**. Displays or hides the workflow buttons on a view or edit screen for a record.
- **ShowNewWorkflowButtons**. Displays or hides a **New** button on a screen that shows a list of records.

Use these properties on tables for which you've configured workflow rules and states and want to display the rules and states as workflow buttons. For example, if you've enabled workflow 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, workflow buttons applicable to that case are displayed.

To use Workflow properties on a new custom CRM table, the table connection must have 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, enter the name of your workflow ID field on the Table Details screen. For more information, see **Database**.
- Configure the workflow rules, states, and tree for the table in **<My Profile> | Administration | Advanced Customization | Workflow**. For more information, see the System Administrator Guide or Help.

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.
  The .asp file specified in the Custom File Name property must:
  - Use a Container block (such as 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.
- 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.

WorkflowTable

Sets the table for the **ShowNewWorkflowButtons** property to use.

Parameters

TableName. Specifies the name of the table.

Example

```
Container = CRM.GetBlock('container');
Container.WorkflowTable = 'company';
```

Causes the **ShowNewWorkflowButtons** property to use the Company table.

ShowWorkflowButtons

Displays or hides workflow buttons on a view or edit screen for a record.

Pass the Record block as the argument to the **Execute(Arg)** method. You cannot set the Record object in the **ArgObj** property.

Values

This property can take one of the following values:

- **true**. Displays the workflow buttons.
- **false**. Hides the workflow buttons.

Examples

```
Record = CRM.FindRecord('MyTable','Table_Id=99');
EntryGroup = CRM.GetBlock('MyTableBlock');
EntryGroup.ShowWorkflowButtons = true;
CRM.AddContent(EntryGroup.Execute(Record));
Response.Write(CRM.GetPage());
```

Displays the workflow buttons on the EntryGroup object.

ShowNewWorkflowButtons

Displays or hides a **New** button on a screen that shows a list of records.

The **New** button creates a record in a workflow. To specify the table to be used by this property, use the **WorkflowTable** property.

Values

This property can take one of the following values:

- **true**. Displays the **New** button.
- **false**. Hides the **New** button.

Example

```
List = CRM.GetBlock('MyTableList');
List.WorkflowTable = 'MyTable';
List.ShowNewWorkflowButtons = true;
CRM.AddContent(List.Execute(''));
Response.Write(CRM.GetPage());
```

Shows the **New** button.

Code example: ShowWorkflowButtons property

The ASP file that contains the below code 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)
    {
        // Display the Delete button.
        DisplayButton(Button_Delete) = true;
        // Display the Continue button. This button takes the user back to the list.
        DisplayButton(Button_Continue) = true;
        Title = "My Table Details";
    }

// If no ID was passed, then switch to the New mode.
if (!Defined(ThisId))
    {
        MyRecord = CRM.CreateRecord("MyTable");
        if (CRM.Mode <= Edit)CRM.Mode = Edit;
    }
else
    {
        MyRecord = CRM.FindRecord("MyTable","Tab_TableId = "+ThisId);
    }

Holder.ShowWorkflowButtons = true;
Holder.WorkflowTable = 'MyTable';
CRM.AddContent(Holder.Execute(MyRecord));
Response.Write(CRM.GetPage());
%>
```

Code example: WorkflowTable and ShowNewWorkflowButtons properties

The following ASP code demonstrates how to use the **WorkflowTable** and **ShowWorkflowButtons** properties. This code 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 the list into a container and add the button to the
container.
Holder = CRM.GetBlock("container");
// Add the list to the container.
Holder.AddBlock(MyList);
// Hide the default Edit/Save button on the container.
Holder.DisplayButton(Button_Default) = false;
// Configure the list to show all records.
MyList.ArgObj = '';
// Show the new workflow buttons for the Primary rules.
Holder.WorkflowTable = 'MyTable';
Holder.ShowNewWorkflowButtons = true;
// Display the container.
CRM.AddContent(Holder.Execute(''));
Response.Write(CRM.GetPage());
%>
```

CRMContentBlock object

This object is a simple block that takes a text string and displays it on the page. Use this object in conjunction with other blocks on a screen.

Syntax to create 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());
```

CRMContentBlock properties

Contents. Sets the text string for the Content block.

Contents

Sets the text string for the Content block.

Values

Text. Specifies the text to display. This must be WideString value.

Examples

```
test = CRM.GetBlock('content');
test.contents = '<table> <td class=tablehead>My Details</td></table>'
CRM.AddContent(test.Execute());
Response.Write(CRM.GetPage());
```

Sets **My Details** as a header for the screen.

CRMEntryBlock object

The CRMEntryBlock object corresponds to a single field that's displayed or edited onscreen. You can create many entry types, such as text blocks, multi-select boxes, and currency input boxes.

The CRMEntryBlock object is a child of the **CRM object**. You usually add entries to an entrygroup or a container block but CRMEntryBlock doesn't inherit the properties or methods of a block to which it's added.

You can use JavaScript scripts on these blocks to perform tasks when they load, change, and are validated.

The CRMEntryBlock object properties are similar to the field properties available when adding entries to a screen in the **<My Profile> | Administration | Customization** area of Sage CRM.

Preceding code:

```
EntryGroup = CRM.GetBlock("GroupBlockName");
EntryGroup.AddEntry("entryname");
Entry = EntryGroup.GetEntry("entryname");
```

- **CRMEntryBlock methods**
- **CRMEntryBlock properties**

CRMEntryBlock methods

RemoveLookup. Removes specified items from the lists.

RemoveLookup

Removes specified items from the lists.

Only use this method if the **EntryType** property value of the target Entry block is set to **21**.

Parameters

String. Specifies the list item to be removed.

Examples

```
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());
```

Removes the **Customer** list item from the **Type** list on the Company entry screen.

CRMEntryBlock properties

- **AllowBlank**. Specifies whether the field can be set to a blank value.
- **Caption**. Allows you to change a field caption on a screen.
- **CaptionPos**. Sets the position of field captions relative to the field values for an entity record.
- **CreateScript**. Specifies the server-side JavaScript that is run upon the creation of the entry instance.
- **DefaultType**. Sets the default value for the field.
- **DefaultValue**. Specifies the default string value to use for the field when a new entity record is created.
- **EntryType**Sets the field type.
- **Fam**. Sets the caption family of the **CRMEntryGroupBlock object**.
- **FieldName**. Specifies the name by which the field is referenced.
- **Hidden**. Hides or shows an entry.
- **JumpEntity**. Hyperlinks the field in view mode to an entity summary screen.
- **LookUpFamily**. Sets the look-up family for an entry.
- **MaxLength**. Sets the maximum length of a field value (**CRMEntryGroupBlock object**).
- **MultipleSelect**. Sets if the user can select more than one item from the entry block list.
- **OnChangeScript**. Specifies the JavaScript to run when the field value is changed.
- **ReadOnly**. Specifies whether the field is read-only or writable.
- **Required**. Specifies whether the field is required or optional.
- **Size**. Specifies the maximum field size in characters.
- **ValidateScript**. Specifies the server-side validation JavaScript to run when the entry is executed in save mode.
- **AllowUnassigned**. Changes the option name from **None** to **Unassigned**
- **Restrictor**. Specifies an Advanced Search Select field that restricts the search values for the field.

- **CopyErrorsToPageErrorContent**. Specifies where to display validation errors related to a particular block on the page.
- **Width**. Specifies the number of columns a field should span.
- **Height**. Specifies the number of rows a field should span.

AllowBlank

Specifies whether the field can be set to a blank value.

Only use this property if the **EntryType** property value of the target Entry block is set to **21**.

Values

This property can take one of the following values:

- **true**(default). Enables blank field values.
- **false**. Disables blank field values. The user must enter a value for the field.

Example

```
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());
```

Disables blank values for the comp_revenue field.

Caption

Allows you to change a field caption on a screen.

This property is applicable to a particular screen only. To permanently change a caption for all screens, go to **<My Profile> | Administration | Customization | <Entity> | Fields** area in Sage CRM.

Values

Value. Specifies the field caption to use. Accepts a string value.

Examples

```
r = CRM.FindRecord('Company','Comp_companyid=30');
```

```
EG = CRM.GetBlock('companyboxlong');
NewE = EG.GetBlock('comp_revenue');
NewE.Caption = 'My new caption';
CRM.AddContent(EG.Execute(r));
Response.Write(CRM.GetPage());
```

Changes the comp_revenue field caption to **My new caption**.

CaptionPos

Sets the position of field captions relative to the field values for an entity record.

This property is normally used with the JavaScript Enumerator object.

Values

This property can take one of the following values:

- **1**. Places the field captions above the field values.
- **2**. Places the field captions to the left of the field values.
- **3**. Places the field captions to the left of the field values;
  field captions and values are aligned to the left.
- **6**. Places the field captions to the left of the field values;
  field captions are aligned to the right;
  field values are aligned to the left.

Examples

```
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());
```

Places the field captions to the left of the field values;
Field captions are aligned to the right, field values are aligned to the left.

CreateScript

Specifies the server-side JavaScript that is run upon the creation of the entry instance.

Values

String. Specifies the JavaScript to run. Within the JavaScript, any of the current entry block properties can be accessed.

Examples

```
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());
```

Sets the maximum length of the Entry block comp_name field instance to 20 characters.

DefaultType

Sets the default value for the field.

Use this property in conjunction with the **EntryType** and **DefaultValue** properties.

Values

This property can take one of the following values:

- **0** (iDefault_NoDefault). Specifies that no default value is used.
- **1** (iDefault_Value). Specifies to use the default value set in the **DefaultValue** property.
- **2** (iDefault_CurrentUserId). For fields that accept a user ID as a value, specifies to use the ID of the current user.
- **6** (iDefault_CurrentDateTime). For DateTime fields, specifies to use the current date and time.
- **14** (iDefault_CurrentDateTimePlus30Mins). For DateTime fields, specifies to use the current date and time increased by 30 minutes.

Examples

```
R = CRM.CreateRecord('company');
EG = CRM.GetBlock('companyboxlong');
CompanyE = EG.GetEntry('comp_name');
CompanyE.DefaultType = 1;
CompanyE.DefaultValue = 'New company name';
CRM.AddContent(EG.Execute(R));
Response.Write(CRM.GetPage());
```

Sets the default value of the comp_name field to the value specified in the **DefaultValue** property.

DefaultValue

Specifies the default string value to use for the field when a new entity record is created.

Only use this property if you set the **DefaultType** property to **1**.

Values

String. Any string value.

Examples

```
R = CRM.CreateRecord('company');
EG = CRM.GetBlock('companyboxlong');
E = EG.GetEntry('comp_name');
E.DefaultType = 1;
E.DefaultValue = 'My company name';
CRM.AddContent(EG.Execute(R));
Response.Write(CRM.GetPage());
```

Sets the default value of the comp_name field to **My company name** for each new company record being created.

EntryType

Sets the field type.

This property is only applicable to EntryBlocks retrieved by using the **GetBlock(BlockName)** method, that is, EntryBlocks not associated with specific fields.

Values

This property can take one of the following values:

- **10** (iEntryType_Text). Single-line text entry.
- **11** (iEntryType_MultiText). Multi-line text entry.
- **12** (iEntryType_EmailText). Email address.
- **13** (iEntryType_UrlText). Web URL (Uniform Resource Locator) address.
- **14** (iEntryType_VarTex). For internal use only.
- **15** (iEntryType_TextWithCBSearch). Text with check box search.
- **21** (iEntryType_Select). Selection from a combo box.

- **22** (iEntryType_UserSelect). Selection from the User table (one user only).
- **23** (iEntryType_ChannelSelect). Selection from the Channel table (team data).
- **24** (iEntryType_UserMultiSelect). Selection from the User table (multiple users).
- **25** (iEntryType_ProductSelect). Selection from the Product table.
- **26** (iEntryType_SearchSelect). Search select. This field type is deprecated. For your custom fields, use field type **56**.
- **27** (iEntryType_TableSelect). Intelligent select.
- **28** (iEntryType_MultiSelect). Multiple selection from a combo box.
- **31** (iEntryType_Integer). Integer.
- **32** (iEntryType_Numeric). Numeric currency value.
- **41** (iEntryType_DateTime). Date and time.
- **42** (iEntryType_Date). Date only.
- **44** (iEntryType_StoredProc). Stored procedure.
- **45** (iEntryType_CheckBox). Check box.
- **46** (iEntryType_Link). Link. This field type is only used in connection with system-generated screens that allow the entry of data into telephone number fields. For your custom fields, use EntryType **50**.
- **50** (iEntryType_Phone). Phone.
- **51** (iEntryType_Currency). Currency.
- **54** (iEntryType_FileList). For internal use only.
- **56** (iEntryType_AdvSearchSelect). Advanced Search Select.
- **57** (iEntryType_Minutes). Minutes.
- **59** (iEntryType_CurrencySelect). Currency symbols.
- **63** (iEntryType_UserGroupsSelect). User group select.
- **64** (iEntryType_SuperAdvSearchSelect). Super Advanced Search Select.

Examples

```
Entry = CRM.GetBlock('entry');
Entry.FieldName = "Check Box";
Entry.EntryType = 45;
```

Sets the field caption of the new entry to **My check box**, and then sets the field type to check box (45).

Fam

Sets the caption family of the **CRMEntryGroupBlock object**. This property controls the captions that appear on each entry.

By default, the caption shown is the translation of the caption family (column names) plus the caption code (field name). You can change the caption by setting the Fam property value and adding a translation for the Fam value and the field name.

To view a list of column names, go to **<My Profile> | Administration | Customization | Translations**.

Values

Fam. Specifies the family name to use to find the translation for the entry caption. This must be a string value.

Examples

```
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 caption family';
Response.Write(CompanyBlock.Execute(CompanyRec));
```

Sets the caption family of the comp_name entry to **My caption family**.

FieldName

Specifies the name by which the field is referenced.

This property is only applicable to EntryBlocks retrieved by using the **GetBlock(BlockName)** method, that is, EntryBlocks not associated with specific fields.

Values

Any string value.

Example

```
Entry = CRM.GetBlock("entry");
Entry.FieldName = "My check box";
Entry.EntryType = 45;
CRM.AddContent(Entry.Execute());
Response.Write(CRM.GetPage());
```

Sets the field caption of the new entry to **My check box**, and then sets the field type to check box (45).

Hidden

Hides or shows an entry.

As a result, the entry is not displayed when the corresponding **CRMEntryGroupBlock object** is executed. For example, this can be useful if you want to assign an entry to an entry group, but don't want the users to be able to view it.

Values

This property can take one of the following values:

- **true**. Specifies to hide an entry.
- **false**. Specifies to show an entry.

Example

```
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());
```

Hides the company_revenue entry block.

JumpEntity

Hyperlinks the field in view mode to an entity summary screen.

The entity must be relevant to the field. The ID field of the entity must exist in the table or view on which the summary screen is based. This property is only useful when the summary screen is based on a view that contains fields from multiple tables.

Values

This property accepts one of the following entity names:

- Company
- Person
- Communications
- Case

- Opportunity
- Solution
- Address
- Library
- Notes

Example

```
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());
```

Hyperlinks the comp_name field to the company summary screen.

LookUpFamily

Sets the look-up family for an entry.

The look-up family defines what entries appear in the list for the entry. For example, if the look-up family is DayName, a list of days is displayed.

This property is only applicable if the **EntryType** property is set to **21**.

Values

LookUpFamily. Specifies the name of the family. This must be a string value.

Examples

```
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());
```

Creates a new selection entry group that uses the DayName family for selection items.

MaxLength

Sets the maximum length of a field value (**CRMEntryGroupBlock object**). This doesn't change the size of the entry box. To change the entry box size, use the **Size** parameter.

Values

MaxLength. Specifies the maximum length of value in characters. This must be an integer value.

Examples

```
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());
```

Sets the maximum value of comp_name to 5 characters.

MultipleSelect

Sets if the user can select more than one item from the entry block list.

You must save the possible entries in a relevant location such as a link table.

This property is only applicable if the **EntryType** property is set to **21**.

Values

This property can take one of the following values:

- **true**. Specifies that the user can select multiple entries.
  If you set this value, make sure that the value set in the **Size** property allows the block to accommodate all possible entries.
- **false**. Specifies that the user can only select one entry.

Example

```
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());
```

Sets the comp_source entry block of the companyboxlong entry group to MultipleSelect; also sets the entry size to 10.

OnChangeScript

Specifies the JavaScript to run when the field value is changed.

This property is only applicable if the **ReadOnly** property is set to **false**.

Values

JavaScript to run. This must be a string value.

Examples

```
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('User's first name has changed')";
CRM.AddContent(PersonBlock.Execute(ThisPersonRecord));
Response.Write(CRM.GetPage());
```

Displays an alert when a user's first name has changed.

ReadOnly

Specifies whether the field is read-only or writable.

Values

This property can take one of the following values:

- **true**. Specifies that the field is read-only.
- **false**. Specifies that the field is writable.

Example

```
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());
```

Makes the pers_titlecode field read-only.

Required

Specifies whether the field is required or optional.

Values

This property can take one of the following values:

- **true**. Specifies that the field is required and must be populated with a value.
- **false**. Specifies that the field is optional.

Examples

```
Block = CRM.GetBlock('PersonBoxShort');
Title = Block.GetEntry('pers_title');
Title.Required = true;
CRM.AddContent(Block.Execute());
Response.Write(CRM.GetPage());
```

Sets the pers_title field as a required field.

Size

Specifies the maximum field size in characters. This is the field size displayed on the screen.

Values

Integer

Examples

```
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());
```

Retrieves the pers_firstname field from the PersonBoxLong screen, and then sets the maximum size of the field to 40 characters.

ValidateScript

Specifies the server-side validation JavaScript to run when the entry is executed in save mode.

The script sets the Valid variable to one of these values:

- **true**. Indicates that validation has succeeded.
- **false**. Indicates that validation has failed. In this case, the screen remains in edit mode and displays an error message. An orange question mark is displayed next to the field whose validation has failed.

Values

JavaScript to run. This must be a string value. You can set the ErrorStr variable in your script to display a custom error message.

Examples

```
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());
```

The script in this example sets the Valid variable to true if the comp_name field is not empty (comp_name.value != '').
If the comp_name field is empty, the following error message is displayed: "Please correct the highlighted entries".

AllowUnassigned

Changes the option name from **None** to **Unassigned**.

This property is only applicable to entry blocks whose entry type is TableSelect (lists to select users). Option name depends on the combination of the AllowUnassigned and **AllowBlank** values, as follows:

AllowUnassigned value	AllowBlank value	Option name
false	false	N/A
true	false	Unassigned
false	true	None
true	true	None

Values

This property can take one of the following values:

- **true**. Sets the option name to **Unassigned**.
- **false**.Does not allow the **Unassigned** option.

Examples

```
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());
```

Changes the option caption from **None** to **Unassigned**.When a user clicks **Change**, the **Account Manager** list includes the **Unassigned** option instead of **None**.

Restrictor

Specifies an Advanced Search Select field that restricts the search values for the field.

Values

Field name in the form of a WideString value.

Read: Get_Restrictor

Write: Set_Restrictor

Examples

```
Restrictor = "cmli_comm_companyid";
```

Adds a new Advanced Search Select field to restrict search results based on an existing Advanced Search Select field. The existing field searches for companies; when a company is selected for the existing field the new Advanced Search field searches for records that belong to that company.

CopyErrorsToPageErrorContent

Specifies where to display validation errors related to a particular block on the page.

Values

This property can take one of the following values:

- **true**. Specifies to show validation errors at the top of the page.
- **false**. Specifies to show validation errors next to the related block on the page.

Examples

```
CompanyEntryGroup = CRM.GetBlock("CompanyBoxLong");
CompanyEntryGroup.Title = "Company";
AddressEntryGroup = CRM.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=CRM.GetBlock("container");
container.AddBlock(CompanyEntryGroup);
container.AddBlock(AddressEntryGroup);
```

Creates two entry blocks, one of which fails validation. The corresponding validation errors are displayed at the top of the page.

Width

Specifies the number of columns a field should span.

Examples

```
var myBlock = CRM.GetBlock("CompanyBoxShort");
myEntryBlock = myBlock.GetEntry("comp_type");
myEntryBlock.Height = "2";
myEntryBlock.Width = "4";
```

Sets the width of the `comp_type` field to four columns. This code generates HTML similar to the following:

```
<td colspan="4" valign="TOP" rowspan="2">
<span id="_Captcomp_type" class="VIEWBOXCAPTION">
Type:
```

```
</span>
<br>
<span id="_Datacomp_type" class="VIEWBOX style='WIDTH:100px;'>
Prospect 
</span>
</td>
```

Height

Specifies the number of rows a field should span.

Examples

```
var myBlock = CRM.GetBlock("CompanyBoxShort");
myEntryBlock = myBlock.GetEntry("comp_type");
myEntryBlock.Height = "2";
myEntryBlock.Width = "4";
```

Sets the height of the `comp_type` field to two rows. This code generates HTML similar to the following:

```
<td colspan="4" valign="TOP" rowspan="2">
<span id="_Captcomp_type" class="VIEWBOXCAPTION">
Type:
</span>
<br>
<span id="_Datacomp_type" class="VIEWBOX style='WIDTH:100px;'>
Prospect 
</span>
</td>
```

CRMEntryGroupBlock object

The CRMEntryGroupBlock object corresponds to a screen in CRM. Use **CRMEntryGroupBlock methods** to create screens and control custom data entry and editing. You can generate many types of entry group, such as a Company search box, a Person entry box, and a Case detail box.

This block also contains the following standard CRM buttons:

- **Change** or **Save**. Displayed as **Change** when the screen is in View mode and **Save** when the screen is in Edit mode. This button is shown by default.
- **Delete** or **Confirm Delete**. Displayed as **Delete** when the screen is in View mode and **Confirm Delete** when the screen is in Confirm Delete mode. This button is not shown by default.
- **Continue** or **Cancel**. Displayed as **Continue** when the screen is in View mode and **Cancel** when the screen is in Edit or Confirm Delete mode. This button is not shown by default.

In this section:

CRMEntryGroupBlock methods

- **AddEntry(EntryName, Position, Newline)**. Enables new entries to be added dynamically to EntryGroups.
- **DeleteEntry(EntryName)**. Deletes the specified entry from the EntryGroup.
- **GetEntry**. Returns a reference to the specified entry.

AddEntry(EntryName, Position, Newline)

Enables new entries to be added dynamically to EntryGroups. The changes only apply to the ASP page in which they are used.

Parameters

- **EntryName** (required). Specifies the new entry to be added. The entry can be passed in as the field name or an existing EntryBlock. In any case, the field must be relevant to the existing EntryGroup block. It must exist in the table on which the EntryGroup block is based.
- **Position** (optional). Specifies the position in the group at which to add the entry. Possible values:

- **<a positive integer>**. Specifies the index of the position at which to add the entry.
- **0**. Adds the entry to the first position in the group.
- **-1** (default). Adds the entry to the last position in the group.

- **NewLine** (optional). Specifies whether to show the entry on a new line.
  Possible values:
  - **true** (default). Shows the entry on a new line.
  - **false**. Shows the entry on the same line.

Return value

CRMEntryBlock object

Examples

```
var EntryGroup;
EntryGroup = CRM.GetBlock("personboxlong");
EntryGroup.AddEntry("pers_faxnumber", 0, false);
EntryGroup.AddEntry("pers_phonenumber", 0, false);
CRM.AddContent(EntryGroup.Execute());
Response.Write(CRM.GetPage());
```

Adds the pers_faxnumber and pers_phonenumber fields to the start of the entry group.

DeleteEntry(EntryName)

Deletes the specified entry from the EntryGroup.

Parameters

EntryName. Specifies the name of the field to be deleted.

Return value

None

Examples

```
var r;
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());
```

Deletes the comp_revenue field from CompanyBoxLong for non-administrators.

GetEntry

Returns the specified entry.

Parameters

EntryName. Specifies the entry name to be returned from the EntryGroup.

Return value

One of the following:

- **CRMEntryBlock object** if the specified entry exists.
- Nil object if the specified entry does not exist.

Examples

```
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());
```

Retrieves the pers_fistname entry from the PersonBoxShort EntryGroup and sets the entry to read-only.

CRMEntryGroupBlock properties

ShowSavedSearch. When the block is executed, shows or hides the Saved Search functionality as part of the entry group.

ShowSavedSearch

When the block is executed, shows or hides the Saved Search functionality as part of the entry group.

This property is only applicable to EntryGroup blocks whose screen type is Search block and that have an associated List.

Values

- **false** (default). Hides saved searches.
- **true**. Shows a list of saved searches for the entity and allows the user to create and edit the saved searches.

Examples

```
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.submit();
"));
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 using ASP in the same way as any other Sage CRM page. You must format the appearance of the files in HTML.

Syntax to use this object:

```
var afile;
afile = CRM.GetBlock('file');
afile.FileName = 'general.htm';
afile.Translate = false;
afile.ProperCase = false;
afile.DirectoryPath = 'C:\\<Folder>\\<Subfolder>\\';
CRM.AddContent(afile.Execute());
Response.Write(CRM.GetPage());
```

You can use the **Translate** property which allows you to dynamically choose a file based on the user's language code, or the **ProperCase** property which allows you to display the text with initial caps. These simple ASP statements include the named file on the page. If the author sets translate to true, the file name included is changed from filename to filename_US or filename_DE, depending on the user's language specified in Sage CRM. If the file name extension is not specified, .txt is used by default. This means you must specify .htm and format the text.

- **CRMFileBlock properties**

CRMFileBlock properties

- **DirectoryPath**. Specifies the folder that stores the files.
- **FileName**. Specifies the name of the file to use.
- **ProperCase**. Applies initial capitals formatting to the text: the first letter of every word begins with a capital letter.
- **Translate**. Specifies to use files containing text in the user's language.

DirectoryPath

Specifies the folder that stores the files. If you omit this parameter, the following path is used by default:
<Sage CRM installation folder>\WWWRoot\Reports

By default, the Sage CRM installation folder is

%ProgramFiles(x86)%\Sage\CRM\CRM

Values

WideString. Specifies the path to the folder.

Examples

```
var afile;
afile = CRM.GetBlock('file');
afile.FileName = 'Results.htm';
afile.DirectoryPath = 'c:\\MyFolder\\Storage';
```

FileName

Specifies the name of the file to use.

Values

WideString. Specifies the file name.

Examples

```
var afile;
afile = CRM.GetBlock('file');
afile.FileName = 'Results.htm';
```

ProperCase

Applies initial capitals formatting to the text: the first letter of every word begins with a capital letter.

Values

This property can take one of the following values:

- **true**. Formats the text using initial capitals.
- **false**. Keeps the existing formatting of the text.

Example

```
var afile;
afile = CRM.GetBlock('file');
afile.FileName = 'general.htm';
```

```
afile.ProperCase = true;
CRM.AddContent(afile.Execute());
Response.Write(CRM.GetPage());
```

Formats text in the general.htm file using initial capitals.

Translate

Specifies to use files containing text in the user's language. Names of these files have the following format:

<file name>_<language>.<extension>

where <language> is the language set for the user in Sage CRM.

Values

This parameter can take one of the following values:

- **true**. Searches for files in the user's language.
- **false**. Searches for files that do not contain the <language> suffix.

Examples

```
var afile;
afile = CRM.GetBlock('file');
afile.FileName = 'Results.htm';
afile.Translate = true;
```

Specifies to use the file containing text in the user's language.

For example, if the user's language in Sage CRM is set to US English, this example looks for the Results_US.htm file.

CRMGraphicBlock object

The CRMGraphicsBlock object is a child of the CRM block and parent of the PipeLineGraphic, OrgChartGraphic, and ChartGraphicBlock objects.

Use CRMGraphicsBlock to display images through an ASP page. Graphics blocks are more powerful than standard static images because you can use variables to create them. The variables may represent live data from a database or incorporate details of the current user such as their privileges or settings. A graphic created by CRMGraphicsBlock is recreated every time it's requested, so if it's created using variables, the graphic is based on real time data.

The option to load 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.

Syntax to initiate this block:

```
graphic=CRM.GetBlock('graphic');
```

- **CRMGraphicBlock methods**
- **CRMGraphicBlock properties**

CRMGraphicBlock methods

- **Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4)**. Draws an elliptically curved line.
- **Animation(Mode, Value)**. Sets animation parameters.
- **Brush(Mode, Value)**. Changes the color and pattern used when drawing the background or filling in graphical shapes.
- **Chord(X1,Y1,X2,Y2,X3,Y3,X4,Y4)**. Creates a shape that is defined by an arc and a line that joins the endpoints of the arc.
- **Effect(Mode, Value)**. Applies effects.
- **Ellipse(X1,Y1,X2,Y2)**. Draws a circle or ellipse.
- **FlipHoriz()**. Flips the image horizontally.
- **FlipVert()**. Flips the image vertically.
- **Font(Mode, Value)**. Changes the current font.
- **FontColor(Color)**. Changes the color of the current font.
- **FontSize(Size)**. Changes the size of the current font.

- **GradientFill(StartColor, EndColor, Direction, Colors)**. Fills the graphic with a gradient of colors.
- **GrayScale()**. Converts an image to grayscale.
- **LoadBMP(Filename)**. Loads a specified bitmap file as the new image.
- **LoadImage(text)**. Loads image from the specified file.
- **LoadJPG(Filename)**. Loads an image from the specified JPEG file.
- **LineTo(X,Y)**. Draws a line from the initial pen position up to the points specified in the method parameters.
- **Monochrome()**. Converts an image to monochrome (black and white).
- **MoveTo(X,Y)**. Moves the pen to the specified coordinates.
- **Pen(Mode, Value)**. Sets the appearance of a line drawn with the current pen.
- **PenColor(Color)**. Sets the color for the current pen.
- **PenWidth(Width)**. Sets the line width for the current pen.
- **PieShape(X1,Y1,X2,Y2,X3,Y3,X4,Y4)**. Draws a pie-shaped wedge on the image.
- **Rectangle(X1,Y1,X2,Y2)**. Draws a rectangle.
- **Resize(Width, Height)**. Sets the dimensions of the image.
- **Rotate(Number)**. Rotates an image.
- **RoundRect(X1,Y1,X2,Y2,X3,Y3)**. Draws a rounded rectangle.
- **SaveAsJPG(text)**. Saves the current image as a .jpeg file with 16-bit color depth.
- **TextOut(X, Y, Text, transparent=True/False)**. Writes text on an image.
- **TextOutCenter(Left, Top, Right, Bottom, Text, Transparent, Ellipse)**. Writes text on an image and centers it in a rectangle area defined by the parameters.

Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4)

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, counterclockwise, 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**. Integer.
- **Y1**. Integer.
- **X2**. Integer.

- **Y2**. Integer.
- **X3**. Integer.
- **Y3**. Integer.
- **X4**. Integer.
- **Y4**. Integer.

Example

```
var graphic;
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());
```

Animation(Mode, Value)

Sets animation parameters.

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)

Changes the color and pattern used when drawing the background or filling in graphical shapes. You can select one of predefined patterns by using the Style mode or load a pattern from an image by using the Load mode.

Parameters

- **Mode**. Specifies the mode to use. This parameter accepts a WideString value.
- **Value**. Specifies the value for the selected mode. This parameter accepts a WideString value.

Examples

```
Graphic.Brush('Load','c:\\MyImages\\SoapBubbles.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)

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**. Integer.
- **Y1**. Integer.
- **X2**. Integer.
- **Y2**. Integer.
- **X3**. Integer.
- **Y3**. Integer.
- **X4**. Integer.
- **Y4**. Integer.

Examples

```
Graphic.Chord(10,10,25,25,30,30,40,40);
```

Effect(Mode, Value)

Applies effects.

Parameters

- **Mode**. Specifies the name of the effect to apply.
- **Value**. Specifies parameters for the selected effect.

Examples

```
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)

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**. Integer.
- **Y1**. Integer.
- **X2**. Integer.
- **Y2**. Integer.

Examples

```
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()

Flips the image horizontally.

Parameters

None

Examples

```
Graphic.FlipHoriz();
```

FlipVert()

Flips the image vertically.

Parameters

None

Example

```
Graphic.FlipVert();
```

Font(Mode, Value)

Changes the current font. For best results, use TrueType fonts. To apply changes, use the **TextOut (X, Y, Text, transparent=True/False)** method.

Parameters

- **Mode**. Specifies changes to apply to the font. This parameter can take one of the following WideString values:
  - **Name**. Changes the current font. Use the **Value** parameter to specify the name of the font to use.
  - **Size**. Changes the font size. Use the **Value** parameter to specify the font size.
  - **Color**. Changes the font color. Use the **Value** parameter to specify the font color.

- **Bold**. Toggles between bold and normal font. Use the **Value** parameter to apply bold (**true**) or normal **(false**) font.
- **Italic**. Toggles between italic and normal font. Use the **Value** parameter to apply italic (**true**) or normal **(false**) font.
- **Underline**. Toggles between underlined and normal font. Use the **Value** parameter to apply underlined (**true**) or normal (**false**) font.
- **Strikeout**. Adds or removes a line through the middle of characters. Use the **Value** parameter to add a line (**true**) or remove the line (**false**).
- **Rotate**. Rotates the font. This parameter may not work for some fonts.

- **Value**. Specifies the value for the **Mode** parameter. This parameter accepts a WideString value.

Examples

```
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 = 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)

Changes the color of the current font. This method is identical to **Font(Color, Value)**.

Parameters

Color. Specifies the name of the font color to use. This parameter accepts a WideString value.

Example

```
Graphic.FontColor('blue');
```

Changes the font color to blue.

FontSize(Size)

Changes the size of the current font. This method is identical to **Font(Size, Value)**.

Parameters

Size. Specifies the font size (in pixels) to use.

Example

```
Graphic.FontSize(24);
```

Changes the font size to 24 pixels.

GradientFill(StartColor, EndColor, Direction, Colors)

Fills the graphic with a gradient of colors.

Parameters

- **StartColor**. Specifies the initial color of the gradient. This parameter accepts a WideString value.
- **EndColor**. Specifies the end color of the gradient. This parameter accepts a WideString value.
- **Direction**. Specifies the direction in which the gradient color changes. This parameter can take one of the following WideString values:
  - **TopToBottom**
  - **BottomToTop**
  - **LeftToRight**
  - **RightToLeft**
- **Colors**. Specifies color depth (or bit depth) of the gradient. This parameter can take an integer value. The default value is **64** (64-bit). Gradients usually look better in 24-bit JPEG images, as the colors that can be used with GIFs are more limiting.

Examples

```
Graphic.GradientFill('Yellow','White','LeftToRight');
```

```
Graphic.GradientFill('Blue','White','TopToBottom',256);
```

GrayScale()

Converts an image to grayscale. This method doesn't reduce the number of colors in use.

Parameters

None

Examples

```
Graphic.Grayscale();
```

LoadBMP(Filename)

Loads a specified bitmap file as the new image.

To change the image dimensions, use the **ImageWidth** and **ImageHeight** properties.

Parameters

Filename. Specifies path to the image in the form of absolute server address. This parameter accepts a WideString value.

Examples

```
Graphic = CRM.GetBlock('graphic');
Graphic.LoadBMP("D:\\Program Files\\Sage\\CRM\\CRM\\WWWRoot\\Img\\plain.bmp");
CRM.AddContent(Graphic.Execute());
Response.Write(CRM.GetPage());
```

Loads a bitmap file named plain.bmp.

LoadImage(text)

Loads image from the specified file.

To change the image dimensions, use the **ImageWidth** and **ImageHeight** properties.

You can load images in the following formats:

- .bmp
- .ico
- .gif
- .jpg
- .wmf
- .emf

Parameters

Text. Specifies the file name and path.

If the file is stored in the **<Sage CRM installation folder>\WWWRoot\Img** folder, you only need to specify the name of the file. If the file is stored in any other location, specify the full path to the file.

This parameter accepts a WideString value.

Examples

```
Graphic.LoadImage('Image.gif');
```

Loads image from a file named Image.gif located in the following folder on the Sage CRM server: <Sage CRM installation folder>\WWWRoot\Img

```
Graphic.LoadImage('c:\\MyImages\\Image.gif');
```

Loads image from a file named Image.gif located in the C:\MyImages folder on the Sage CRM server.

LoadJPG(Filename)

Loads an image from the specified JPEG file.

To change the image dimensions, use the **ImageWidth** and **ImageHeight** properties.

Parameters

Filename. Specifies the name and path of the file to load.

Examples

```
Graphic = CRM.GetBlock('graphic');
Graphic.LoadJPG("C:\\Program Files\\Sage\\CRM\\CRM58\\WWWRoot\\Img\\Image.jpg");
CRM.AddContent(Graphic.Execute());
Response.Write(CRM.GetPage());
```

Displays an image stored in the Image.jpg file.

LineTo(X,Y)

Draws a line from the current pen position up to the points specified in the method parameters.

The points specified in the parameters are not included in the line. Changes the pen position to the specified points. The line is drawn using the current pen and color.

To set the initial pen position for drawing a line, use the **MoveTo(X,Y)**.

Parameters

Use the below parameters to specify the coordinates for drawing a line.

- **X**. Integer.
- **Y**. Integer.

Examples

```
Graphic.LineTo(50,50);
```

Monochrome()

Converts an image to monochrome (black and white).

This method makes irreversible changes to the image. To return to color, you need to redraw the image.

Parameters

Boolean. Specifies whether to convert the image to monochrome. This parameter can take one of the following values:

- **true**. Specifies to convert the image to monochrome.
- **false**. Specifies to keep the current image color.

Examples

```
Graphic.Monochrome(true);
```

Converts the image to monochrome.

MoveTo(X,Y)

Moves the pen to the specified coordinates.

Use this method to set the initial pen position before calling the **LineTo(X,Y)** method.

Parameters

Use the below parameters to specify the pen coordinates:

- **X**. Integer.
- **Y**. Integer.

Examples

```
Graphic.MoveTo(50,50);
```

Pen(Mode, Value)

Sets the appearance of a line drawn with the current pen.

Affects any line-drawing actions performed after invoking this method.

Parameters

- **Mode**. Specifies the pen style to use. This parameter can take one of the following WideString values:
  - **Style**. Specifies the line style to use, for example, DashDot.
  - **Color**. Specifies the color to use.
  - **Width**. Specifies the line width in pixels.
- **Value**. Specifies the value for the **Mode** parameter. This parameter accepts a WideString value.

Examples

```
Graphic.Pen('Style','DashDot');
Graphic.Pen('Color','Blue');
Graphic.Pen('Width','3');
```

PenColor(Color)

Sets the color for the current pen.

This method is identical to **Pen(Color, Value)**

Parameters

Color. Specifies the color to use. This parameter accepts a WideString value.

Examples

```
Graphic.PenColor('green');
```

PenWidth(Width)

Sets the line width for the current pen.

This method is identical to **Pen(Width, Value)**.

Parameters

Width. Specifies the line width in pixels.

Example

```
Example Graphic.PenWidth('3');
```

PieShape(X1,Y1,X2,Y2,X3,Y3,X4,Y4)

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**. Integer.
- **Y1**. Integer.
- **X2**. Integer.
- **Y2**. Integer.
- **X3**. Integer.
- **Y3**. Integer.
- **X4**. Integer.
- **Y4**. Integer.

Examples

```
Graphic.PieShape(10,10,25,25,30,30,40,40);
```

Rectangle(X1,Y1,X2,Y2)

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**. Integer.
- **Y1**. Integer.
- **X2**. Integer.
- **Y2**. Integer.

Examples

```
Graphic.Rectangle(10,10,100,100);
```

Resize(Width, Height)

Sets the dimensions of the image.

The image is scaled to the specified size (which doesn't happen if you use the **ImageHeight** and **ImageWidth** properties). Do not set the **ImageHeight** and **ImageWidth** properties in the same block, as they override the value set by this method.

Parameters

- **Width**. Integer.
- **Height**. Integer.

Examples

```
Graphic.Resize(150,100);
```

Rotate(Number)

Rotates an image.

The corners of a rotated image are colored in the current brush color.

Parameters

Number. Specifies the number of degrees by which to rotate the image. This parameter accepts an integer value.

Examples

```
Graphic.Rotate(90);
```

RoundRect(X1,Y1,X2,Y2,X3,Y3)

Draws a rounded rectangle.

The rectangle has edges defined by the points (X1,Y1), (X2,Y1), (X2,Y2), (X1,Y2), but its corners are rounded. 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**. Integer.
- **Y1**. Integer.
- **X2**. Integer.

- **Y2**. Integer.
- **X3**. Integer.
- **Y3**. Integer.

Examples

```
Graphic.RoundRect(10,10,12,12,15,15);
```

SaveAsJPG(text)

Saves the current image as a .jpeg file with 16-bit color depth.

Parameters

Text. Specifies path to the file. This parameter accepts a WideString value.

Examples

```
Graphic.SaveAsJPG('c:\\cancel.jpg');
```

TextOut(X, Y, Text, transparent=True/False)

Writes text on an image.

Optionally, you can make the text transparent. By default, the text creates a blank rectangle where it is placed. It's written in co-ordinates specified in (X,Y) and is written in the current font color and size.

Parameters

- **X**, **Y**. Specify coordinates for writing text. These parameters accept integer values.
- **Text**. Specifies the text to write. Accepts a WideString value.
- **Transparent**. Specifies whether the text is transparent. This parameter accepts one of the following values:
    - **true**. Makes the text transparent.
    - **false**. Makes the text non-transparent.

Examples

```
Graphic.TextOut(10,10,'My text',true);
```

TextOutCenter(Left, Top, Right, Bottom, Text, Transparent, Ellipse)

Writes text on an image and centers it in a rectangle area defined by the parameters.

Parameters

- **Left**, **Top**, **Right**, **Bottom**. Integer.
- **Text**. Specifies the text to write. Accepts a WideString value.
- **Transparent**. Specifies whether the text is transparent. This parameter can take one of the following values:
  - **true**. Makes the text transparent.
  - **false**. Makes the text non-transparent.
- **Ellipse**. Adds an ellipsis (...) at the end of the text if it doesn't fit into the rectangle area and gets truncated. This parameter can take one of the following values:
  - **true**. Adds an ellipsis if the text gets truncated.
  - **false**. Doesn't add an ellipsis if the text gets truncated.

Example

```
Graphic.TextOutCenter(10,10,100,30,'My text',true,true);
```

CRMGraphicBlock properties

- **Border**. Sets the thickness of the border around the image.
- **Description**. Sets the description of the image.
- **hSpace**. Adds a horizontal space above and below the image.
- **ImageHeight**. Sets the height of the box in which the image is loaded.
- **ImageWidth**. Sets the width of the box in which the image is loaded.
- **SaveAsGifs**. Saves an image as a .gif or .jpeg file.
- **vSpace**. Adds a vertical space to the left and to the right of the image.

Border

Sets the thickness of the border around the image.

Values

Integer. Specifies the border thickness. The default value is 0.

Examples

```
Graphic.Border = 1;
```

Description

Sets the description of the image.

The specified description is displayed in place of the image if images are disabled in the user's browser.

Values

Text. Specifies the image description. This must be a WideString value.

Examples

```
Graphic.Description = "My image description";
```

hSpace

Adds a horizontal space above and below the image.

Values

Integer. Specifies the size of the horizontal space to add. By default, this value is 0.

Examples

```
Graphic.hSpace = 10;
```

ImageHeight

Sets the height of the box in which the image is loaded.

Values

Integer. Specifies the box height in pixels.

Example

```
Graphic.ImageHeight = 200;
```

ImageWidth

Sets the width of the box in which the image is loaded.

Values

Integer. Specifies the box width in pixels.

Examples

```
Graphic.ImageWidth = 200;
```

SaveAsGifs

Saves an image as a .gif or .jpeg file.

Values

- **true**. Saves the image as a .gif file with 256-color depth.
- **false**. Saves the image as a .jpeg file with 16-bit color depth.

When the Sage CRM server graphics adapter is configured to allow for 16-bit color depth or greater, this property is set to **false** by default. Otherwise, this property is set to **true**.

Examples

```
Graphic.SaveAsGifs = true;
```

vSpace

Adds a vertical space to the left and to the right of the image.

Values

Integer. Specifies the size of the vertical space to add. By default, this value is 0.

Examples

```
Graphic.vSpace = 10;
```

CRMGridColBlock object

Use the CRMGridColBlock object to set the properties of an individual column within a list. CRMGridColBlock is related to CRMList Block but is a child of the CRM object.

The properties that apply are similar to the fields available when adding columns to a custom list in the **<My Profile> | Administration | Customization | <Entity> | Lists** area of Sage CRM.

Syntax to use this object:

```
ListBlock = CRM.GetBlock("companygrid");
ListBlock.AddGridCol("gridcolname");
ListBlock.GetGridCol("gridcolname");
```


CRMGridColBlock properties

- **Alignment**. Sets the alignment of text within the column.
- **AllowOrderBy**. Sorts entries in the list by the values in the column.
- **CreateScript**. Specifies the server-side JavaScript to run upon the creation of the CRMGridColBlock instance.
- **CustomActionFile**. Hyperlinks a column to an ASP file.
- **CustomIdField**. Allows a value to be passed to the custom file when the corresponding column is selected.
- **JumpAction**. Specifies the jump action for the **CustomActionFile** property.
- **JumpEntity**. Adds a hyperlink that opens the summary screen of an entity record.
- **JumpKey**. Specifies the jump key for the **CustomActionFile** property.
- **OrderByDesc**. Sorts entries in the list in the descending order. To use this property, set **AllowOrderBy** to `true`.
- **ShowHeading**. Shows or hides the column heading.
- **ShowSelectAsGif**. Shows the values in the column as GIF image.
- **Visible**. Shows or hides a column when the **CRMGridColBlock object** is executed.

Alignment

Sets the alignment of text within the column.

Values

This property can take one of the following values:

- **Left** (default)
- **Right**
- **Center**

Examples

```
CaseListBlock = CRM.GetBlock('CaseListBlock');
Source = CaseListBlock.AddGridCol('Case_Source');
Source.AllowOrderBy = true;
Source.Alignment = 'right';
CRM.AddContent(CaseListBlock.Execute());
Response.Write(CRM.GetPage());
```

Aligns the text in the **Source** column to the right.

AllowOrderBy

Sorts entries in the list by the values in the column. By default, the entries are sorted in the ascending order.

Values

- **true**. Enables sorting.
- **false**. Disables sorting.

Example

```
CaseListBlock = CRM.GetBlock('CaseListBlock');
FoundIn = CaseListBlock.AddGridCol('Case_FoundVer');
FoundIn.AllowOrderBy = true;
CRM.AddContent(CaseListBlock.Execute());
Response.Write(CRM.GetPage());
```

Adds a new column (`Case_FoundVer`) to the list and sorts entries in the list by that column.

CreateScript

Specifies the server-side JavaScript to run upon the creation of the CRMGridColBlock instance.

Values

- **String**. Specifies the JavaScript to run.

Example

```
CaseListBlock = CRM.GetBlock('CaseListBlock');
FoundIn = CaseListBlock.AddGridCol('pers_firstname');
FoundIn.AllowOrderBy = true;
FoundIn.OrderByDesc = true;
FoundIn.CreateScript = "if(CRM.GetContextInfo("company","comp_companyid")){Visible = false;}";
CRM.AddContent(CaseListBlock.Execute());
Response.Write(CRM.GetPage());
```

Adds a new column (`pers_firstname`) to the list. Sorts entries in the pers_firstname column in the descending order. Hides the **Company name** column if it is present in the context.

CustomActionFile

Hyperlinks a column to an ASP file.

When a user clicks an item in the column, the ASP file is called up, passing in the value of the field set in the **CustomIdField** property in the query string. This property is only applicable when the **JumpAction** property is set to `430`.

Values

Name of the ASP file.

Examples

```
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());
```

Sets the custom jump to the invoices ASP page.

Note that when you reference a value from the QueryString (or a form field), always reference the value rather than the object itself, for example:

```
ThisComp = Request.QueryString('comp_companyid');
```

or

```
a = Request.QueryString("field");
```

If there's a possibility of a QueryString field being duplicated, test its length and reassign the variable.

CustomIdField

Allows a value to be passed to the custom file when the corresponding column is selected. The value is passed to the query string in the following form:

`<FieldName> = <Value>`

This property is only applicable to columns for which **CustomActionFile** is set.

Values

Name of any field in the view for the list.

Examples

```
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());
```

Sets the ID field for the custom jump to the `comp_companyid` field.

JumpAction

Specifies the jump action for the **CustomActionFile** property.

Example

```
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());
```

Sets the ID field for the custom jump to the `comp_companyid` field. The `JumpAction` value specifies the action to perform. In this case, it is custompage (`430`) .

JumpEntity

Adds a hyperlink that opens the summary screen of an entity record.

The entity must be relevant to the list. The column of the entity must exist within the view or table on which the list is based. For example, it's possible to jump to an Opportunity from an Opportunity list but not from a Case list.

Values

This property can take one of the following values:

- company
- person
- communication
- case
- address
- library
- notes
- custom table

Examples

```
PersonList = CRM.GetBlock("persongrid");
GridCol = PersonList.GetGridCol("pers_firstname");
GridCol.JumpEntity = "person";
CRM.AddContent(PersonList.Execute(''));
Response.Write(CRM.GetPage());
```

Adds a hyperlink to the `pers_firstname` field. When a user clicks the hyperlink, the person's summary screen opens.

JumpKey

Specifies the dominant key of the entity to open.

Example

```
list = CRM.GetBlock('CompanyGrid');
g = list.GetGridCol('comp_name');
g.JumpAction = 430;
g.JumpKey = 1;
g.CustomIdField = 'comp_companyid';
g.CustomActionFile = 'test.asp';
CRM.AddContent(list.Execute("comp_name like 'o%'"));
Response.Write(CRM.GetPage());
```

Adds a new column (`pers_firstname`) to the list. Sorts entries in the `pers_firstname` column in the descending order. The `JumpKey` value specifies the dominant key (entity). In this example, it is Company (`1`).

OrderByDesc

Sorts entries in the list in the ascending or descending order. To use this property, set **AllowOrderBy** to `true`.

Values

- **true**. Sorts entries in the descending order: 9 to 0 and/or Z to A.
- **false** (default). Sorts entries in the ascending order: 0 to 9 and/or A to Z.

Example

```
CaseListBlock = CRM.GetBlock('CaseListBlock');
FoundIn = CaseListBlock.AddGridCol('pers_firstname');
FoundIn.AllowOrderBy = true;
FoundIn.OrderByDesc = true;
CRM.AddContent(CaseListBlock.Execute());
Response.Write(CRM.GetPage());
```

Adds a new column (`pers_firstname`) to the list and sorts entries in the list by that column in the descending order.

ShowHeading

Shows or hides the column heading.

Values

This property can take one of the following values:

- **true** (default). Shows the column heading.
- **false**. Hides the column heading.

Examples

```
CaseListBlock = CRM.GetBlock('CaseListBlock');
Source = CaseListBlock.AddGridCol('Case_Source');
Source.ShowHeading = false;
Source.Alignment = 'LEFT';
CRM.AddContent(CaseListBlock.Execute());
Response.Write(CRM.GetPage());
```

Adds a column `case_source` to the case list. The column heading is hidden.

ShowSelectAsGif

Shows the values in the column as GIF images.

This property is only applicable if the column type is **Select** and there are .gif files in the folder for each option in the list.

Values

This attribute can take one of the following values:

- **true**. Shows the values as GIF images.
- **false** (default). Shows the values as text.

Examples

```
GridCol.ShowSelectAsGif = true;
```

Shows the values in the column as GIF images.

Visible

Shows or hides a column when the **CRMGridColBlock object** is executed.

Values

- **true** (default). Shows a column.
- **false** Hides a column.

Example

```
CaseListBlock = CRM.GetBlock('CaseListBlock');
FoundIn = CaseListBlock.GetGridCol('pers_firstname');
FoundIn.Visible = false;
CRM.AddContent(CaseListBlock.Execute());
Response.Write(CRM.GetPage());
```

Hides the `pers_firstname` column.

Code example: CRMGridColBlock object

This example uses the CRMGridColBlock object to add and remove columns in a list and to edit the properties of columns in a list. The page is used in 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 alternative number as the third 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());

%>
```

CRMListBlock object

Use the CRMListBlock object to create and display lists.

CRMListBlock is a child of the **CRMBlock object** and parent of the **CRMGridColBlock object**. You can link the list block to a search box in the **CRMEntryGroupBlock object** and use the search result as the argument for the list block.

Preceding code:

```
ListBlock = CRM.GetBlock('companygrid');
```

- **CRMListBlock methods**
- **CRMListBlock properties**
- **Code example: CRMListBlock object**

CRMListBlock methods

- **AddGridCol(ColName, Position, AllowOrderBy)**. Adds a new grid column dynamically to a List block.
- **DeleteGridCol(ColName)**. Deletes the specifies column from the list.
- **Execute(Arg)**. Displays a list.
- **GetGridCol**. Returns a reference to the specified grid column.

AddGridCol(ColName, Position, AllowOrderBy)

Adds a new grid column dynamically to a List block.

The changes don't apply outside the ASP pages they're used in.

Parameters

- **ColName** (required). Specifies the name of the field to be added as a column. The field must be relevant to the List block. It must be available in the table or view on which the List block is based.
- **Position** (optional). Specifies the position at which to add the column.
  Possible values:
  - **<a positive integer>**. Specifies the index of the position at which to add the column.

    - **0**. Adds the column to the first position.
    - **-1** (default). Adds the column to the last position.
- **AllowOrderBy** (optional). Specifies if the entries in the column can be sorted. Possible values:
    - **true**. Specifies that the entries cannot be sorted.
    - **false** (default). Specifies that the entries can be sorted.

Examples

```
MyList = CRM.GetBlock('CompanyGrid');
MyList.AddGridCol('comp_revenue', -1, true);
CRM.AddContent(MyList.Execute());
Response.Write(CRM.GetPage());
```

Adds a new column comp_revenue to the end of the company grid list. The entries in the new column can be sorted.

DeleteGridCol(ColName)

Deletes the specifies column from the list.

Parameters

ColName. Specifies the name of the column to delete.

Examples

```
ListBlock = CRM.GetBlock("companygrid");
ListBlock.DeleteGridCol("comp_website");
CRM.AddContent(ListBlock.Execute());
Response.Write(CRM.GetPage());
```

Deletes the comp_website column in the company list.

Execute(Arg)

Displays a list.

Parameters

Arg. This parameter accepts values of the following types:

- **String**. This type of value is processed as the WHERE clause of an SQL statement.
- **Entrygroup**. This type of value is used to form the WHERE clause of an SQL satement.

Examples

```
ListBlock = CRM.GetBlock("companygrid");
CRM.AddContent(ListBlock.Execute("comp_type='Customer'"));
Response.Write(CRM.GetPage());
```

Lists all companies whose type is Customer.

```
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());
```

Uses the result of the entrygroup search as the argument for the list.

GetGridCol

Returns a reference to the specified grid column.

To set the properties of the column, use the **CRMGridColBlock properties**.

Parameters

GridColName. Specifies the column name.

Return value

- **CRMGridColBlock object**. Indicates that the specified column exists.
- **Nil object**. Indicates that the specified column doesn't exist.

Examples

```
ListBlock = CRM.GetBlock("companygrid");
Column = ListBlock.GetGridCol("comp_name");
Column.allowOrderby = true;
```

```
CRM.AddContent(ListBlock.Execute());
Response.Write(CRM.GetPage());
```

Returns the comp_name column and configures the list to be ordered by this column.

CRMListBlock properties

- **CaptionFamily**. Sets the caption family for a list.
- **PadBottom**. Shows or hides empty rows in a list.
- **prevURL**. Specifies the URL of the ASP page to return to.
- **RowsPerScreen**. Sets the maximum number of rows displayed on each screen.
- **SelectSql**. Specifies the SQL statement to select items in the list.

CaptionFamily

Sets the caption family for a list. As a result, translations can be added for the captions at the top of the list.

When this parameter is set, translations are added to the caption family using the following codes:

- **Name**. Caption family name.
- **NoRecordsFound**. Caption that is displayed when there are no entries in the list.
- **RecordsFound**. Caption that is displayed when entries are present in the list.
- **RecordFound**. Caption that is displayed when there is only one entry in the list.
- **PreRecordsFound**. Caption that is displayed before the number of records found.
- **PreRecordFound**. Caption that is displayed if only one entry is found.

Values

String. Specifies the caption family name.

Examples

```
MyList = CRM.GetBlock('CompanyGrid');
MyList.CaptionFamily = "Campaigns";
CRM.AddContent(MyList.Execute());
Response.Write(CRM.GetPage());
```

Changes the caption family of the company list to **Campaigns**.

PadBottom

Shows or hides empty rows in a list.

Values

This property can take one of the following values:

- **true**(default). Shows empty rows. In this case, the number of rows shown always equals the value set in the **RowsPerScreen** property.
- **false**. Hides empty rows.

Examples

```
List = CRM.GetBlock('companygrid');
List.RowsPerScreen = 8;
List.PadBottom = false;
CRM.AddContent(List.Execute());
Response.Write(CRM.GetPage());
```

Hides empty rows. The column heading is displayed even if there are no rows to display.

prevURL

Specifies the URL of the ASP page to return to.

Use this property if any columns in the List block have links to a main entity (such as Company, Person, Opportunity, Case, Lead, or Solution).

Values

Value. Specifies the ASP page URL.

Examples

```
&Key-1=iKey_CustomEntity&PrevCustomURL=PrevUrl
```

Specifies that the previous dominant key was a custom page and where to go back to.

RowsPerScreen

Sets the maximum number of rows displayed on each screen.

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. Each user has a Grid Size setting in their Preferences. This setting overrides the RowsPerScreen setting except where you're using the ListBlock in a **CRMSelfService object**.

Parameters

None

Examples

```
ListBlock = CRM.GetBlock("CompanyGrid");
ListBlock.RowsPerScreen = 8;
CRM.AddContent(ListBlock.Execute());
Response.Write(CRM.GetPage());
```

Displays a list of companies, 8 entries per screen.

SelectSql

Specifies the SQL statement to select items in the list.

This property is only applicable when the List block is not based on an existing grid or list; for example, when the List block is returned by the **GetBlock(BlockName)** method.

Value

String. Specifies the SQL SELECT statement.

Use the following syntax:

```
SELECT * FROM <TableName>
SELECT * FROM <ViewName>
```

Do not put anything after the table or view name. The WHERE statement is configured by the list.

Examples

```
NewList = CRM.GetBlock("list");
NewList.SelectSql = "Select * from vCompany";
NewList.AddGridCol("Comp_Name");
CRM.AddContent(NewList.Execute());
Response.Write(CRM.GetPage());
```

Displays a list of companies from the vCompany view.

Code example: CRMListBlock object

This example creates a case list from the company context, where the status and stage columns are removed if the user isn't on the Sales team. A FoundIn column is added if the user is on 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 user's team ID.
UserChannel = CRM.GetContextInfo('User','User_PrimaryChannelId');

// Remove fields if the user's team is not Sales.
// 1 in the code below is the Channel ID of the Sales team.
if (UserChannel != 1)
    {
        CaseListBlock.DeleteGridCol('Case_Status');
        CaseListBlock.DeleteGridCol('Case_Stage');
    }

// Add field for Development team.
// 5 in the code below is the Channel ID of the Operations team.
if (UserChannel == 5)
    {
        FoundIn = CaseListBlock.AddGridCol('Case_FoundVer');

        // Enable sorting by the Found In column.
        FoundIn.AllowOrderBy = true;
    }

// Execute the block, pass in the SQL clause.
CRM.AddContent(CaseListBlock.Execute(SearchSql));
Response.Write(CRM.GetPage());

%>
```

CRMMarqueeBlock object

Use the CRMMarqueeBlock object to add scrolling text to a page. For example, a news ticker. CRMMarqueeBlock is a child of **CRMBlock object**.

The Marquee block reads from the Custom Captions table for news headlines and 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 object provides a dismiss button which is overwritten when the news changes.

The news headlines and news stories must be created using CRM translation handling. You can configure translation handling in the **<My Profile> | Administration | Customization | Translations** area of Sage CRM.

The caption family for news headlines is 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 the associated headline.

You call the Marquee block from an ASP page as follows:

```
var Marq;
Marq = CRM.GetBlock('marquee');
Marq.VerticalMinimum = 150;
Marq.VerticalMaximum = 150;
Marq.HorizontalMinimum = 70;
Marq.HorizontalMinimum = 70;
CRM.AddContent(Marq.Execute());
Response.Write(CRM.GetPage());
```

- **CRMMarqueeBlock properties**

CRMMarqueeBlock properties

- **HorizontalMinimum**. Sets the start point of horizontal scrolling.
- **HorizontalMaximum**. Sets the end point of horizontal scrolling.
- **VerticalMinimum**. Sets the start point of vertical scrolling.
- **VerticalMaximum**. Sets the end point of vertical scrolling.
- **ScrollSpeed**. Sets the scrolling speed.
- **StyleSheet**. Specifies the Cascading Style Sheet (CSS) to use.

To implement...	Do this...
Vertical scrolling	• Set **VerticalMinimum** and **VerticalMaximum** to different values. • Set **HorizontalMinimum** and **HorizontalMaximum** to the same value.
Horizontal scrolling	• Set **HorizontalMinimum** and **HorizontalMaximum** to different values. • Set **VerticalMinimum** and **VerticalMaximum** to the same value.

HorizontalMinimum

Sets the start point of horizontal scrolling.

This is the point from which horizontal scrolling begins.

Values

Integer. The default value is 0.

Example

```
Marq = CRM.GetBlock('marquee');
Marq.HorizontalMinimum = 0;
```

Sets the start point of horizontal scrolling to 0.

HorizontalMaximum

Sets the end point of horizontal scrolling.

This is how far the Marquee block can be scrolled to the right of the screen.

Values

Integer. The default value is 800.

Example

```
Marq = CRM.GetBlock('marquee');
Marq.HorizontalMaximum = 850;
```

Sets the end point of horizontal scrolling to 850.

VerticalMinimum

Sets the start point of vertical scrolling.

This is the point from which vertical scrolling begins.

Values

Integer. The default value is 300.

Examples

```
Marq = CRM.GetBlock('marquee');
Marq.VerticalMinimum = 100;
```

Sets the start point of vertical scrolling to 100.

VerticalMaximum

Sets the end point of vertical scrolling.

This is how far the Marquee block can be scrolled to the bottom of the screen.

Values

Integer. The default value is 300.

Examples

```
Marq = CRM.GetBlock('marquee');
Marq.VerticalMaximum = 350;
```

Sets the end point of vertical scrolling to 350.

ScrollSpeed

Sets the scrolling speed.

Values

Integer. The default value is 120.

Example

```
Marq = CRM.GetBlock('marquee');
Marq.ScrollSpeed = 200;
```

Sets the scrolling speed to 200.

StyleSheet

Specifies the Cascading Style Sheet (CSS) file to use.

You can use this property to change the appearance of the Marquee block.

Values

Text. Specifies the .css file name. The default value is DiagonalText. This must be a WideString value.

Examples

```
Marq = CRM.GetBlock('marquee');
Marq.StyleSheet = 'NewStyle.css';
```

CRMMessageBlock object

Use the CRMMessageBlock object to send email messages or SMS texts. CRMMessageBlock is a child of the **CRM object**. You can include the block in ASP pages to show a simple email form or to automate the message sent in response to an event. It can be used in visual and in hidden mode.

Syntax to initiate this block:

```
MessageBlock = CRM.GetBlock('messageblock');
```

- **Enabling CRMMessageBlock object**
- **CRMMessageBlock methods**
- **CRMMessageBlock properties**

Enabling CRMMessageBlock object

Messaging needs the following system components:

- An email server configured to redirect all incoming messages originating from a specified domain to the same folder.
- An SMS gateway referring to this folder and the related mobile phone connection.

To enable the **CRMMessageBlock object**, log on to Sage CRM as a system administrator, go to **<My Profile> | Administration | E-mail And Documents | E-mail Configuration**, and then configure the following options:

- **Outgoing Mail Server (SMTP)**. Specify the IP address of the SMTP server you want to use for sending email messages.
- **SMTP User Name**. Enter the user name of the account with which you want to access the SMTP server.
- **SMTP Password**. Enter the password for the specified access account.
- **SMTP Port**. Specify the SMTP port you want to use. By default, this is port 25.
- **SMS Domain Name**. Specify the mail domain used to hold the SMS messages, for example sms.domain.com.
- **SMTP Server For SMS Messaging**. Specify the SMTP server you want to use for sending SMS. You can specify a valid IP address or FQDN, such as mail.sms.domain.com.
- **Use SMS Feature**. Set this option to **Yes** to enable SMS messaging.
- **SMS From Address**. Specify a valid email address in the person profile.

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 email 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.

CRMMessageBlock methods

- **AddRecipient(Address, Name, MsgType)**. Adds a recipient to an email message.

AddRecipient(Address, Name, MsgType)

Adds a recipient to an email message.

Parameters

- **Address**. Specifies an email address to add.
- **Name**. Specifies the recipient name associated with the email address.
- **MsgType**. Specifies the email message field to add the email address to.

  Possible values:

  - **TO**. Adds the email address to the **To** field.
  - **CC**. Adds the email address to the **Cc** field.
  - **BCC**. Adds the email address to the **Bcc** field.

Examples

```
// Create a message block.
var myMailObject = CRM.GetBlock("messageblock");

// Display an email form and set an email subject.
myMailObject.DisplayForm = true;
myMailObject.mSubject = "My email subject";

// Set email body and display the Cc and Bcc fields.
myMailObject.mBody = "My email body";
myMailObject.mShowCC = true;
myMailObject.mShowBCC = true;

// Add email recipients to the To, Cc, and Bcc fields.
myMailObject.AddRecipient("training@sagecrm.com","Training","TO");
myMailObject.AddRecipient("support@sagecrm.com","Support","CC");
myMailObject.AddRecipient("manager@sagecrm.com","Manager","BCC");
```

```
17 | Response.Write(myMailObject.Execute());
```

Creates and displays an email form, sets an email subject, and adds recipients to the email message fields.

```
1  // Create a message block.
2  var myMailObject = CRM.GetBlock("messageblock");
3
4  // Hide the email form and set an email subject and body.
5  myMailObject.DisplayForm = false;
6  myMailObject.mSubject = "My email subject";
7  myMailObject.mBody = "My email body";
8
9  // Add a recipient to the To field.
10 myMailObject.AddRecipient("training@sagecrm.com","Training","TO");
11
12 // Send the message immediately.
13 myMailObject.Mode = 2;
14
15 // Write the operation output.
16 Response.Write(myMailObject.Execute());
17
18 // Record whether the message has been sent successfully.
19 if (myMailObject.mSentOK)
20     {
21         Response.Write("Email was sent successfully.");
22     }
23
24 else
25     {
26         Response.Write("An error has occurred. Details: "+myMailObject.mErrorMessage);
27     }
```

Creates a hidden email form, adds a recipient to the **To** field, sends the email, and records the operation output.

CRMMessageBlock properties

- **DisplayForm**. Shows or hides an email form.
- **mAddressFrom**. Sets sender's email address.
- **mNameFrom**. Sets sender's name.
- **mBody**. Sets the body text of the message.
- **mErrorMessage**. Displays the related error if the message isn't sent successfully.
- **mSentOK**. Displays the status of the sent message.
- **mShowCC**. Shows or hides the CC field (carbon copy) in the Sage CRM user interface.
- **mShowBCC**. Shows or hides the BCC field (blind carbon copy) in the Sage CRM user

interface.

- **mSubject**. Sets the subject of the message.

DisplayForm

Shows or hides an email form.

If the message contains errors the form is displayed regardless of the value set in this parameter.

Values

- **true** (default). Shows the email form.
- **false**. Hides the email form.

Example

```
var MailObj;
MailObj = CRM.GetBlock("messageblock");
MailObj.Mode = 2;
MailObj.DisplayForm = false;
CRM.AddContent(MailObj.Execute());
Response.Write(CRM.GetPage());
```

Sends a message without displaying the email form. Displays the email form only if there are errors.

mAddressFrom

Sets sender's email address.

This property is used in the Self Service mode when the user is logged in and the email address is retrieved from the current user details.

Values

Valid email address.

Examples

```
MailObj.mAddressFrom = 'messagesender@domain.com';
MailObj.mNameFrom = 'George Smith';
```

Sets the email address and name of sender.

mNameFrom

Sets sender's name.

This property is used in the Self Service mode when the user is logged in and the name is retrieved from the current user details.

Values

Valid sender's name.

Examples

```
MailObj.mAddressFrom = 'messagesender@domain.com';
MailObj.mNameFrom = 'George Smith';
```

Sets the email address and name of sender.

mBody

Sets the body text of the message.

The maximum body length for SMS messages is 160 characters.

Values

Text

Example

```
MailObj = CRM.GetBlock("messageblock");
MailObj.mBody = 'This is message text.';
CRM.AddContent(mailObj.execute());
Response.Write(CRM.GetPage());
```

mErrorMessage

Displays the related error if the message isn't sent successfully.

Values

Text (read-only)

Examples

```
if(!mSentOK)
    {
        // If errors occurred then show the proper message.
        CRM.AddContent('Error: '+mErrorMessage);
    }

else
    {
        CRM.AddContent('Message Sent OK'());
    }

Response.Write(CRM.GetPage());
```

Displays the error if the message isn't sent successfully.

mSentOK

Displays the status of the sent message.

Values

This parameter can take one of the following values:

- **true**. Indicates that the message is sent successfully.
- **false** (read-only). Indicates that message sending has failed and shows the corresponding error.

Examples

```
if(!MailObj.mSentOK)
    {
        // If an error occurs, then show the corresponding error message.
        CRM.AddContent('ERROR: ' + MailObj.mErrorMessage);
    }

else
    {
        CRM.AddContent('Message was sent successfully.');
    }

Response.Write(CRM.GetPage());
```

```
with (MailObj)
    {
        if(!mSentOK)
            {
```

```
                // If an error occurs, then show the corresponding error message.
                CRM.AddContent('ERROR: ' + mErrorMessage);
            }

        else {CRM.AddContent('Message was sent successfully.');
    }
```

The examples above display "Message was sent successfully" if the message was sent successfully. Otherwise, they display the corresponding error message.

mShowCC

Shows or hides the **CC** field (carbon copy) in the Sage CRM user interface.

Values

This property can take one of the following values:

- **true**. Shows the **CC** field.
- **false**. Hides the **CC** field.

Examples

```
MailObj = CRM.GetBlock("messageblock");
MailObj.mSubject = 'New Message';
MailObj.mBody = "This is where you put the content of the message.";
MailObj.mShowCC = true;
CRM.AddContent(mailObj.execute());
Response.Write(CRM.GetPage());
```

Shows the **CC** field in the Sage CRM user interface.

mShowBCC

Shows or hides the **BCC** field (blind carbon copy) in the Sage CRM user interface.

Values

This property can take one of the following values:

- **true**. Shows the **BCC** field.
- **false**. Hides the **BCC** field.

Examples

```
MailObj = CRM.GetBlock("messageblock");
MailObj.mSubject = 'New Message';
MailObj.mBody = "This is where you put the content of the message.";
MailObj.mShowBCC = true;
CRM.AddContent(mailObj.execute());
Response.Write(CRM.GetPage());
```

Shows the **BCC** field in the Sage CRM user interface.

mSubject

Sets the subject of the message.

Values

String

Examples

```
MailObj = CRM.GetBlock("messageblock");
MailObj.mSubject = 'My message subject';
CRM.AddContent(mailObj.execute());
Response.Write(CRM.GetPage());
```

Sets the subject of the message to "My message subject".

CRMOrgGraphicBlock object

The CRMOrgGraphicBlock object is an implementation of **CRMGraphicBlock object**. Use CRMOrgGraphicBlock for organizational charting. The most common use is to display an employee hierarchy for a company. You can pass data to the diagram from an ASP page or from a table. You can set parameters to describe the look of the diagram. The organizational graphic is recreated every time it's requested and can therefore be based on real time data.

Syntax to initiate this block:

```
OrgGraph = CRM.GetBlock('orgchart');
```

CRMOrgGraphicBlock methods

OrgTree(Mode, Value). Adds parent and child items for the **CRMOrgGraphicBlock object** and sets the appearance of the organizational chart.

OrgTree(Mode, Value)

Adds parent and child items for the **CRMOrgGraphicBlock object** and sets the appearance of the organizational chart.

Syntax

```
OrgTree
('Add','<ParentName>,<Name>,<Child=true/false>,<URL>,<Description>,<Relationship>');
```

Parameters

- **Mode**. WideString.
- **Value**. WideString.

Examples

```
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 CRMPipelineGraphicBlock object is an implementation of the **CRMGraphicBlock object** that includes extra functionality. Use CRMPipelineGraphicBlock to create cross-sectional diagrams that represent data from an ASP page or table. 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 change when the user clicks them. The pipeline graphic is recreated every time it's requested and can therefore be based on real time data. It can also use all the features of the graphics block.

The default size of pipeline graphic is 600 pixels wide by 100 pixels high, but you can change it using the **Resize(Width, Height)** method.

Syntax to initiate this block:

```
MyObj = CRM.GetBlock('pipeline');
```

- **CRMPipelineGraphicBlock methods**
- **CRMPipelineGraphicBlock properties**

CRMPipelineGraphicBlock methods

- **AddPipeEntry(Name, Value, Description)**. Adds a pipe section to the pipeline diagram.
- **ChooseBackGround(Value)**. Sets the background of the pipeline diagram.
- **PipelineStyle(Mode, Value)**. Changes the appearance and size of individual sections of the pipeline graphic.

AddPipeEntry(Name, Value, Description)

Adds a pipe section to the pipeline diagram.

Parameters

- **Name**. Specifies the name of the pipe section to show in the pipeline legend. This parameter accepts a WideString value.
- **Value**. Specifies the percentage that the pipe section takes in the pipeline diagram. This parameter accepts an integer value.
- **Description**. Specifies the tooltip that appears when the user points to the pipe section.

- **Url**. Specifies the URL (such as web page or ASP page) to open when the user clicks the pipe section.

Examples

```
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)

Sets the background of the pipeline diagram.

Parameters

Value. Specifies the background image file to use. The default background is white. This parameter can take one of the following values:

- **1**. accpacblue.gif
- **2**. accpacwhite.gif
- **5**. listrow1gif
- **8**. lightpurplemarblebright.gif
- **14**. accpaccream.gif
- **15**. listrow2.gif

You can find these files in the following folder on the Sage CRM server:

<Sage CRM installation folder>\WWWRoot\Themes\Img\default\Backgrounds

Examples

```
Pipe.ChooseBackGround(8);
```

Uses the lightpurplemarblebright.gif file as the background for the pipeline diagram.

PipelineStyle(Mode, Value)

Changes the appearance and size of individual sections of the pipeline graphic. For example, you can change gradients or legends.

Parameters

Mode. WideString.

Example

```
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());
```

CRMPipelineGraphicBlock properties

- **Pipe_Summary**. Sets a summary for the pipe section in HTML format.
- **Selected**. Selects a pipe section.

Pipe_Summary

Sets a summary for the pipe section in HTML format.

When a user points to the pipe section, the summary is displayed to the right of the section. This property can be used to display a legend or description for the selected pipe section.

Parameters

Value. HTML code.

Example

```
Pipleline = CRM.GetBlock('pipeline');
Pipe = Pipleline.Selected(1);
Pipe.Pipe_Summary = '<table><td class=tablehead>Negotiating Selected (70)</td></table>';
```

Selected

Selects a pipe section.

Once a pipe section is selected, you can change the section style.

Parameters

Value. Specifies the number of the section to select.

Examples

```
Pipeline = CRM.GetBlock('pipeline');
Pipeline.Selected(1);
```

CRMQuery object

Use the CRMQuery Object to enter and execute SQL statements against a known system database. This can be either the system database or an external database that's known and connected to Sage CRM.

You can perform more powerful queries with CRMQuery than with the **CRMRecord object**. You can use it to execute SQL statements that return results, for example, SELECT statements or statements that don't return results, for example, DELETE statements.

Preceding code:

```
var Query;
Query = CRM.CreateQueryObj('Select * from tablename', 'databasename');
```

- **CRMQuery properties**

CRMQuery methods

- **ExecSql()**. Executes the SQL statement.
- **Next()**. Selects the next row or SELECT statement in the query.
- **NextRecord()**. Moves the specified query to the next record.
- **Previous()**. Selects the previous row or SELECT statement in the query.
- **SelectSql()**. Executes SQL statements.
- **BeginTrans()**. Starts transaction for the following SQL statements.
- **CommitTrans()**. Commits a transaction initiated by **BeginTrans()**.
- **RollbackTrans()**. Rolls back a transaction initiated by **BeginTrans()**.

ExecSql()

Executes the SQL statement.

Use this method to execute statements that do not return rows. For example, DELETE, INSERT, or UPDATE.

To execute statements that do return rows (SELECT statements), use **SelectSql()**.

Parameters

None

Examples

```
var sql;
sql = "UPDATE Company SET Comp_PrimaryUserID='"+AccountMgr+"'
WHERE "+" Comp_CompanyId="+Values('Comp_CompanyId');
CRM.ExecSql(sql);
```

Executes the SQL UPDATE statement.

```
var myQuery;
var myResult;
myQuery = CRM.CreateQueryObj("update company set comp_source = 'TEST'
where comp_source is null","");
myResult = myQuery.ExecSql();
CRM.AddContent("Number of updated records: "+myResult);Response.Write(CRM.GetPage());
```

Updates company records and shows the number of updated records.

Next()

Selects the next row or SELECT statement in the query.

Parameters

None

Examples

```
var Query;
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());
```

Returns the next row in the query that displays company identifiers and names.

NextRecord()

Moves the specified query to the next record.

Parameters

None

Examples

```
Query.NextRecord();
```

Previous()

Selects the previous row or SELECT statement in the query.

Parameters

None

Examples

```
Query.Previous();
```

SelectSql()

Executes SQL statements.

Use this method to execute statements that return rows (SELECT statements).

Parameters

None

Examples

```
var 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());
```

Displays the company identifier and name field from the selected SQL query until the end of the query.

BeginTrans()

Starts transaction for the following SQL statements.

This transaction must be closed either by **CommitTrans()** or **RollbackTrans()**.

Use this method to prevent dead locking when using table level scripts. Exercise extreme caution when using this method because all transactions must be closed properly. Use the coding practice showed in the example below.

Parameters

None

Examples

```
var updatequery;
updatequery = CRM.CreateQueryObj(sql);
try
    {
        updatequery.BeginTrans();
        updatequery.ExecSql();
        updatequery.CommitTrans();
    }
catch(ex)
    {
        updatequery.RollbackTrans();
    }
```

CommitTrans()

Commits a transaction initiated by **BeginTrans()**.

Use this method to prevent dead locking when using table level scripts. Exercise extreme caution when using this method because all transactions must be closed properly. Use the coding practice shown in the example below.

Parameters

None

Example

```
var updatequery;
updatequery = CRM.CreateQueryObj(sql);
try
    {
        updatequery.BeginTrans();
        updatequery.ExecSql();
        updatequery.CommitTrans();
    }
catch(ex)
    {
        updatequery.RollbackTrans();
    }
```

RollbackTrans()

Rolls back a transaction initiated by **BeginTrans()**.

Use this method to prevent dead locking when using table level scripts. Exercise extreme caution when using this method because all transactions must be closed properly. Use the coding practice shown in the example below.

Parameters

None

Examples

```
var updatequery;
updatequery = CRM.CreateQueryObj(sql);
try
    {
        updatequery.BeginTrans();
        updatequery.ExecSql();
        updatequery.CommitTrans();
    }
catch(ex)
    {
        updatequery.RollbackTrans();
    }
```

CRMQuery properties

- **Bof**. Indicates the beginning of query.
- **DatabaseName**. Specifies a database other than the default system database.

- **Eof**. Indicates the last row of query.
- **FieldValue**. Gets or sets individual fields in a query.
- **RecordCount**. Gets the number of records referred to by the **CRMQuery object**.
- **SQL**. Sets the SQL statement for the query.

Bof

Indicates the beginning of query.

Example

```
var comp;
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());
```

Displays the company name if it is not at the beginning of the query.

DatabaseName

Specifies a database other than the default system database.

If this parameter is omitted, the default system database is used.

Parameters

Name. Specifies the database name. This parameter accepts a string value.

Examples

```
var Query;
Query = CRM.CreateQueryObj('Select * from company', 'crm');
Query.SelectSQL();
Query.DatabaseName(crm);
```

Executes the SQL statement on a database named crm.

Eof

Indicates the end of query.

Example

```
var Query;
Query = CRM.CreateQueryObj("Select * from vCompany");
Query.SelectSql();
while (!Query.eof)
    {
        Response.Write(Query("comp_companyid") + " = " + Query("comp_name")+' ');
        Query.NextRecord();
    }
```

Displays the company identifiers and name fields from the selected SQL query until the end of the query.

FieldValue

Gets or sets individual fields in a query.

Parameters

FieldName. Specifies the field name.

Examples

```
var value;
value = Query.FieldValue("MyField");
```

Retrieves a field named MyField.

```
var value;
value=Query("MyField");
```

Retrieves a field named MyField

```
var 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());
```

Displays the company identifier and the name field from the selected SQL query.

RecordCount

Gets the number of records referred to by the **CRMQuery object**.

Parameters

None

Examples

```
var Query;
Query = CRM.CreateQueryObj("Select * from company");
Query.SelectSQL();
CRM.AddContent("There are " +Query.RecordCount+ " records.");
Response.Write(CRM.GetPage());
```

Displays a record count of all records in the Company table of the default database.

SQL

Sets the SQL statement for the 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()**).

Examples

```
var Query;
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());
```

Resets the SQL SELECT statement to query the Person table instead of the Company table.

CRMRecord object

The CRMRecord object represents records in a table. This object is an enumerator that returns the specified fields in a table.

CRMRecord contains a higher-level understanding of the columns than CRMQuery. Use CRMRecord properties and methods to manipulate information in columns and save any edits.

To return the record that you manipulate, use the **CreateRecord(TableName)** and **FindRecord (TableName, QueryString)** methods.

Examples of syntax to create the record object:

```
var record;
record = CRM.CreateRecord("cases");
```

```
var record;
record = CRM.FindRecord("cases","case_caseid=20");
```

- **CRMRecord methods**
- **CRMRecord properties**

CRMRecord methods

- **FirstRecord()**. Moves the record to point to the first record that matches the SQL passed in when the Record object was created.
- **NextRecord()**. Returns the next record, if any.
- **RecordLock**. Locks the current Record object.
- **SaveChanges()**. Saves changes made to the current record to the database.
- **SaveChangesNoTLS()**. Saves changes made to the current record in the database. Does not trigger any table-level scripts that exist for the table being updated.
- **SetWorkflowInfo(vWorkflowName, vWorkflowState)**. Saves a new record to a workflow.

FirstRecord()

Moves the record to point to the first record that matches the SQL passed in when the Record object was created.

When the Record object is created, it automatically points to the first record. Use this method to set it.

Example

```
var o;
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());
```

Finds companies whose names start with the letter o and displays the first company record.

NextRecord()

Returns the next record, if any.

Examples

```
var People;
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());
```

Displays a list of first and last names of people in the Person table.

RecordLock

Locks the current Record object.

If the record is already locked because someone else is using it, an error message is returned.

Locking is usually automatically handled by Container blocks. Use the RecordLock method only when the standard container locking functionality is disabled (by setting the relevant Container block property CheckLocks to false).

Parameters

None

Examples

```
var r;
r = CRM.FindRecord('company','comp_companyid=30');
CompBlock = CRM.GetBlock('CompanyBoxLong');
CompBlock.CheckLocks = false;
if (CRM.Mode == 1)
    {
        e = r.RecordLock();

        if (e != '')
            {
                // Keep in view mode.
                CRM.Mode = 0;
                CRM.AddContent(e+'');
            }
    }
CRM.AddContent(CompBlock.Execute(r));
Response.Write(CRM.GetPage());
```

Locks the record. If the record is already locked, displays an error message and places the record in view mode.

SaveChanges()

Saves changes made to the current record to the database.

This method refreshes the RecordObject to point back to the beginning of the selected record set. You can't use it on a RecordObject where the same RecordObject is used in the condition in a while loop. For a workaround, see example 2 below.

Examples

```
var Comp;
var block;
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());
```

Adds and saves a new record to the company table and displays in a list.

```
var companies;
var company;
companies = CRM.FindRecord("company","comp_name like 'Gate%'");
while (!companies.eof)
    {
```

```
        Response.Write(companies('comp_name')+'');
        Response.Flush();
        company = CRM.FindRecord('company', 'comp_companyid=' + companies.comp_companyid);
        company.comp_type = 'Member';
        company.SaveChanges();
        companies.NextRecord();
    }
Response.Write('End');
```

Demonstrates a workaround for using the SaveChanges() method in a loop.

SaveChangesNoTLS()

Saves changes made to the current record in the database. Does not trigger any table-level scripts that exist for the table being updated.

Example

```
var Comp;
var block;
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());
```

Adds and saves a new record to the company table. Does not trigger the company table-level script.

SetWorkflowInfo(vWorkflowName, vWorkflowState)

Saves a new record to a workflow.

This method works when the **CRMRecord object** is retrieved by using the **CreateRecord (TableName)** method.

You can use the SetWorkflowInfo(vWorkflowName, vWorkflowState) method in one of the following cases:

- When in the **ArgObj** property of an **CRMEntryGroupBlock object**.
- When the **CRMRecord object** is passed to the **Execute(Arg)** method of an **CRMEntryGroupBlock object**.

Parameters

- **vWorkflowName**. Specifies a description of the workflow to which the record is saved. This is the workflow description entered when the workflow was created.

- **vWorkflowState**. Specifies the state in the workflow in which the record is to be saved. This is the State Name value entered when the workflow state was created.

Examples

```
var NewOppo;
NewOppo = CRM.CreateRecord("Opportunity");
NewOppo.SetWorkflowInfo("SalesOpportunityWorkflow","Lead");
NewOppo.Item("oppo_description") = "My new Oppo";
NewOppo.SaveChanges();
```

Creates a new opportunity, saves it to the SalesOpportunityWorkflow, and assigns "In Progress" state to the opportunity. When the opportunity is viewed, the valid actions for the assigned state are available.

CRMRecord properties

- **DeleteRecord**. Marks a record for deletion.
- **Eof** Indicates whether the last record has been reached.
- **IdField**. Returns the name of the ID field (also known as primary key) for the current table of the **CRMRecord object**.
- **Item**. Gets or sets the field value in its native format.
- **ItemAsString**. Gets the field value as a string.
- **OrderBy**. Specifies the field name by which to order record objects.
- **RecordCount**. Gets the number of records referred to by the object.
- **RecordID**. Gets the unique ID of the current record.
- **XML**. Generates the result as XML.

DeleteRecord

Marks a record for deletion.

The delete operation does not apply to the children of the record. As a result, the deletion of a record may leave some orphaned child records. To identify orphaned records, run the following SQL statement using either the **CRMQuery object** or **CRMRecord object**:

```
select bord_name, bord_companyupdatefieldname from custom_tables where bord_
companyupdatefieldname is not null
```

Values

This property can take one of the following values:

- **true**. Marks the record for deletion. The deletion occurs when the **SaveChanges()** method is called.
- **false**. Specifies that the record will not be deleted when the **SaveChanges()** method is called.

Examples

```
var Comp;
Comp = CRM.FindRecord('company', "comp_name = 'Eurolandia'");
Comp.DeleteRecord = true;
Comp.SaveChanges();
```

Deletes a record representing the Eurolandia company in the company table.

Eof

Indicates whether the last record has been reached.

Return values

- **true**. Indicates that the last record has been reached or there are no records.
- **false**. Indicates that the last record hasn't been reached yet.

Examples

```
while (!record.eof)
    {
        record.NextRecord();
    }
```

Retrieves the next record until the last record is reached.

IdField

Returns the name of the ID field (also known as primary key) for the current table of the **CRMRecord object**.

Normally, this is the first field name in the table.

Example

```
var Comp;
```

```
var idname;
Comp = CRM.FindRecord('company', "comp_name = 'Design Right Inc.'");
idname = Comp.IdField;
CRM.AddContent(Comp.item(idname));
Response.Write(CRM.GetPage());
```

Returns the ID field of a company named Design Right Inc.

Item

Gets or sets the field value in its native format.

Parameters

FieldName. Specifies the name of the field to get or set as the column in the table.

Example

```
record.Item("item");
```

```
record("item");
```

The two examples above perform the same action.

```
var Comp;
var Block;
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());
```

Creates a new record in the Company table, names the company "3D Communications International", and displays it in a company list.

ItemAsString

Gets the field value as a string.

This property uses metadata to convert the native field value to a string.

Parameters

FieldName. Specifies the name of the field.

Examples

```
var Case;
Case = CRM.FindRecord('cases', "case_assigneduserid=5");
CRM.AddContent(Case.itemasstring("case_assigneduserid"));
Response.Write(CRM.GetPage());
```

Finds and displays the name of the user assigned to a case from userid.

OrderBy

Specifies the field name by which to order record objects.

This property accepts a string value. The specified value is used to build up an SQL statement for the record. Use ASC or DESC parameters to order record objects in the ascending or descending order.

Examples

```
var People;
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());
    }
```

Orders person records in descending order by the Pers_LastName and Pers_FirstName fields.

RecordCount

Gets the number of records referred to by the object. This property returns an integer value.

Examples

```
var Users;
Users = CRM.FindRecord('users','');
CRM.AddContent("There are " + Users.RecordCount+ " system users.");
Response.Write(CRM.GetPage());
```

Displays the number of current system users.

RecordID

Gets the unique ID of the current record. The unique ID is created automatically when the record is created.

Examples

```
Response.Write(Record.RecordID);
```

Gets the unique ID of the current record.

```
var Record;
Record = CRM.FindRecord("company","");
CRM.AddContent(Record("comp_name"));
CRM.AddContent(Record.RecordID);
Response.Write(CRM.GetPage());
```

Displays the name and identifier of the current company record.

XML

Generates the result as XML.

Example

```
var intRecordId = CRM.GetContextInfo("company","comp_companyid");
var myRecord = CRM.FindRecord("company, vsummarycompany","comp_companyid="+intRecordId);
Response.Write(myRecord.xml);
```

CRMSelfService object

The CRMSelfService object provides access to the CRM database and to many CRM object methods, from outside the CRM application. Use it in a web application to allow visitors to your website access, and interact with, aspects of your CRM system.

For example, visitors to your website could log cases or directly update their contact information. These visitors don't have to be CRM users, but could be People in your CRM database.

CRMSelfService object enables authenticated and anonymous visitors to access varying levels of CRM data in View Only mode.

You can install a sample Self Service application as part of the CRM setup if your CRM license key includes Self Service. For more information about this application, see the Self Service section in the Sage CRM *Administrator Help* and *User Help*.

Although Self Service is a COM-based API, it's separate to the main ASP API for building Application Extensions, and uses blocks in a different way. The Self Service environment doesn't have a logon that generates a CRM Session ID (SID) or context. So you can't use API objects or methods that rely on a SID to build URLs. For example, CRM.Button(), CRM.GetTabs(), and CRM.URL().

The following syntax results in an error:

```
CRM.AddContent(myBlock.Execute(Arg));
Response.Write(CRM.GetPage());
Response.Write(myBlock.Execute(Arg));
```

- **CRMSelfService methods**
- **CRMSelfService properties**

CRMSelfService methods

- **EndSSSession(QueryString, ContentString, Cookie)**. Terminates the Self Service session and resets the Sage CRM cookies.
- **Init(QueryString, ContentString, Cookie)**. Initializes the Self Service session and Sage CRM cookies.

EndSSSession(QueryString, ContentString, Cookie)

Terminates the Self Service session and resets the Sage CRM cookies.

Parameters

- **QueryString**. Specifies querystring for the current page. `Request.Querystring`.
- **ContentString**. Specifies form string for the current page. `Request.Form`.
- **Cookie**. Specifies a reference to the Sage CRM cookies object. `Request.Cookies("CRM")`.

Examples

```
CRM.EndSSSession(Request.Querystring, Request.Form, Request.Cookies("CRM"));
Response.Write("The following user has been logged out:
"+CRM.VisitorInfo("visi_FirstName")+" "+CRM.VisitorInfo("visi_LastName"));
```

Terminates the Self Service session, resets Sage CRM cookies, and displays a log out message to the user.

Init(QueryString, ContentString, Cookie)

Initializes the Self Service session and Sage CRM cookies.

The **CRMSelfService object** must be initialized and connected to the database before it can be used.

If you've installed the CRM Self Service Demo site, you'll see the Init method called in the ewaress.js file. This file can be included at the top of each of your Self Service ASP pages by using the following syntax:

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

Parameters

- **QueryString**. Specifies the query string.
- **ContentString**. Specifies the content string, usually a form.
- **Cookie**. Specifies the initial Sage CRM cookie.

Examples

```
var CRM;
CRM = Server.CreateObject("eWare.eWareSelfService");
CRM.init(Request.Querystring,Request.Form,Request.Cookies("CRM"));
Response.Expires = -1;
```

Initializes the **CRMSelfService object** and Sage CRM cookies.

CRMSelfService properties

- **Authenticated**. Gets a value indicating whether the current user is authenticated.
- **AuthenticationError**. Sets an authentication error to display.
- **VisitorInfo**. Gets or sets the value associated with a key for the current authenticated visitor.

Authenticated

Gets a value indicating whether the current user is authenticated.

Parameters

None

Return value

- **true**. Indicates that the user is authenticated.
- **false**. Indicates that the current user is not authenticated.

Examples

```
if (CRM.Authenticated)
    {
        // This could be any function.    getmembermenu();
    }
else
    {
        Response.Redirect("index.asp");
    }
```

Allows an authenticated user to access a member menu. Takes unauthenticated users back to an index page.

AuthenticationError

Sets an authentication error to display.

Parameters

None

Examples

```
if (CRM.Authenticated)
    {
        // Perform action for authenticated users
    }

else
    {
        Response.Write('You are not a valid user' + CRM.AuthenticationError);
    }
```

VisitorInfo

Gets or sets the value associated with a key for the current authenticated visitor.

The key can be a column on the visitor table beginning with "Visi", or any text.

Parameters

Key. Specifies either a column in the visitor table or a string value.

Examples

```
if((CRM.Authenticated)&&(CRM.VisitorInfo("Visi_NotificationCriteria")!=""))
    {
        // This could be any method.getmembermenu();
    };
```

Grants access to any authenticated visitor who has submitted any notification criteria.

CRMTargetLists object

Use the CRMTargetLists object to create and save a target list in conjunction with CRMTargetListFields and CRMTargetListField. The target list must be based on a Company, Person, or Lead.

In Sage CRM version 7.2 and later, target lists are called **groups**. To ensure that legacy code works with new installations, the term **target lists** is maintained in the API terminology.

- **CRMTargetLists methods**
- **CRMTargetLists properties**
- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

CRMTargetLists methods

- **Save()**. Saves the Target list.
- **Include(ATargetID)**. Includes a target in the Target list.
- **Exclude(ATargetID)**. Excludes a target from the Target list.
- **Retrieve()**. Retrieves a target from the Target list.

Save()

Saves the Target list.

If the **TargetListID** property is set to zero, a new Target list is saved. Otherwise, the Target list specified in the **TargetListID** property is updated.

Parameters

None

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

Include(ATargetID)

Includes a target in the Target list.

Parameters

None

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

Exclude(ATargetID)

Excludes a target from the Target list.

Parameters

ATargetID. Specifies the ID of the target to exclude. This parameter accepts an integer value.

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

Retrieve()

Retrieves a target from the Target list.

Parameters

None

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

CRMTargetLists properties

- **Category**. Specifies the category of the Target list.
- **Description**. Specifies the description of the Target list.
- **Fields**. Specifies the list of display fields.
- **GroupAccessLevel**. Specifies who can access the Target list.
- **IsFixedGroup**. Specifies whether the Target list is a fixed group or a dynamic group.
- **IsNewGroupFromFind**. Specifies whether the Target list is a group or a saved search from Find.
- **Name**. Specifies the name of the Target list.
- **OrderByFields**. Specifies the list of order by fields.
- **TargetListID**. Specifies the ID of the Target list.
- **ViewName**. Specifies the view used by the Target list.
- **WhereClause**. Filters the list of targets.

Category

Specifies the category of the Target list.

Values

String

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

Description

Specifies the description of the Target list.

Values

String

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

Fields

Specifies the list of display fields.

Value

CRMTargetListFields object (read-only)

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

GroupAccessLevel

Specifies who can access the Target list. This property is applicable only if the Target list is a group; for more information, see **IsNewGroupFromFind**

Values

This property can take one of the following values:

- **<a positive integer>**. Specifies the ID of the user who can access the group.
- **0**. Specifies that all users, info managers, and system administrators can access the group.
- **-1**. Specifies that only info managers and system administrators can access the group.

Examples

See **Example: Creating and saving a Target list**.

IsFixedGroup

Specifies whether the Target list is a fixed group or a dynamic group. This property is applicable only if the Target list is a group; for more information, see **IsNewGroupFromFind**

Values

This property can take one of the following values:

- **true**. The group is fixed.
- **false**. The group is dynamic.

Examples

See **Example: Creating and saving a Target list**.

IsNewGroupFromFind

Specifies whether the Target list is a group or a saved search from Find.

Values

This property can take one of the following values:

- **true**. The Target list is a group.
- **false**. The Target list is a saved search.

Examples

See **Example: Creating and saving a Target list**.

Name

Specifies the name of the Target list.

Values

String

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

OrderByFields

Specifies the list of order by fields.

Value

CRMTargetListFields object (read-only)

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

TargetListID

Specifies the ID of the Target list.

Values

Integer

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

ViewName

Specifies the view used by the Target list.

Values

String

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

WhereClause

Filters the list of targets.

This property is mandatory when you create or modify a Target list.

Values

String

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

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 objectTargetBlock.
TargetListID = 0;

// Set the ID to zero, to indicate a new target listTargetBlock.
Category = "Person";

// Set the category. Other valid categories are Company and LeadTargetBlock.
Name = "COM List 1";

// Set the name of the target list, should be unique
TargetBlock.IsNewGroupFromFind = true;

// Set the target list to be a group rather than a saved search
TargetBlock.GroupAccessLevel  = 0;

// Enable all users to access the group
TargetBlock.IsFixedGroup = false;

// Specify that the group is dynamic
TargetBlock.ViewName = "vTargetListPerson";

// Set the view to be usedTargetBlock.
WhereClause = "Addr_City = N'London'";

// If required, 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 fieldTargetField.
DataField = "Comp_Name";

// Specify its database field name
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
listTargetField = TargetBlock.OrderByFields.New();

// Create a new order by field
TargetField.DataField = "Pers_LastName";

// Specify its database field name
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) && (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));
    }

%>
```

CRMTargetListField object

Use the CRMTargetListField objects to define fields to be included in a target list. You must specify the actual field names in the CRM database.

In Sage CRM version 7.2 and later, target lists are called **groups**. To ensure that legacy code works with new installations, the term **target lists** is maintained in the API terminology.

- **CRMTargetListField properties**

CRMTargetListField properties

DataField. Specifies the name of the field to display on the Target list.

DataField

Specifies the name of the field to display on the Target list.

Values

String

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

CRMTargetListFields object

Use the CRMTargetListFields object to set up a list of **CRMTargetListField** objects. There are two instances of the object: one for display fields and one for order fields.

In Sage CRM version 7.2 and later, target lists are called **groups**. To ensure that legacy code works with new installations, the term **target lists** is maintained in the API terminology.

- **CRMTargetListFields methods**
- **CRMTargetListFields properties**

CRMTargetListFields methods

- **New(CRMTargetListField)**. Creates and returns a new field.
- **Delete(Index)**. Deletes the specified field.

New(CRMTargetListField)

Creates and returns a new field.

Values

CRMTargetListField object

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

Delete(Index)

Deletes the specified field.

Values

Index. Specifies the index of the field to delete.

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

CRMTargetListFields properties

- **Parent**. Specifies the parent **CRMTargetLists object**.
- **Count**. Specifies the number of new fields in the list.
- **Item**. Returns the field specified by the index (an integer).

Parent

Specifies the parent **CRMTargetLists object**.

Value

CRMTargetLists object (read-only)

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

Count

Specifies the number of new fields in the list.

Values

Integer (read-only)

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

Item

Returns the field specified by the index (an integer).

Values

CRMTargetLists object (read-only)

Examples

See the following topics:

- **Example: Creating and saving a Target list**
- **Example: Retrieving a Target list**

Email object

Use the Email object to customize scripts deployed by E-mail Management. The Email object provides access to an email through its properties and methods.

The 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
```

For more information about E-mail Management, see the *System Administrator Help*.

- **Email methods**
- **Email object properties**

Email methods

- **Send()**. Sends an email using the contents of the Email object.
- **AddFile(Value)**. Adds a file as an attachment to the Email object.
- **Clear()**. Clears the contents of the Email object.
- **Header(Value)**. Returns any named header value from the email.

Send()

Sends an email using the contents of the Email object.

Parameters

None

Examples

```
email.Send();
```

AddFile(Value)

Adds a file as an attachment to the Email object.

Parameters

Value. Specifies the full path to the file to be attached.

Return value

- **true**. Indicates that the specified file exists.
- **false**. Indicates that the specified file doesn't exist.

Examples

```
email.AddFile('C:\MyFolder\MyWordFile.docx');
```

Clear()

Clears the contents of the Email object. Typically use this before you want to send a new email.

Parameters

None

Examples

```
email.Clear();
```

Header(Value)

Returns any named header value from the email. Returns a blank string if the header value does not exist.

Parameters

Value. Specifies the name of the header value to retrieve.

Examples

```
comm("comm_replyto") = email.Header("Reply_To");
```

Email object properties

- **Body**. Specifies the body text of the email message.
- **IsHTML**. Sets a format for the email message.
- **Subject**. Sets the email message subject.
- **Priority**. Sets the email message priority.
- **Recipients**. Gets the recipients of the email message specified in the **To** field.
- **SenderName**. Gets or sets the name of the email sender.
- **SenderAddress**. Gets or sets the email address of the sender.
- **DeliveryTime**. Specifies the time and date when the email message was delivered to the inbox.
- **Attachments**. Gets the email message attachments.
- **BCC**. Gets the email addresses in the **BCC** field of the email message.
- **CC**. Gets the email addresses in the **CC** field of the email message.

Body

Gets or sets the body text of the email message.

Values

String

Examples

```
comm("Comm_Email") = eMail.Body
```

IsHTML

Sets a format for the email message.

Values

- **true**. Specifies to use HTML format.
- **false**. Specifies to use text format.

Examples

```
eMail.IsHTML = true;
```

Sets the email message format to HTML.

Subject

Gets or sets the email message subject.

Values

String

Examples

```
comm("Comm_Note") = eMail.Subject;
```

Priority

Sets the email message priority.

Values

This property can take one of the following values:

- **0**. Low priority.
- **1**. Normal priority.
- **2**. High priority.

Examples

```
var prHigh;
phHigh = 2;
eMail.Priority = prHigh;
```

Sets email message priority to high.

Recipients

Gets the recipients of the email message specified in the **To** field.

Values

AddressList object

Examples

```
var MyAddressList;
var singleaddress;
MyAddressList = email.Recipients;

// Get the first email address from the list.
singleaddress = MyAddressList.Items(0).Address;
```

Returns the first email address from the **To** field.

SenderName

Gets or sets the name of the email sender.

Values

String

Examples

```
comm("comm_from") = "\""+eMail.SenderName + "\" " + "<" + eMail.SenderAddress + "> ";
```

SenderAddress

Gets or sets the email address of the sender.

Values

String

Examples

```
comm("comm_from") = "\""+eMail.SenderName + "\" " + "<" + eMail.SenderAddress + "> ";
```

DeliveryTime

Specifies the time and date when the email message was delivered to the inbox.

Values

None

Examples

```
var commdate;
commdate = new Date(eMail.DeliveryTime);
comm("Comm_datetime") = commdate.getVarDate();
```

Attachments

Gets the email message attachments.

Values

AttachmentList object

Examples

```
var myAttachments;
myAttachments = email.Attachments;
var myAttachment;myAttachment = eMail.Attachments.Items(1);
```

BCC

Gets the email addresses in the **BCC** field of the email message.

Values

AttachmentList object

Examples

```
var singleaddress;
singleaddress = email.BCC.Items(0).Address;
```

CC

Gets the email addresses in the **CC** field of the email message.

Values

AttachmentList object

Examples

```
var singleaddress;
singleaddress = email.CC.Items(0).Address;
```

MailAddress object

Use the MailAddress object to customize scripts deployed by E-mail Management. The MailAddress object provides access to an individual address from the **AddressList object**.

Use the following syntax to retrieve a MailAddress object:

```
var myaddress;
myaddress = eMail.CC.Items(1);
```

For more information about E-mail Management, see the System Administrator Guide or Help.

- **MailAddress properties**

MailAddress properties

- **Name**. Gets or sets the user-friendly name associated with the email address.
- **Address**. Gets or sets the email address.

Name

Gets or sets the user-friendly name associated with the email address.

Values

String

Examples

```
var FromName;
FromName = eMail.CC.Items(1).Name;
```

Address

Gets or sets the email address.

Values

String

Examples

```
var FromAddress;
FromAddress = eMail.CC.Items(1).Address;
```

MsgHandler object

Use the MsgHandler object customize scripts deployed by E-mail Management. It provides basic access to the **Email object** and functionality for the system. It's the top level object within E-mail Management and scripting. It's passed into the script at run time.

For more information on E-mail Management, see the *System Administrator Help*.

- **MsgHandler methods**
- **MsgHandler properties**

MsgHandler methods

- **Log(value)**. Sets the message to write to a log file in the Sage CRM installation folder.
- **MailAdmin(Subject, Body)**. Sends an email to the system administrator.
- **GetUniqueFileName(Path, FileName)**. Checks if the specified file exists.

Log(value)

Sets the message to write to a log file in the Sage CRM installation folder. This method is only available when the **Debug** property is set to **true**.

Parameters

String. Specifies the message to write.

Examples

```
MsgHandler.Debug = true;
MsgHandler.Log("testing application");
```

MailAdmin(Subject, Body)

Sends an email to the system administrator. This method uses the email address specified in the EmSe_AdminAddress in the custom_emailaddress table.

Parameters

- **Subject**. Specifies the email message subject. This parameter accepts a string value.
- **Body**. Specifies the email message body text. This parameter accepts a string value.

Examples

```
var sSubject;
var Body;
sSubject = "Unknown customer";
sBody = "An unknown customer attempted to mail the service";
MsgHandler.MailAdmin(sSubject,sBody);
```

GetUniqueFileName(Path, FileName)

Checks if the specified file exists.

Return value

If the file exists, this method returns the name of the file.
If the file does not exist, this method returns the next valid name for the file.

Parameters

- **Path**. Specifies the file path.
- **Filename**. Specifies the file name.

Examples

```
var NewName;
NewName = MsgHandler.GetUniqueFileName(libdir, AttItem.Name);
AttItem.SaveAs(NewName, libdir);
```

MsgHandler properties

- **Msg**. Returns the **Email object**.
- **Debug**. Sets whether to write messages sent via the **Log(value)** method to the log file.
- **EmailAddress**. Gets the email address of the service.

Msg

Returns the **Email object**. Do not access this parameter from the script, as it's already been passed in as the **Email object**.

Values

Email object

Examples

```
var myemail;
myemail = MsgHandler.msg;
```

Debug

Sets whether to write messages sent via the **Log(value)** method to the log file.

Values

This property can take one of the following values:

- **true**. Specifies to write the messages to the log file.
- **false**. Specifies not to write the messages to the log file.

Examples

```
MsgHandler.Debug = true;
```

Writes messages to the log file.

EmailAddress

Gets the email address of the service.

Values

String

Examples

```
var serviceaddress;
serviceaddress = MsgHandler.EmailAddress;
```

Component Manager methods

This section describes the methods you can use when modifying generated script files in the Component Manager. For more information about the Component Manager, see **Transferring customizations to another Sage CRM instance**.

The methods are grouped by the action they perform.

- **Add methods**
- **Copy methods**
- **Create methods**
- **Delete and Drop methods**
- **Get methods**
- **SearchAndReplace methods**
- **Other methods**

Add methods

- **AddCoachingCaptions method**. Adds a new coaching caption.
- **AddColumn method**. Adds a column to an existing table.
- **AddCustom_Captions method**. Adds or modifies translations.
- **AddCustom_ContainerItems method**. Adds blocks to a container.
- **AddCustom_Data method**. Adds or updates data in the specified table.
- **AddCustom_Databases method**. Adds links to external databases.
- **AddCustom_Edits method**. Adds or modifies the properties of an entity field.
- **AddCustom_Lists method**. Adds columns to a customized List group.
- **AddCustom_Relationship method**. Add entity relationships for SData provider.
- **AddCustom_Report method**. Creates a report.
- **AddCustom_ReportBand method**. Adds a report band.
- **AddCustom_ReportChart method**. Adds a chart to a report.
- **AddCustom_ReportField method**. Adds a report field.
- **AddCustom_ReportGroup method**. Adds a report group.
- **AddCustom_ScreenObjects method**. Adds a new custom screen object, such as list, screen, search screen, tab group, filter box, or block.
- **AddCustom_Screens method**. Adds or modifies a field on a screen.
- **AddCustom_Scripts method**. Adds a table- or entity-level script.
- **AddCustom_Tables method**. Adds a table.
- **AddCustom_Tabs method**. Adds or modifies a tab in a tab group.
- **AddLPCategory method**. Adds a dashboard category.
- **AddLPGadget method**. Adds a dashboard gadget.
- **AddLPLayout method**. Adds a dashboard.
- **AddLPUserLayout method**. Assigns a dashboard template to a user.
- **AddMessage method**. Sets the message to show to a user while the component is being installed.
- **AddProduct method**. Adds a product.
- **AddView method**. Adds a view.

AddCoachingCaptions method

Adds a new coaching caption.

Parameters

- **Coch_ActionID**. Specifies the action number of the caption.
- **Coch_CaptCode**. Specifies the caption code for the caption.

Return value

ID of the added coaching caption.

AddColumn method

Adds a column to an existing table.

Parameters

- **Col_TableName**. Specifies the name of the table to which the column is to be added.
- **Col_ColumnName**. Specifies the column name.
- **Col_Type**. Specifies the entry type for the column.
- **Col_Size**. Specifies the column size (in characters) for certain field types, for example, text fields.
- **Col_AllowNulls**. Specifies if the column can take null values. Can take one of the following values:
    - **True**. The column can take null values.
    - **False**. The column doesn't accept null values.
- **Col_IsUnique**. Specifies if the values in the column must be unique. Can take one of the following values:
    - **True** Values in the column must be unique.
    - **False**. Values in the column do not have to be unique.
- **Col_IsIdentity**. Indicates whether the column should be created as an auto-incrementing field. Boolean.

Return value

None

AddCustom_Captions method

Adds or modifies translations in Sage CRM.

Parameters

- **Capt_FamilyType**. Specifies the family type to which the caption belongs. For example, Tags.
- **Capt_Family**. Specifies the family for the caption.
- **Capt_Code**. Specifies the code for the caption. Captions are identified by their family and code.
- **Capt_Order**. Specifies the order that this should appear in.
- **Capt_US**. Specifies the US English translation.
- **Capt_UK**. Specifies the UK English translation.
- **Capt_FR**. Specifies the French translation.
- **Capt_DE**. Specifies the German translation.
- **Capt_ES**. Specifies the Spanish translation.
- **Capt_DU**. Specifies the Dutch translation.
- **Capt_JP**. Specifies the Japanese translation.
- **Capt_IntegrationID**. Optional. Specifies the identifier of the integration to which the caption belongs. Can only be used within the integration module.

Return value

ID of the added record.

AddCustom_ContainerItems method

Adds blocks to a container.

Parameters

- **Cont_Container**. Specifies the name of the container.
- **Cont_BlockName**. Specifies the name of the block.
- **Cont_Order**. SYSINT.
- **Cont_NewLine**. Set to 1 for new line and 0 for the same line.
- **Cont_Width**. Sets the block width.

- **Cont_Height**. Sets the block height.
- **Cont_Deleted**. Flag to indicate if record is deleted.

Return value

ID of the added record.

AddCustom_Data method

Adds or updates data in the specified database table.

Note that this method may not work to update certain custom tables, because updates to the foreign key are not set automatically. The affected custom tables include Custom_Edits, Custom_Views, Custom_ScreenObjects, Custom_List, Custom_ContainerItems, Custom_Tabs, and Custom_Screen.

Parameters

- **TableName**. Specifies the name of the table in which you want to add or update data.
- **TablePrefix**. Specifies the table prefix.
- **IdColumn**. Specifies the name of the table column that holds the ID.
- **Fields**. Specifies the fields to update. When specifying multiple fields, use a comma as a separator.
- **Values**. Specifies the values to assign to the fields. When specifying multiple values, use a comma as a separator. The list of values must match the list of fields provided in the Fields parameter. To insert the value into a string field, surround the value with double quotes.
  You can use the following variables in place of actual values:
  - **ISBLANK**. Inserts a blank space into a text field (identical to setting the value to "").
  - **ISNULL**. Marks column as `NULL`.
  - **ISNOW**. Inserts the current date and time up to the nearest second.
- **KeyFields**. Specifies the fields in the Fields parameter whose values you want to use to uniquely identify records. If a record with the specified ID exists, it will be updated. Otherwise, the record will be created.
  To specify a field, enter the index the field has in the Fields parameter. When specifying multiple fields, use a comma as a separator.

Return value

None

Example

```
AddCustom_Data('Custom_Captions', 'Capt', 'Capt_CaptionId',
'Capt_FamilyType,Capt_Family,Capt_Code,Capt_DU,CreatedDate',
'"Tags","ColNames","Comp_Name","Company",ISNOW', '1,2,3');
```

AddCustom_Databases method

Adds links to external databases.

Parameters

- **Cdbo_Description**. Specifies the database description.
- **Cdbo_AliasName**. Specifies the database alias.
- **Cdbo_UserName**. Specifies the user name of the account with which to connect to the database.
- **Cdbo_Password**. Specifies the password that matches the user name in the Cdbo_ UserName parameter.
- **Cdbo_Deleted**. Indicates if the record is deleted. Leave as null.
- **Cdbo_DriverName**. Specifies the driver to access the database.
- **Cdbo_ServerName**. Specifies the name of the server that hosts the database.
- **Cdbo_DatabaseName**. Specifies the database name.

Return value

ID of the added record.

AddCustom_Edits method

Adds or modifies the properties of an entity field.

Parameters

- **ColP_Entity**. Specifies the entity name.
- **ColP_ColName**. Specifies the field name.
- **ColP_EntryType**. Specifies the entry type. For a list of possible entry types, see **EntryType**.
- **ColP_DefaultType**. Specifies the default type of the column.

- **ColP_DefaultValue**. Specifies the default value, if any, for the default type of the column.
- **ColP_EntrySize**. Specifies the number of visible characters when editing the column.
- **ColP_LookUpFamily**. Specifies a string with the look-up family name (if applicable to the entry type).
- **ColP_LookUpWidth**. Specifies the column width.
- **ColP_Required**. Specifies if the column is required. This parameter can take one of the following values:
    - **Y**. The column is required.
    - **N**. The column is optional.
- **ColP_AllowEdit**. Specifies if the column is writable. This parameter can take one of the following values:
    - **Y**. The column is writable.
    - **N**. The column is read-only.
- **ColP_SearchDefaultValue**. Specifies the default search value (if applicable to the entry type).
- **ColP_System**. Specifies if the column is system. This parameter can take one of the following values:
    - **Y**. The column is system and does not appear on customization screens.
    - **N**. The column is non-system and is available on customization screens.

Return value

ID of the added record.

AddCustom_Lists method

Adds columns to a customized List group.

Parameters

- **GriP_GridNam**. Specifies the name of the grid into which to add the column.
- **GriP_Order**. Specifies the order in which the column appears in the grid.
- **GriP_ColName**. Specifies the column name.
- **GriP_AllowRemove**. Specifies if the column is removable. This parameter can take one of the following values:
    - **Y**. The column is removable.
    - **N**. The column is not removable.

- **GriP_AllowOrderBy**. Specifies if the column can be used to order the list. This parameter can take one of the following values:
    - **Y**. The column can order the list.
    - **N**. The column cannot order the list.
- **GriP_OrderByDesc**. Sets the default sort mode for ordering items in the column.
    - **Y**. The default mode is descending.
    - **N**. The default mode is ascending.
- **GriP_Alignment**. Sets the alignment for the text in the column. This parameter can take one of the following values:
    - **CENTER** (or **NULL**). Center text.
    - **LEFT**. Align text to the left.
    - **RIGHT**. Align text to the right.
- **GriP_Jump**. Specifies if the column can contain a hyperlink to another page.
- **GriP_ShowHeading**. Specifies if the column heading is displayed in the list. This parameter can take one of the following values:
    - **Y**. Shows the column heading in the list.
    - **N**. Hides the column heading in the list.
- **GriP_ShowSelectAsGif**. Specifies if this column should show as a GIF. This parameter can take one of the following values:
    - **Y**. Shows the column as a GIF.
    - **N** (or **NULL**). Doesn't show the column as a GIF.
- **GriP_CustomAction**. Specifies the name of page to hyperlink to if the **GriP_Jump** parameter is set to **Custom**.
- **GriP_CustomIdField**. Specifies the name of field to use as the ID field if the **GriP_Jump** is set to **Custom**.
- **GriP_DeviceID**. Specifies the device ID of the list. The device ID is taken from the Devices table.
- **Grip_CreateScript**. Allows you to enter a create script for a column. The default value is blank.

Return value

ID of the added record.

AddCustom_Relationship method

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.

Return value

AddCustom_Report method

Creates 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.

Return value

AddCustom_ReportBand method

Adds a report band.

Parameter

- **ReBa_ReportID**. Integer.
- **ReBa_DetailLevel**. Integer.
- **ReBa_CrossTabField**. String.
- **ReBa_DisplayOptions**. Integer.
- **ReBa_ViewName**. String.
- **ReBa_WhereClause**. String.
- **ReBa_DetailLinkField**. String.
- **ReBa_MasterLinkField**. String.

Return value

AddCustom_ReportChart method

Adds 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

Return value

AddCustom_ReportField method

Adds 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.

Return value

AddCustom_ReportGroup method

Adds 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.

Return value

AddCustom_ScreenObjects method

Adds a new custom screen object, such as list, screen, search screen, tab group, filter box, or block.

Parameters

- **CObj_Name**. Specifies the name of the custom screen object. This name must be unique.
- **CObj_Type**. Specifies the type of the custom object. This parameter can take one of the following values:
    - **List**
    - **Screen**
    - **SearchScreen**
    - **TabGroup**
    - **FilterBox**
    - **Block**
- **CObj_EntityName**. Specifies the name of the entity on which the custom object is based.
- **CObj_AllowDelete**. Specifies if a user can delete the custom object. This parameter can take one of the following values:
    - **Y**. A user can delete the custom object.
    - **N**. A user cannot delete the custom object.
- **CObj_Deleted**. Specifies if the custom object is deleted. This parameter can take one of the following values:
    - **0**. The custom object is not deleted.
    - **1**. The custom object is deleted.
- **CObj_TargetTable**. Specifies the table from which fields can be added to the custom object.
- **CObj_Properties**. Internal use only. Specifies the list of properties the custom object has. When specifying multiple properties use a comma as a separator.
- **CObj_CustomContent**. Allows you to enter a custom script for the custom object.
- **CObj_UseEntity**. Specifies the table from which fields can be added to the custom object.
- **CObj_TargetList**. Internal use only.
- **CObj_Ftable**. Internal use only.
- **CObj_FtableFCol**. Internal use only.
- **CObj_CheckNameOnly**. Optional. Specifies how to check the custom object during update operations. This parameter can take one of the following values:

- **TRUE**. Allows object updates by checking the object name only.
- **FALSE**. Specifies that object name and entity are checked during object updates.

Return value

ID of the added object.

AddCustom_Screens method

Adds or modifies a field on a screen.

Parameters

- **SeaP_SearchBoxName**. Specifies the name of the screen where you want to add or modify the field. This should match the **Cobj_Name** value specified in the **Custom_ ScreenObjects** table.
- **SeaP_Order**. Specifies the order in which the field appears on the screen.
- **SeaP_ColName**. Specifies the column name of the field.
- **SeaP_Newline**. Specifies if the field should appear on a new line. This parameter can take one of the following values:
  - **1**. The field appears on a new line.
  - **0**. The field appears on the same line.
- **SeaP_RowSpan**. Specifies the number of rows that the field should span on the screen.
- **SeaP_ColSpan**. Specifies the number of columns that the field should span on the screen.
- **SeaP_Required**. Specifies if the field is required.
  - **Y**. The field is required.
  - **N**. The field is optional.
- **SeaP_DeviceID**. Specifies the ID of the device that this screen is for. Look up from the Devices table.
- **SeaP_OnChangeScript**. Allows you to enter the client-side JavaScript to apply to the OnChange event of the field.
- **SeaP_ValidateScript**. Specifies the server-side script to run to validate the field.
- **SeaP_CreatedScript**. Specifies the server-side script to run when the field is created.
- **SeaP_Jump**. Specifies the location to open when the field is clicked.

Return value

Returns the ID of the added field.

Example

To add fields on a screen, you can also use the `AddEntryScreenField` method.

In the examples below, the `AddCustom_Screens` and `AddEntryScreenField` methods add the same fields.

Using AddCustom_Screens method:

```
AddCustom_Screens('GlobalLibraryFilterBox',1,'libr_filename',0,1,1,'N',0,'','','','');
```

Using AddEntryScreenField method:

```
EntryScreenName='GlobalLibraryFilterBox';
FieldOrder=1;
FieldName='libr_filename';
NewLine=false;
RowSpan=1;
ColSpan=1;
Required=false;
AddEntryScreenField();
```

AddCustom_Scripts method

Adds a table- or entity-level script.

Parameters

- **CScr_TableName**. Specifies the name of the table to which the script applies.
- **CScr_ScriptName**. Specifies the script name.
- **CScr_Order**. Specifies the 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**. Specifies the user name under which to run the script, if applicable.
- **CScr_ScriptType**. Specifies the script type. This parameter can take one of the following values:
    - **entity**. Specifies that the script is entity-level.
    - **entitywrb**. Specifies that the script is entity-level with rollback.
    - **tls**. Specifies that the script is table-level.
    - **tlsdetached**. Specifies that the script is detached table-level.

- **CScr_UseRollBack**. Specifies if rollback is available on the script. This parameter can take one of the following values:
  - **Y**. Rollback is available.
  - **N**. Rollback is not available.
- **CScr_ViewName**. Specifies the name of the view from which the script gets information.

Note that the `CScr_IsEntityScript` parameter of this method has been deprecated. Please use the `CScr_ScriptType` parameter instead.

Return value

ID of the added record.

AddCustom_Tables method

Adds a table.

Parameters

- **Bord_Caption**. Specifies the table caption.
- **Bord_System**. Specifies whether this is a system table. In the Sage CRM user interface, system tables cannot be viewed through **<My Profile> | Administration | Customization**.
  This parameter can take one of the following values:
  - **Y**. Specifies that the table is system.
  - **N**. Specifies that the table is non-system.
- **Bord_Hidden**. Specifies whether the table is hidden. In the Sage CRM user interface, hidden tables cannot be viewed through **<My Profile> | Administration | Customization**
  This parameter can take one of the following values:
  - **Y**. Specifies that the table is hidden.
  - **N**. Specifies that the table is visible. Flag to indicate if this table is to be hidden.
- **Bord_Name**. Specifies the table name.
- **Bord_Prefix**. Specifies the prefix to add to all the fields in the table.
- **Bord_IdField**. Specifies the name of the table field that holds the unique ID of each row.
- **Bord_PrimaryTable**. Specifies whether the table is primary and has territory security applied.
  This parameter can take one of the following values:
  - **Y**. Specifies that the table is primary.
  - **N** or (**NULL**). Specifies that the table is not primary.

- **Bord_ProgressTableName**. Specifies the name of the table used for workflow progress.
- **Bord_ProgressNoteField**. Specifies the name of the field used for workflow tracking notes.
- **Bord_WorkflowIdField**. Specifies whether the table has a <prefix>_WorkflowId field. This parameter can take one of the following values:
  - **Y**. The <prefix>_WorkflowId field is present.
  - **N**. The <prefix>_WorkflowId field is not available.
- **Bord_DatabaseId**. Specifies the ID of the database where the table resides. To look up this ID, use Custom_Databases. If the table resides in the main Sage CRM database, leave this parameter blank.

Return value

ID of the added record.

AddCustom_Tabs method

Adds or modifies a tab in a tab group.

Parameters

- **Tabs_Permission**. Internal use only. Should be set to **0**.
- **Tabs_PerLevel**. Internal use only. Should be set to **0**.
- **Tabs_Order**. Specifies the order in which the tab appears in the tab group.
- **Tabs_Entity**. Specifies the name of the tab group.
- **Tabs_Caption**. Specifies the tab caption.
- **Tabs_Action**. Specifies the tab action name as a string.
- **Tabs_Customfilename**. Specifies the name of the ASP page to use for this tab. Use this parameter if the **Tabs_Action** parameter is set to **CustomFile**.
- **Tabs_WhereSQL**. Specifies the SQL clause that restricts the appearance of the tab.
- **Tabs_Bitmap**. Specifies the bitmap to use for the tab.
- **Tabs_DeviceId**. Specifies the device ID. Use a device ID from the Devices table.
- **Tabs_SecurityEntity**. Specifies the entity that is used to determine whether a particular user has permissions to view or use the tab.
- **Tabs_Deleted**. Specifies whether the tab is deleted. This parameter can take one of the following values:

    - **1**. The tab is deleted.
    - **0**. The tab is not deleted.
- **Tabs_InButtonGroup**. Specifies if a tab group should be implemented. This parameter can take one of the following values:
    - **1**. A tab group should be implemented.
    - **0**. No tab group needs to be implemented.

Return value

ID of the added record.

AddLPCategory method

Adds a dashboard category.

Parameters

- **Name**. Specifies the dashboard category name.
- **ParentId**. Specifies the ID of the parent category.
- **Deleted**. Specifies whether the dashboard category is deleted. This parameter can take one of the following values:
    - **1**. The category is deleted.
    - **0**. The category is not deleted.

Return value

ID of the added category.

AddLPGadget method

Adds a dashboard gadget.

Parameters

- **Name**. Specifies the gadget name.
- **Description**. Specifies the gadget description.
- **GadgetType**. Specifies the gadget type.
- **LayoutXml**. Specifies the gadget layout XML data. Make sure to replace all gadget IDs with markers for the FinishLandingPage method.

- **DataBinding**. Specifies the CRM data source (such as report ID and entity ID) to be used with the gadget, in XML format.
- **LayoutId**. Specifies the ID of the dashboard to which the gadget belongs.
- **CategoryId**. Specifies the ID of the category to which the gadget belongs.
- **Deleted**. Specifies whether the gadget is deleted. This parameter can take one of the following values:
    - **1**. The gadget is deleted.
    - **0**. The gadget is not deleted.
- **SourceId**. Specifies the source gadget ID for the FinishLandingPage method.

Return value

AddLPLayout method

Adds a dashboard.

Parameters

- **Name**. Specifies the dashboard name.
- **Description**. Specifies the dashboard description.
- **CategoryId**. Specifies the category to which the dashboard belongs. If the dashboard has no category, set this parameter to **0**.
- **LayoutXml**. Specifies the dashboard layout XML data. Make sure to replace all gadget IDs with markers for the FinishLandingPage method.
- **LayoutType**. Specifies the layout type. Currently there is only one layout type available.
- **Deleted**. Specifies whether the dashboard is deleted. This parameter can take one of the following values:
    - **1**. The dashboard is deleted.
    - **0**. The dashboard is not deleted.
- **IsTemplate**. Specifies if the dashboard is a template.
- **TemplChannels**. Specifies the CRM teams to which the dashboard is assigned.
- **TemplUsers**. Specifies the CRM users to whom the dashboard is assigned.
- **SourceId**. Specifies the source dashboard ID for the FinishLandingPage method.

Return value

ID of the added dashboard.

AddLPUserLayout method

Assigns a dashboard template to a user.

Parameters

- **LayoutId**. Specifies the dashboard ID.
- **TemplateLayoutId**. Specifies the dashboard template ID.
- **UserId**. Specifies the user ID.
- **Deleted**. Specifies whether the dashboard template is deleted. This parameter can take one of the following values:
    - **1**. The dashboard template is deleted.
    - **0**. The dashboard template is not deleted.

Return value

AddMessage method

Sets the message to show to a user while the component is being installed.

Parameters

Ms_Message. Sets the message.

Return value

AddProduct method

Adds a product.

Parameters

- **Prod_Name**. String.
- **Prod_Description**. String.
- **Prod_ListPrice**. String.

- **Prod_Image**. String.
- **Prod_APR**. String.

Return value

AddView method

Adds a view.

Parameters

- **AViewName**. Specifies the view name.
- **AEntity**. Specifies the entity to which the view relates. For system and hidden entities, use the System entity.
- **ADescription**. Specifies the view description.
- **AViewScript**. Specifies a SQL query for the view.
- **ACanEdit**. Specifies whether the view can be edited. This parameter can take one of the following values:
    - **TRUE**. The view can be edited.
    - **FALSE**. The view cannot be edited.
- **ACanDelete**. Specifies whether the view can be deleted. This parameter can take one of the following values:
    - **TRUE**. The view can be deleted.
    - **FALSE**. The view cannot be deleted.
- **AReportsView**. Specifies whether the view can be used in reports. This parameter can take one of the following values:
    - **TRUE**. The view can be used in reports.
    - **FALSE**. The view cannot be used in reports.
- **ATargetsView**. Specifies whether the view can be used in groups. This parameter can take one of the following values:
    - **TRUE**. The view can be used in groups.
    - **FALSE**. The view cannot be used in groups.
- **ForceOverwrite**. Specifies whether to overwrite an existing view in the database when adding the new view, provided that these two views are different. This parameter can take one of the following values:

    - **TRUE**. Overwrites the existing view in the database.
    - **FALSE**. Keeps the existing view in the database.
- **ASearchView**. Specifies whether to include the new view in keyword searches. This parameter can take one of the following values:
    - **TRUE**. The view is included in keyword searches.
    - **FALSE**. The view is excluded from keyword searches.

Return value

Copy methods

- **CopyAndDropColumn method**. Modifies the properties of a column.
- **CopyAspTo method**. Copies an ASP file from one location to another.
- **CopyFile method**. Copies a file from one location to the other.

CopyAndDropColumn method

Modifies the properties of a column.

This method creates a new column, copies data from the source column to that new column, updates the new column properties according to the specified parameters, deletes the source column, and then assigns the name of the source column to the new column.

Parameters

- **Col_TableName**. Specifies the name of the table that contains the column to be modified.
- **Col_ColumnName**. Specifies the column name.
- **Col_Type**. Specifies the new data type of the column.
- **Col_Size**. Specifies the new size of the column.
- **Col_Allow_Nuls**. Specifies whether the column allows null values.

Return value

None

CopyAspTo method

Copies an ASP file from one location to another. Relative paths are allowed in the parameters of this method.

Parameters

- **CTo_FileName**. Specifies the source file path and name (copy from).
- **CTo_NewFileName**. Specifies the target file path and name (copy to).

Return value

Example

```
CopyAspTo('custompages\\Edit.asp','..\\custompages\\system\\Edit.asp');
```

CopyFile method

Copies a file from one location to the other.

Parameters

- **SourceFile**. Specifies the source file name (copy from).
- **TargetFile**. Specifies the target file name (copy to).

Return value

None

Create methods

- **CreateNewDir method**. Creates a new directory.
- **CreateTable method**. Creates a table in the database.

CreateNewDir method

Creates a new directory.

Parameters

DirName. Specifies the directory name.

Return value

None

CreateTable method

Creates a table in the database.

Created table has the following standard Sage CRM fields: CreatedBy, CreatedDate, UpdatedBy, UpdatedDate, TimeStamp, and Deleted.

Parameters

- **Cr_Tablename**. Specifies the table name.
- **Cr_Prefix**. Specifies the prefix to be added to every column in the table.
- **Cr_Identity**. Specifies the name of the identity column in the table. The name you specify must start with the prefix.
- **Cr_PrimaryTable**. Specifies whether the table is primary in Sage CRM. A primary table includes the security territory column. This parameter can take one of the following values:
    - **True**. Indicates that the table is primary.
    - **False**. Indicates that the table is not primary.
- **SystemTable**. Specifies whether the table is system. Accepts one of the following values:
    - **True**. Indicates that the table is system.
    - **False**. Indicates that the table is non-system.

- **HiddenTable**. Specifies whether the table is hidden. A hidden table is not accessible from **<My Profile> | Administration | Customization**. This parameter can take one of the following values:
    - **True**. Indicates that the table is hidden.
    - **False**. Indicates that the table is not hidden.
- **Cr_WorkflowIdField**. Specifies whether this table has a <prefix>_WorkflowId field.
- **Cr_ProgressTableName**. Specifies the name of the table used to set and track progress. For example, CaseProgress.
- **Cr_ProgressNoteField**. Specifies the name of the field used to store progress notes.
- **Cr_NoIDCol**. Specifies whether the table has an auto-incrementing field.

Return value

Delete and Drop methods

- **DeleteColumn method**. Deletes the specified table column.
- **DeleteCustom_Caption method**. Deletes the specified caption.
- **DeleteCustom_Captions method**. Deletes the specified caption family.
- **DeleteCustom_Field method**. Deletes the specified field from the system.
- **DeleteCustom_Screen method**. Deletes the specified screen.
- **DeleteCustom_ScreenObjects method**. Deletes screen objects.
- **DropConstraint method**. Deletes the specified constraint from a table in the database, such as foreign key.
- **DropTable method**. Deletes the specified table from the database.
- **DropView method**. Deletes the specified view from the database.

DeleteColumn method

Deletes the specified table column.

Parameters

- **TableName**. Specifies the name of the table that includes the column to be deleted.
- **ColumnName**. Specifies the column name.

To delete table columns, we recommend that you use the **DeleteCustom_Field method**.

Return value

None

DeleteCustom_Caption method

Deletes the specified caption.

Parameters

- **Capt_FamilyType**. Specifies the caption family type.
- **Capt_Family**. Specifies the caption family.
- **Capt_Code**. Specifies the caption code.

Return value

None

DeleteCustom_Captions method

Deletes the specified caption family.

Parameters

- **Capt_FamilyType**. Specifies the caption family type.
- **Capt_Family**. Specifies the caption family.

Return value

None

DeleteCustom_Field method

Deletes the specified field from the system.

Deleting a field by using this method removes the field from all screens, lists, reports, saved searches, notifications and other locations in the system.

Parameters

- **ATableName**.
- **AColumnName**.

Return value

None

DeleteCustom_Screen method

Deletes the specified screen.

This method deletes all items for the specified screen regardless of device.

Parameters

SeaP_SearchBoxName. Specifies the name of the screen to delete.

Return value

DeleteCustom_ScreenObjects method

Deletes screen objects.

Parameters

- **CObj_Name**. Specifies the name of the Custom_ScreenObject for which you want to delete data.
- **DeleteHeader**. Specifies whether to delete the Custom_ScreenObject record from the Custom_ScreenObjects table. This parameter can take one of the following values:
    - **True**. Specifies to delete the object record in the Custom_ScreenObjects table.
    - **False**. Keeps the object record in the Custom_ScreenObjects table, but deletes subitems from Custom_Lists, Custom_Screens, and Custom_ContainerItems.
- **CObj_DeviceID**. Specifies the device records to delete. This parameter can take one of the following values:
    - **<DeviceID>**. Specifies the ID of the device whose records are to be deleted.
    - **0**. Specifies to delete all records for every device.
    - **1**. Specifies to delete desktop information only.

Return value

DropConstraint method

Deletes the specified constraint from a table in the database, such as foreign key.

Parameters

- **AConstraintName**. Specifies the name of the constraint to delete from the database.
- **ATableName**. Specifies the name of the table that contains the constraint.

Return value

DropTable method

Deletes the specified table from the database.

Parameters

ATableName. Specifies the name of the table to delete.

Return value

DropView method

Deletes the specified view from the database.

Parameters

AViewName. Specifies the name of the view to delete.

Return value

Get methods

- **GetDLLDir method**. Returns the full path to the eware.dll file.
- **GetInstallDir method**. Returns the path to the Sage CRM installation folder.
- **Param method**. Returns the value of the specified parameter.

GetDLLDir method

Returns the full path to the eware.dll file.

Parameters

None

Return value

Full path to the eware.dll file as a string.

GetInstallDir method

Returns the path to the Sage CRM installation folder.

Parameters

None

Return value

Full path and name of the Sage CRM installation folder as a string.

Example

```
var x = GetInstallDir();
```

Gets the path to the Sage CRM installation folder and stores it in the x variable.

Param method

Returns the value of the specified parameter.

Parameters

Pr_SearchNam. Specifies the parameter name.

Return value

Parameter value.

SearchAndReplace methods

- **SearchAndReplaceCustomFile method**. Finds and replaces text in the specified file.
- **SearchAndReplaceInDir method**. Finds and replaces text in all files located in the specified folder.
- **SearchAndReplaceInFile method**. Finds and replaces text in the specified file located in <Sage CRM installation folder>\CustomPages.

SearchAndReplaceCustomFile method

Finds and replaces text in the specified file.

Parameters

- **Sr_FileName**. Specifies the full path and name of the target file.
- **Sr_StringToSearch**. Specifies the text to search for and replace.
- **Sr_ReplaceString**. Specifies the replacement text.

Return value

SearchAndReplaceInDir method

Finds and replaces text in all files located in the specified folder.

Parameters

- **ADirPath**. Specifies the full path and name of the target folder.
- **AStringToSearch**. Specifies the text to search for and replace.
- **AReplaceString**. Specifies the replacement text.

Return value

SearchAndReplaceInFile method

Finds and replaces text in the specified file located in <Sage CRM installation folder>\CustomPages.

Parameters

- **Sr_FileName**. Specifies the name of the target file.
- **Sr_StringToSearch**. Specifies the text to search for and replace.
- **Sr_ReplaceString**. Specifies the replacement text.

Return value

Other methods

- **FinishLandingPage method**. Replaces all the markers in layoutXml with new IDs collected during creation of dashboards and gadgets.
- **FileOpen method**. Returns values stored in the specified comma-separated values (.csv) or Microsoft Excel file.
- **ProgressScriptTransaction method**. Commits to the database what has already been processed and starts a new transaction.
- **QueryResultsToFile method**. Runs the specified SQL select statement and writes its result to a comma-separated values (.csv) file on the Sage CRM server.
- **RunSQL method**. Runs the specified SQL script.
- **TableExists method**. Checks whether the specified table exists in custom_table.

FinishLandingPage method

Replaces all the markers in layoutXml with new IDs collected during creation of dashboards and gadgets. Make sure to run this method after running any other interactive dashboard methods.

The interactive dashboard methods should be run in the following order:

1. AddLPCategory
2. AddLPLayout
3. AddLPGadget
4. AddLPUserLayout
5. FinishLandingPage

Return value

FileOpen method

Returns values stored in the specified comma-separated values (.csv) or Microsoft Excel file.

Parameters

AFileName. Specifies the full path and name of the .csv or Microsoft Excel file.

Return value

Returns the DataFile object that can be used to process the values in the file. The DataFile object has the following properties:

- **EOF**. Indicates if the end of the file has been reached. Can take one of the following values:
  - **True**. The end of the file has been reached.
  - **False**. The end of the file has not been reached.
- **FieldCount**. Returns the number of columns in the file based on the current row.

The DataFile object exposes the following methods:

- **NextRow**. Skips the file pointer on to the next row in the file. This method does not have any parameters.
- **GetField**. Returns the value in the given field for the current row.
  This method has the AIndex parameter, which indicates the column number to return the value for. When the AIndex is set to 0, it returns the value in the first column of the current row, and so on.

ProgressScriptTransaction method

Commits to the database what has already been processed and starts a new transaction.

Parameters

CommitOnError. This optional parameter commits database transactions even if an error has occurred.

Return value

None

QueryResultsToFile method

Runs the specified SQL select statement and writes its result to a comma-separated values (.csv) file on the Sage CRM server.

Parameters

- **FileName**. Specifies the full path and name of the .csv file to write to.
- **QueryString**. Specifies the SQL select statement.

Return value

A message providing information whether the operation completed successfully.

Example

```
QueryResultsToFile("c:\\test.csv","SELECT capt_family,
capt_code,capt_fr FROM custom_captions ORDER BY capt_family,
capt_order");
```

RunSQL method

Runs the specified SQL script.

You can use this method to run simple or complex SQL scripts, from inserting or updating a record to creating and deleting tables and views.

We recommend that you test your SQL script before running it in a production environment.

Parameters

Sql. Specifies the SQL script to run.

Return value

None

TableExists method

Checks whether the specified table exists in custom_table.

Parameters

TableName. Specifies the table name.

Return value

A Boolean value indicating whether the table exists.