



An Article on ACT! By Sage  
[By: ACT! By Sage Team METAOPTION]

## What is ACT! By Sage?

Act! By Sage is a complete customer management system Application. ACT! Helps individuals and small business owners work more effectively. With ACT!, you can easily access a complete integrated view of your contact relationship, impression contacts with your follow-up. Leave no task undone, and make informed decision to advance your business.

## Introduction

The ACT! Software Development Kit (SDK) contains an extensive collection of tools and samples to help you build powerful applications and services based on ACT! Framework technology. The ACT! SDK extends the functionality of ACT!, enables external applications to control ACT!, reads and writes to ACT! database tables, and adds auxiliary commands to the user interface to execute external programs.

The online Help provides information about the tools contained in the SDK. For a full list of documents included with the SDK with descriptions.

## Foundation for the ACT! Product Line

- Security
  - Authentication
  - Users and Team
  - Access Control
- Access to all of the ACT! Entities surrounding data level
- Business logic and features
- Schema metadata & modifications
- .Net Based
- Core SDK



## ACT! Extensibility Models

### • File Types

The ACT! database folder contains the ACT! database file (.ADF), the ACT! log file (.ALF) which is the internal transaction log for the ACT! data, and the indicator or "pointer" to the ACT! database files (.PAD). The .ADF and .ALF must always exist, and must exist in the same folder to comprise the SQL Server database.

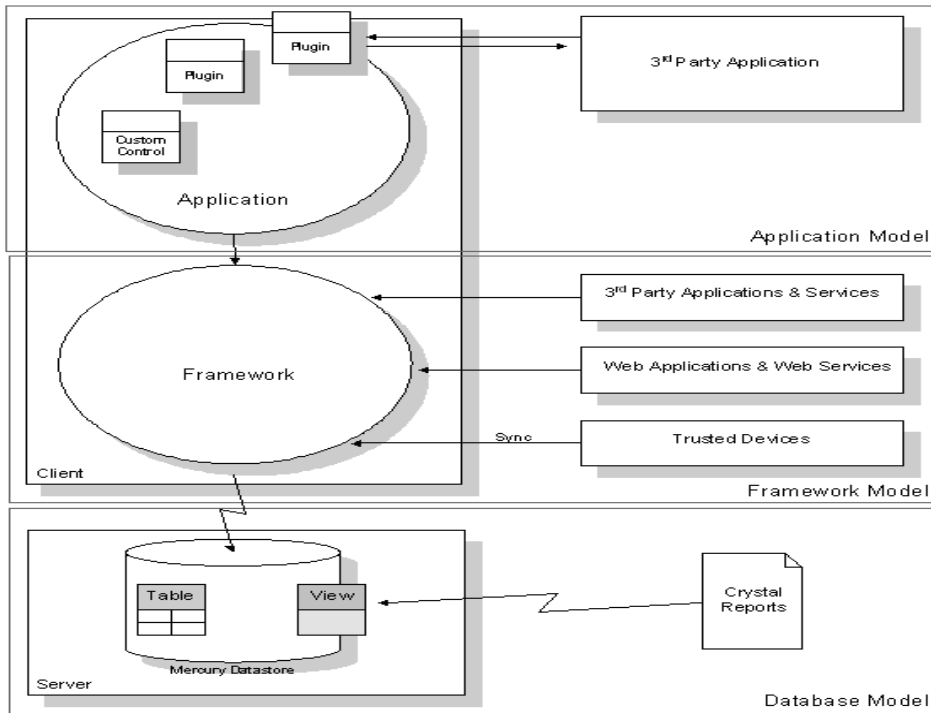
Another file type that is associated with the .adf is the remote database file (.RDF) this file is automatically created when a remote database is created for synchronization. The file itself is not a database, but contains the .adf file which is transported to a remote user. Think of the .rdf as a .zip file that must be unpacked and restored to a remote user's computer in order to use and synchronize ACT! data.

### • Act! Extensibility Model

ACT! is a highly extensible platform that allows rich customization and integration with other products. The ACT! Application empowers users to customize the software

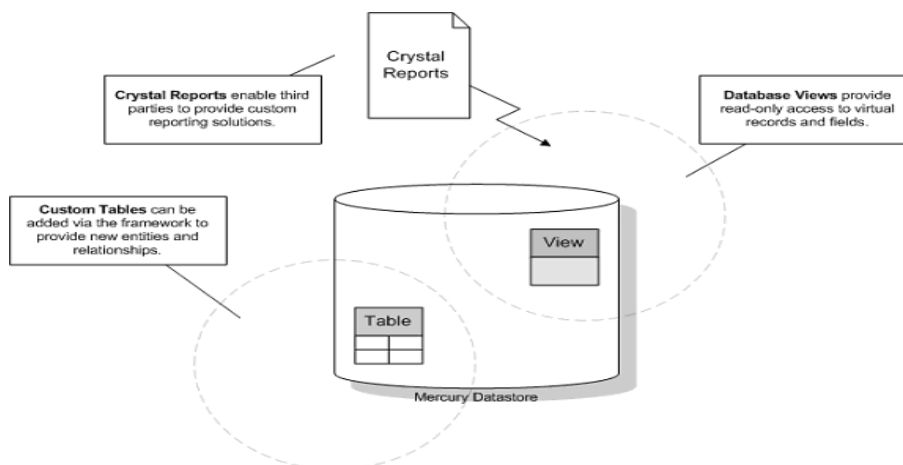
according to their needs, and enables third party developers to extend that courtesy to their clientele and other markets.

ACT! is comprised of three major components: The **database**, the **framework**, and the **application**. Each of these components allow developers to access and customize data, and to integrate with, automate, extend, and replace portions of the ACT! application. The unique needs and interactions of third party developers largely determine which ACT! integration paths to take.



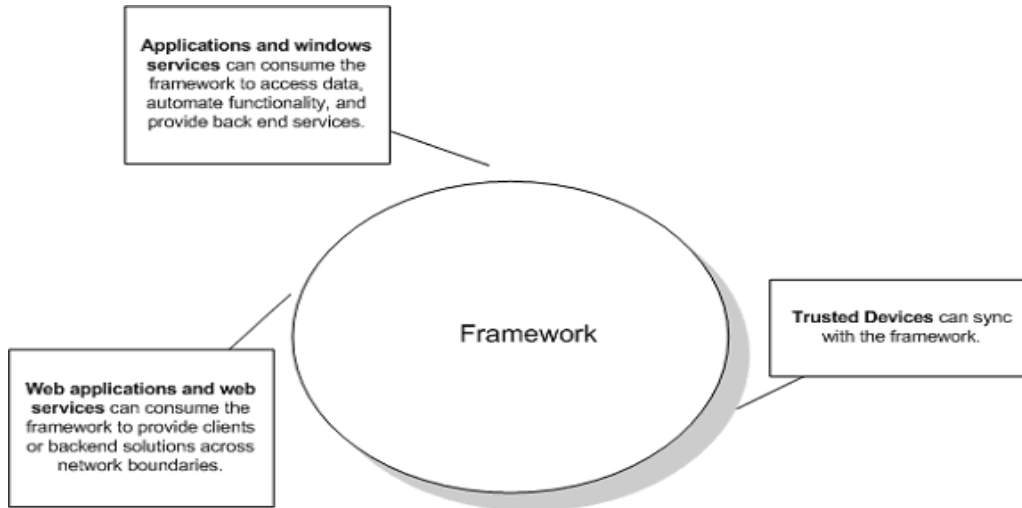
- **Database Extensibility Model**

The ACT! database contains all of the client data and the integrity of the information. There are several extensible points to the database:



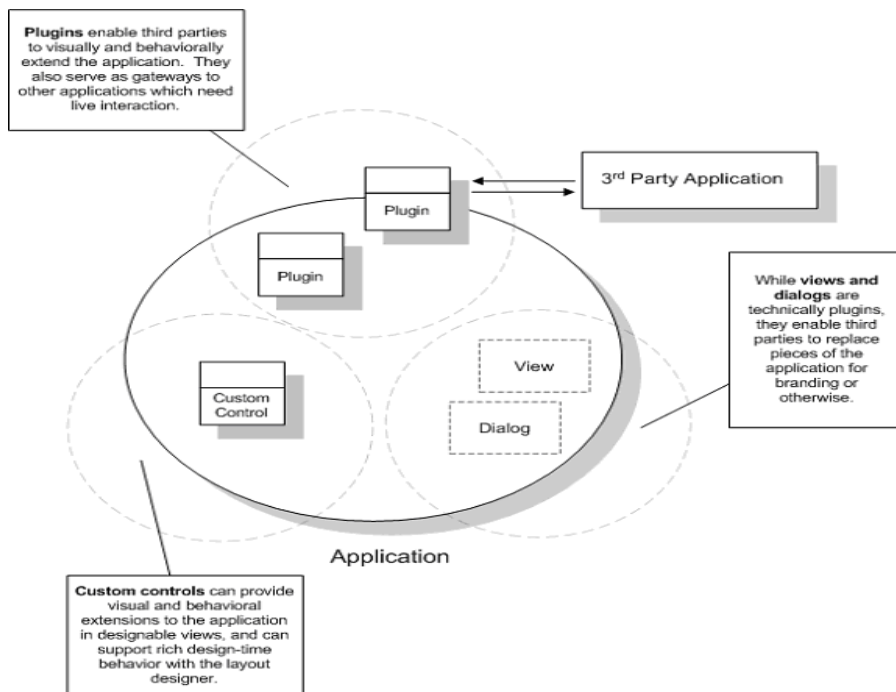
- **Framework Extensibility Model**

The ACT! framework is the core engine of ACT! functionality and data access. It is the business layer and contains no user interface. It can be extended in the following manner:



- **Application Extensibility Model**

The ACT! application is the user interface portal to the framework, and provides the rich Graphical User Interface (GUI) functionality and features present in the ACT! application. The application has many extensibility edges:



## Act! Security Model

The ACT! Software Development Kit (SDK) provides a security model to protect data and to allow users access to specific features and functionalities.

### Security Model

The Database security is defined and stored at the database level in a set of tables. This includes user information as well as record-level access. Security is enforced for access from third-party developers with the SDK serving as the "gatekeeper."

Functional security, as defined by user roles, sets privileges that control a user's access to functions (such as creating, editing, or deleting records). User roles are predefined and cannot be changed. At the record level, record managers can specify contact records as public or private.

### Users Roles

User roles determine the areas and features of ACT! that a user can access or utilize.

ACT! includes five roles: **Administrator**, **Manager**, **Standard**, **Restricted**, and **Browse**. A user can only be assigned one role. Each role is given permission to access one or more areas of the database.

The areas, such as creating or deleting a contact, are combined in feature sets. Feature sets are pre-defined and specify which feature can be accessed by each role. For example, the standard user role is given permission to create, edit, and delete a contact record as defined in the "Contacts" feature set. The restricted user role, however, may only be granted permission to create and edit contact records. For detailed information about each role, see the ACT! online Help or the ACT! User's Guide.

Role rules:

- You cannot create or remove roles.
- You cannot modify roles. For example, you cannot add or remove the functionality assigned to a role.
- Security cannot be modified at the Standard or lower roles. Only an Administrator or Manager role can modify user security.

### Getting Started

- Software Requirement  
To develop application & interfaces using the ACT! SDK, you must use:
  - Microsoft Framework 2.0 or higher
  - Visual Studio.Net 2005 or higher

The ACT! SDK does not support other compiling languages. For a list of requirements to use Visual Studio.NET, see Microsoft's System Requirements for Visual Studio.NET.

### **We have used ACT! By Sage Custom Control in different scenario; here is Project scope (requirement) that we achieved.**

**Requirement Scope:** To incorporate custom control functions within ACT 2008 framework (10.0v) for managing prospect/investor relationships with fund Managers. We have to create a (ACT) "Company" based system that is enabling the tracking of both Managers and Institutional investors from a corporate entity level. The information is filtering down to the associated ACT contacts of these Company entities. A key feature to be developed the advent of the "Demo" records. On the Database level, "Demo" is storing in their own table. This allows fund records to be flexible associated to both the Money Managers and the investors/clients who are associated with them. There are also some dynamic reports views that can be easily exported to Excel sheet.

### **Our solution**

We have used following architecture to achieve the our aspiration at design level

### Architecture – Technology

- VB.Net 2005, ACT Database
- COM + Interoperability
- 3-tier architecture
- Operating system windows XP and latter version

### Third party tools used in the application:

- KryptonToolkit 2.8.5
- DX Toolkit 8.2.2

### The Master views (at design level)

Here, we have added control to display Master view of user custom control that already integrated with ACT! By Sage Application in navigation bar:

Other part where we integrated custom Tab views, which appears on the lower tab of contact/company screen and they are dynamically filtered by current contact/company records as the user navigates from one record to next in ACT record.

### Investor View:

There is one screen with 4 custom Tabs: Forecasting, Marketing, Inv. Summary, and Transaction History. All are associated with "Demo", "Investor" & "Manager".

Here we are going to explain, how we achieved the target with ACT! Custom controls.

**Description:** We have embedded our control within ACT, Tab view with custom control, grid with filter options & context menu with right click features.

### Some Code samples:

#### Following references are used:

```
Imports Act.Framework
Imports Act.Framework.CustomEntities
Imports Act.Shared.Collections
```

#### To create a custom company record (DemoManager)

```
Private DemoManager As CustomSubEntityManager(Of Demo)
Public fieldDemoID As CustomEntityFieldDescriptor
Public fieldDemoName As CustomEntityFieldDescriptor

Public Property Guid() As Integer
Get
    Return DataManager.DemoManager.fieldDemoID.GetValue(Me)
End Get
Set(ByVal value As Integer)
    DataManager.DemoManager.fieldDemoID.SetValue(Me, value)
End Set
End Property
```

#### To get value of Demo Manager

---

```
Private Sub GetDemoManager(ByVal actFramework As ActFramework)
    demoManager =
    actFramework.CustomEntities.GetSubEntityManager(Of Demo) _
    (actFramework.CustomEntities.GetCustomEntityDescriptor("Demo"))

    fieldDemoID = demoManager.GetCustomEntityFieldDescriptor("DemoGuid",
    MutableEntities.FieldNameType.Alias)
```



```
fieldDemoName = demoManager.GetCustomEntityFieldDescriptor("DemoName",  
MutableEntities.FieldNameType.Alias)
```

```
End Sub
```

---

**To Get New GUID**

---

```
Public Function GetNewDemoId() As Integer  
    Dim demoIdSort As New SortCriteria(fieldDemoID,  
System.ComponentModel.ListSortDirection.Descending)  
    Dim demoList As CustomEntityList(Of Demo) =  
demoManager.GetCustomEntities(New SortCriteria() {demoIdSort})  
  
    If demoList.Count > 0 Then  
        Return CType(fieldDemoID.GetValue(demoList(0)), Integer) + 1  
    Else  
        Return 1  
    End If  
End Function
```

---

**To save Demo record**

---

```
Private Sub cmdSave_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles cmdSave.Click  
  
    'Some brief samle codings  
Try  
    Me.Cursor = Cursors.WaitCursor  
        DataManager.demoManager.fieldDemoName.SetValue(Me.demo,  
txtDemoName.Text)  
    Me.demo.Update()  
    Me.Cursor = Cursors.Default  
Catch ex As Exception  
    Me.Cursor = Cursors.Default  
    ErrorManager.ShowDialog(ex, actApp.ActFramework)  
End Try  
End Sub
```

---

**To delete selected records in the Grid**

---

```
Private Sub cmdDelete_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles cmdDelete.Click  
    Try  
        If grdDemos.Rows.Count = 0 Then Exit Sub  
        If grdDemos.CurrentRow.DataBoundItem Is Nothing Then Exit Sub  
        If MessageBox.Show("Are you sure?", "Delete!",  
MessageBoxButtons.YesNo, _  
MessageBoxIcon.Question, MessageBoxDefaultButton.Button2) =  
DialogResult.No Then Exit Sub  
        Dim dataSetRow As dtsDemoList.DemoListRow = _  
CType(grdDemos.CurrentRow.DataBoundItem, DataRowView).Row  
        For Each dmo As Demo In demoList  
            If dmo.Guid = dataSetRow.DemoId Then  
                demoList.Remove(dmo)  
                dataSetRow.Delete()  
                grdDemos.ClearSelection()  
            End If  
        Next  
    End Try
```



METAOPTION LLC, 574 NEWARK AVANUE, SUITE 210  
JERSEY CITY, NJ 07306; PH: 201-377-3150  
EMAIL: [info@metaoption.com](mailto:info@metaoption.com); WEB SITE: [www.metaoption.com](http://www.metaoption.com)

---

```
        grdDemos.Refresh()  
        Exit For  
    End If  
Next  
    FilterGrid()  
Catch ex As Exception  
    ErrorManager.ShowDialog(ex, actApplication.ActFramework)  
End Try  
End Sub
```

---

**To Bind data into Grid**

```
Private Sub FilterGrid()  
  
    Dim criteria As New List(Of IFilterCriteria)  
    Dim demoManagerCriteria As InFilterCriteria  
    Dim demoTypeCriteria As InFilterCriteria  
  
    If cmbDemoManager.Text.Length > 0 Then  
        demoManagerCriteria = New  
InFilterCriteria(DataManager.demoManager.fieldDemoName, _  
New String()  
{cmbDemoManager.Text})  
        criteria.Add(demoManagerCriteria)  
  
    End If  
    If cmbStatus.Text.Trim.Length > 0 Then  
        Dim statusCriteria As New  
InFilterCriteria(DataManager.demoManager.fieldStatus, _  
New String() {cmbStatus.Text})  
        criteria.Add(statusCriteria)  
    End If  
    If cmbDemoType.Text.Trim.Length > 0 Then  
        demoTypeCriteria = New  
InFilterCriteria(DataManager.demoManager.fieldDemoType, _  
New String() {cmbDemoType.Text})  
        criteria.Add(demoTypeCriteria)  
    End If  
    If Me.dtpStartDate.SelectionLength > 0 Then  
        Dim dateCriteria As New  
DateFilterCriteria(DataManager.demoManager.fieldStartDate, _  
dtpStartDate.SelectedText)  
        criteria.Add(dateCriteria)  
    End If  
  
    demoList = DataManager.demoManager.Manager.GetCustomEntities(Nothing,  
criteria.ToArray)  
  
    With demoList  
        .FieldDescriptors.Add(DataManager.demoManager.fieldDemoID)  
        .FieldDescriptors.Add(DataManager.demoManager.fieldDemoName)  
        .FieldDescriptors.Add(DataManager.demoManager.fieldDemoManagerID)  
        .FieldDescriptors.Add(DataManager.demoManager.fieldDemoType)  
        .FieldDescriptors.Add(DataManager.demoManager.fieldStartDate)  
        .FieldDescriptors.Add(DataManager.demoManager.fieldStatus)
```

```
End With
grdDemos.DataSource = demoList
Dim gridLayout As New Preferences.GridLayout.GridLayout(grdDemos,
actApplication.ActFramework)
End Sub
```

#### To export grid records into Excel

---

```
Private Sub btnExportExcel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnExportExcel.Click
Try
If grdDemos.Rows.Count > 0 Then
Dim excelReport As New Helpers.ExcelExport
excelReport.OpenObjects()
excelReport.FromGrid(grdDemos)
End If
Catch ex As Exception
ErrorManager.ShowDialog(ex, actApplication.ActFramework)
End Try
End Sub
```

Note: Helpers class contains a bit of Excel export code!

```
Public Sub OpenObjects()
System.Threading.Thread.CurrentThread.CurrentCulture =
System.Globalization.CultureInfo.CreateSpecificCulture("en-US")
newfrmProgress.lblStatus.Text = "Creating Excel Document..."
newfrmProgress.Show()
Application.DoEvents()
excelApp = CreateObject("Excel.Application")
excelApp.Visible = False
excelWorkbook = excelApp.Workbooks.Add
excelSheet = excelWorkbook.Worksheets.Add
End Sub
```

Demo screen contains couple of different icon and each icon is having unique functionalities. Some code of the functionalities like save, delete, Excel export etc are mentioned above.

#### How to achieve to create custom tables with code in ACT! Database?

```
Private Shared Function CreateNewEntities(ByVal mainForm As MainForm, _
ByVal updateStatus As
MainForm.UpdateStatusEventHandler) As Boolean
FW.Database.LockDatabase(Act.Framework.DatabaseLockReason.SchemaChanges, "",
True, True)
mainForm.Invoke(updateStatus, New Object() {"Creating Demos Table...", 50})
If Not CreateDemosTable() Then
FW.Database.UnlockDatabase()
Return False
End If
Return True
End Function
Private Shared Function CreateDemosTable() As Boolean
Try
Dim demoTable As CustomEntityDescriptor = _
```



```
clsACT.FW.CustomEntities.GetCustomEntityDescriptor("Demo")
Dim OlddemoTable As CustomEntityDescriptor = _

clsACT.FW.CustomEntities.GetCustomEntityDescriptor("tblDemos")

    If Not demoTable Is Nothing Then
clsACT.FW.CustomEntities.DeleteCustomEntity(demoTable)
    If Not OlddemoTable Is Nothing Then
clsACT.FW.CustomEntities.DeleteCustomEntity(OlddemoTable)
        demoTable =
clsACT.FW.CustomEntities.CreateCustomSubEntity("Demo", "Demo",
Act.Framework.CustomEntities.ParentEntity.Contacts,
Act.Framework.CustomEntities.ParentEntity.Companies, False)
        'Primary fields
        Dim fd As New Act.Framework.Database.FieldDescriptor("DemoGuid",
"DemoGuid", demoTable, Act.Framework.Database.FieldDataType.Number)
        fd.AllowEmpty = False
        fd.IsPrimary = True
        clsACT.FW.Fields.Save(fd)
        'Attribute fields
        CreateACTField("DemoName", demoTable, FieldDataType.Character,
FW)
        FieldDataType.Character, FW)
    Catch ex As Exception
        WriteErrorLog(ex.ToString)
    End Try
    Return True
End Function
```

By Anwarul Haque, MetaOption LLC



METAOPTION LLC, 574 NEWARK AVANUE, SUITE 210  
JERSEY CITY, NJ 07306; PH: 201-377-3150  
EMAIL: [info@metaoption.com](mailto:info@metaoption.com); WEB SITE: [www.metaoption.com](http://www.metaoption.com)

---