

ACT!® Developer's Reference

ACT![®] Developer's Reference

The software described in this book is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Copyright Notice

Copyright © 2005 Best Software SB, Inc.

Released: 1/2005 for ACT! (v7) for Windows and ACT! (v7) Premium/Professional for Workgroups.

This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent in writing from Best Software SB, Inc., 1505 Pavilion Place, Norcross, GA 30093 ATTN: Legal Department.

ALL EXAMPLES WITH NAMES, COMPANY NAMES, OR COMPANIES THAT APPEAR IN THIS MANUAL ARE FICTIONAL AND DO NOT REFER TO, OR PORTRAY, IN NAME OR SUBSTANCE, ANY ACTUAL NAMES, COMPANIES, ENTITIES, OR INSTITUTIONS. ANY RESEMBLANCE TO ANY REAL PERSON, COMPANY, ENTITY, OR INSTITUTION IS PURELY COINCIDENTAL.

Every effort has been made to ensure the accuracy of this manual. However, Best Software makes no warranties with respect to this documentation and disclaims any implied warranties of merchantability and fitness for a particular purpose. Best Software shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual or the examples herein. The information in this document is subject to change without notice.

Trademarks

ACT! is a registered trademark of Best Software SB, Inc. Best Software Insights For The Life Of Your Business is a trademark of Best Software, Inc. Palm OS is a registered trademark, and Palm is a trademark of PalmSource, Inc. Microsoft, Outlook and Windows are registered trademarks of Microsoft Corporation. All other trademarks are the property of their respective owners.

End User License Agreement

ACT! (v7) for Windows and ACT! (v7) Premium/Professional for Workgroups are protected by an End User License Agreement. To view the agreement, go to the Help menu in the product, click About ACT!, and then click the End User Agreement button. The ACT! Software Development Kit is protected by the End User License Agreement provided in this Developer's Reference and in the SDK online help.

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

Best Software SB, Inc. End User License Agreement for the ACT! 2005 (7.x) Developer's Reference for the Software Development Kit

NOTICE: BEST SOFTWARE SB, INC. ("BEST") LICENSES THIS DEVELOPERS' REFERENCE FOR THE SOFTWARE DEVELOPMENT KIT TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS END USER LICENSE AGREEMENT. PLEASE READ THE TERMS CAREFULLY BY PRINTING AND/OR USING THIS DEVELOPERS' REFERENCE, YOU INDICATE YOUR ASSENT TO THEM; IF YOU DO NOT ACCEPT THE TERMS OF THIS AGREEMENT, YOU ARE PROHIBITED FROM PRINTING AND/OR USING THIS DEVELOPERS' REFERENCE AND YOU WILL NOT HAVE A LICENSE FOR THE SDK.

The Developers' Reference for the ACT! 2005 (7.x) Software Development Kit (the "SDK") and any printed and electronic manuals, guides, bulletins, and online Help (the "Documentation") that accompany this End User License Agreement (the "Agreement") are the property of Best or its licensors and are protected by copyright law and international treaty. While Best or its licensors continue to own the SDK, you will have certain rights to use the SDK after your acceptance of this Agreement. "Use" means: downloading a copy of the SDK on a hard disk drive within a single computer, executing or displaying the SDK.

Your rights and obligations with respect to the use of this SDK are as follows:

1. GRANT OF LICENSE

Best hereby grants to you a limited, nontransferable, non-exclusive license to use the SDK under the terms stated in this Agreement for use in your business or profession. Best reserves all rights not expressly granted by this Agreement and you hereby acknowledge that all title and ownership of the SDK and all associated intellectual property rights are and shall remain with Best. This Agreement permits you to:

- (a) use the SDK: (i) on a single primary computer; and (ii) on a secondary computer that may be either your home computer or a portable computer that you own or use in your business or profession;
- (b) retrieve, modify, or delete ACT! database data or database structure only by way of the ACT! products, the ACT! SDK, or the ACT! OleDb provider; and
- (c) make one copy of the SDK for archival purposes, or copy the SDK onto the hard disk of your computer and retain the original for archival purposes.

2. LICENSE RESTRICTIONS

This Agreement does not include the right to perform any of the following and you agree to refrain from performing any of the following:

- (a) participate in deceptive, destructive or illegal practices related in any way to use of the SDK or this Agreement;
- (b) copy any Documentation that accompanies the SDK;
- (c) make any copies of all or part of the SDK other than as expressly permitted in this Agreement;
- (d) sublicense, rent, lease, or loan, any portion of the SDK or host the SDK on your computer for others to use;
- (e) re-sell or distribute any portion of the SDK to another person or entity;

- (f) reverse engineer, decompile, disassemble, modify, translate, make any attempt to discover the source code of the SDK;

- (g) use the ACT! trademarks as part of a product name, trademark or business name without prior written approval from Best;

- (h) develop a product that directly competes with ACT! and/or build conversion functionality that converts end user data from ACT! to a competing product or service; market or distribute add-ons or enhancements to ACT! without the prior written consent of Best;

- (i) circumvent technological measures to prevent direct database access, nor manufacture tools or products to that effect; or

- (j) copy any portion of the ACT! product graphical user interface for incorporation into or use for any software or other product without the prior written consent of Best.

3. SUPPORT

Best disclaims any responsibility to provide any customer support for the SDK.

4. TERMINATION

This Agreement may be terminated by Best without notice if you fail to comply with any term or condition of this Agreement. This Agreement may also be terminated for any reason or for no reason at all with 30 days notice. Upon termination, you must immediately destroy all copies of the SDK.

5. NO WARRANTY

The SDK is accepted by you "AS IS" AND "WITH ALL FAULTS." ALL WARRANTIES CONCERNING THE SDK, EXPRESS OR IMPLIED, STATUTORY, OR IN ANY OTHER PROVISION OF THIS AGREEMENT INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE, ARE HEREBY EXPRESSLY DISCLAIMED AND EXCLUDED. YOUR SOLE AND EXCLUSIVE REMEDY FOR A BREACH OF THIS AGREEMENT BY BEST SHALL BE TO TERMINATE THIS AGREEMENT.

6. LIMITATION OF LIABILITY AND DAMAGES

REGARDLESS OF WHETHER ANY PROVISION SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE, IN NO EVENT WILL BEST OR ITS LICENSORS BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT OR SIMILAR DAMAGES, INCLUDING ANY LOST PROFITS OR LOST DATA ARISING OUT OF THE USE OR INABILITY TO USE THE SDK EVEN IF BEST OR ITS LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. THE DISCLAIMERS AND LIMITATIONS SET FORTH ABOVE WILL APPLY REGARDLESS OF WHETHER YOU ACCEPT THE AGREEMENT.

7. EXPORT RESTRICTIONS

You agree to comply to the extent applicable with the United States Export Administration regulations, the International Traffic in Arms regulations and any regulations or licenses administered by the Department of the Treasury's Office of Foreign Assets Control.

8. GENERAL

(a) To the fullest extent permitted by law and consistent with valid entry into a binding agreement, the controlling language of this Agreement is English and any translation you have received has been provided solely for your convenience. In the event you have entered into this Agreement by means of the display of a translated version of this Agreement in a language other than U.S. English, you may request a U.S. English language version of this Agreement by notice to Best. To the fullest extent permitted by law, all correspondence and communication between you and Best under this Agreement must be in English language.

(b) The exclusive judicial forum for any action related to this Agreement shall be an appropriate federal or state court located in Georgia. This Agreement shall be governed by the internal laws of the forum state without regard to the conflict of laws provisions thereof.

(c) This Agreement allocates risk between you and Best as authorized by applicable law, and pricing of Best's products reflects this allocation of risk and the limitation of liability contained in this Agreement. If any provision of this Agreement is found invalid or unenforceable pursuant to judicial decree, the remainder of this Agreement shall be valid and enforceable according to its terms.

(d) ACT! is a registered trademark of Best Software SB, Inc. For an up-to-date list of copyright and trademark statements, refer either to the copyright page of the Software User's Guide for your ACT! software or the Help About window within the ACT! software. Other product names mentioned may be service marks, trademarks, or registered trademarks of their respective companies and are hereby acknowledged.

(e) No failure or delay of either party to exercise any rights or remedies under this Agreement shall operate as a waiver thereof, nor shall any single or partial exercise of the same or other rights or remedies preclude any further or other exercise of the same or other rights or remedies, nor shall any waiver of any rights or remedies with respect to any circumstances be constructed as a waiver thereof with respect to any other circumstances.

(f) Quebec. With regard to Quebec, the parties declare that they have required that this Agreement and all documents related hereto, either present or future, be drawn up in the English language only. Les parties déclarent par les présentes qu'elles exigent que cette entente et tous les documents y afférents, soit pour le présent ou l'avenir, soient rédigés en langue anglaise seulement.

(g) Sections 5 (No Warranty), 6 (Limitation of Liability and Damages), 8(b) (Governing Law), and this Section 8(g) shall survive the expiration or termination of this Agreement.

(h) This Agreement constitutes the entire agreement between you and Best with respect to the subject matter hereof, and supersedes all proposals, oral or written, and all other communications between the parties with respect to such subject matter. This Agreement shall not be modified, except by written agreement signed by the parties hereto. Employees, officers, and agents of Best are not authorized to modify this Agreement, or make any additional representations, commitments, or warranties binding on Best, unless made in writing and signed by an authorized officer of Best.

(i) Best shall not be liable for and shall be excused from any failure to deliver or perform or for delay in delivery or performance due to causes beyond its reasonable control, including but not limited to, work stoppages, shortages, civil disturbances, terrorist actions, transportation problems, interruptions or power or communications, failure or suppliers or subcontractors, natural disasters or other acts of God.

(j) All notices given hereunder shall be in writing and sent by overnight courier or delivered in person to Best Software SB, Inc., Legal Department, 1505 Pavilion Place, Norcross, Georgia 30093.

Contents

	Introduction	1
	Overview of the ACT! Development Platform	1
	Best practices for working with the SDK	2
	Supported ACT! SDK assemblies	2
	Third-party components	2
	Add-on components	2
	ACT! version support	3
	About the Developer's Reference	3
Chapter 1	Extensibility Model	5
	Consuming the Framework	5
	Extending the Application	6
	Plugins	6
	Custom Controls	6
	Custom Tabs	7
Chapter 2	Entities and Relationships	9
	Entities	9
	Contacts	9
	Groups	10
	Companies	10
	Opportunities	11
Chapter 3	The Framework Object Model	13
	The ActFramework class	13
	Getting started	13
	Managers	13
	Metadata	13
	Entity lists	14
	Working with data	15
Chapter 4	The Application Object Model	17
	The ActApplication class	17
	UI Managers	17
	Plugins	17
	Views	17
	Application state	18
	Menus and toolbars	18
	Custom controls	18
	Using .NET design-time attributes and types.	18

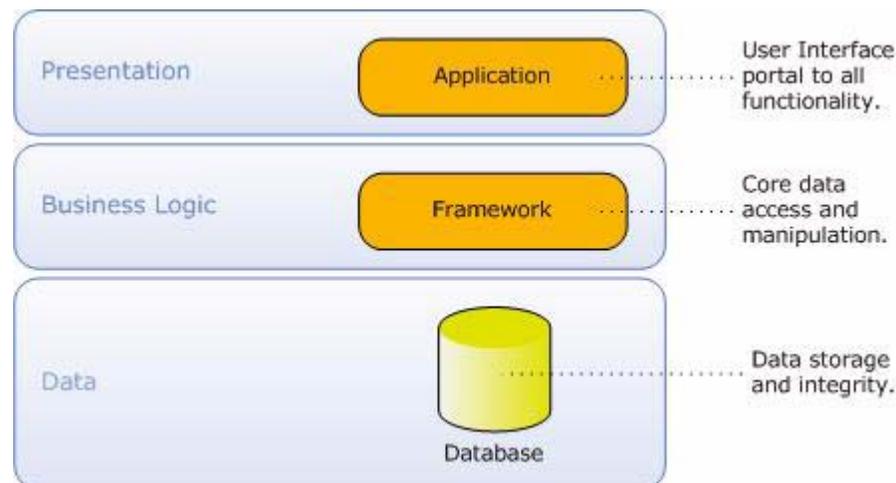
	Using design-time instruments.	18
	Using .NET serialization techniques	19
	Using ACT! design-time attributes, interfaces, and types	19
	Binding custom controls to a field	19
	Custom Tab	20
Chapter 5	Sample Code	21
	Using Framework metadata	21
	Getting a contact list.	22
Index.		23

Introduction

Overview of the ACT! Development Platform

The ACT! platform consist of feature-rich components that are highly extensible. The platform is built on the .NET Framework. As a development platform, ACT! meets ACT!'s historically rich customization, personalization, and integration goals. Where ACT! seeks to empower the end users to customize the product to their business, the ACT! SDK helps third parties extend that vision through independent development.

The ACT! platform has three logical tiers, the Application, the Framework, and the Database, as shown in the following figure.



The Application encompasses all user interface aspects of ACT!, including user interface screens known as views, navigational components such as menus and toolbars, and design-time components.

The Framework is the engine of ACT!, providing core functionality including data access, schema metadata and modifications, security, synchronization, database creation and maintenance, data exchange and interoperability, and other essential elements. The Framework includes access to first-class entities such as contacts, groups, companies, and opportunities. It also provides access to extended data including notes, histories, activities, and documents.

The Database is the storage container for ACT! primary data. It maintains data integrity and relationships.

Best practices for working with the SDK

This section identifies the supported ACT! assemblies and gives recommendations for working with third-party components, placing DLL's, and ensuring version support.

Supported ACT! SDK assemblies

Developers should build applications only against the supported ACT! SDK assemblies, as follows.

- Framework assemblies:
 - Act.Devices.Entities
 - Act.Devices.Synchronization
 - Act.Framework
 - Act.Framework.ComponentModel.Core
 - Act.Framework.DataExchange
 - Act.Framework.DataExchange.OutlookSync
 - Act.Framework.DataExchange.PalmReader
 - Act.Framework.StandaloneActivityRecurUtility
 - Act.Framework.Synchronization
 - Act.Shared.Collections
 - Act.Shared.Diagnostics
 - Act.Shared.ComponentModel
- UI assemblies:
 - Act.UI
 - Act.UI.Core

Other ACT! assemblies exist, but are not supported or maintained for compatibility. Future versions of ACT! may include non-compatible versions of these assemblies.

Developers should not depend on any DLL's or EXE's that ACT! installs other than the supported assemblies listed above. Unsupported DLL's and EXE's may change or be removed.

Third-party components

Developers should not build products against third-party components deployed by ACT!. Third-party components should be considered unsupported and may change or be removed over inline releases. Using third-party components is not only unsupported but likely illegal. A developer who uses third-party components must purchase a copy and install it as part of the application's install.

Add-on components

Developers should not place DLL's that are dependant on an Add-on in the Plugins folder. If you build an Add-on that has other dependent DLL's, including third-party DLL's, either install these in the GAC or create a product subfolder beneath the Plugins folder and place your dependencies there (you will need to load those dynamically). This ensures that neither ACT! nor other Add-on .NET dependencies interfere with yours and ensures prompt loading of plugins.

If you develop an Add-on, limit CPU time in IPlugin interface methods and in event handlers of the ACT! Framework or Application. Extensive CPU time may degrade ACT! general performance. Instead, use asynchronous patterns to defer work to another thread.

Add-ons should use proper exception handling, especially in handlers for ACT! events. Because Add-ons are hosted by ACT!, exceptions created by Add-ons affect normal ACT! behavior and stability.

Use your company name in all your Add-on assembly and interop wrapper file names and in folder names in the Plugin folder. This will prevent conflict with assembly and file names of other Add-on vendors.

ACT! version support

To have your product support any ACT! 7.x version, build your product against 7.0. Do not release products built against any prerelease versions of ACT!.

About the Developer's Reference

This Developer's Reference provides information on using the ACT! SDK to extend the application by developing plugins, custom controls, and custom tabs. The Developer's Reference also provides an overview of entities, relationships, the Framework Object Model, and the Application Object Model. [Chapter 5, "Sample Code"](#) provides sample code for using Framework metadata.

Chapter 1

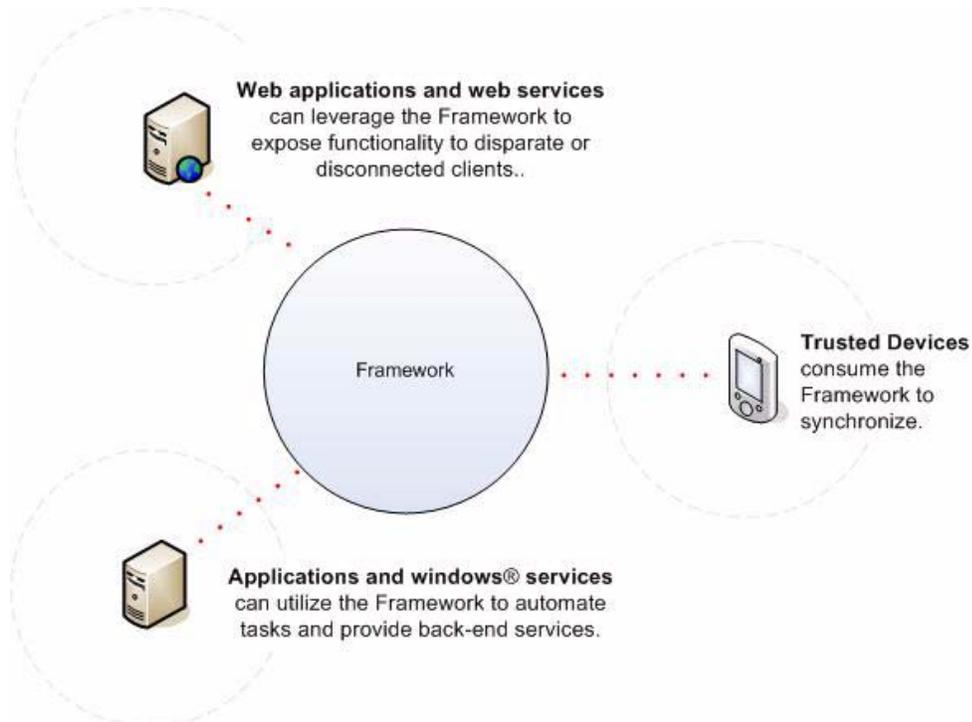
Extensibility Model

Both the Framework and Application tiers have special ways for third parties to access data and customize, integrate with, automate, and extend ACT!. The unique needs and interactions of third parties will determine which ACT! integration path they should use.

This chapter explains when third parties will need to use the Framework and explains some ways to extend the application using plugins, custom controls, and custom tabs.

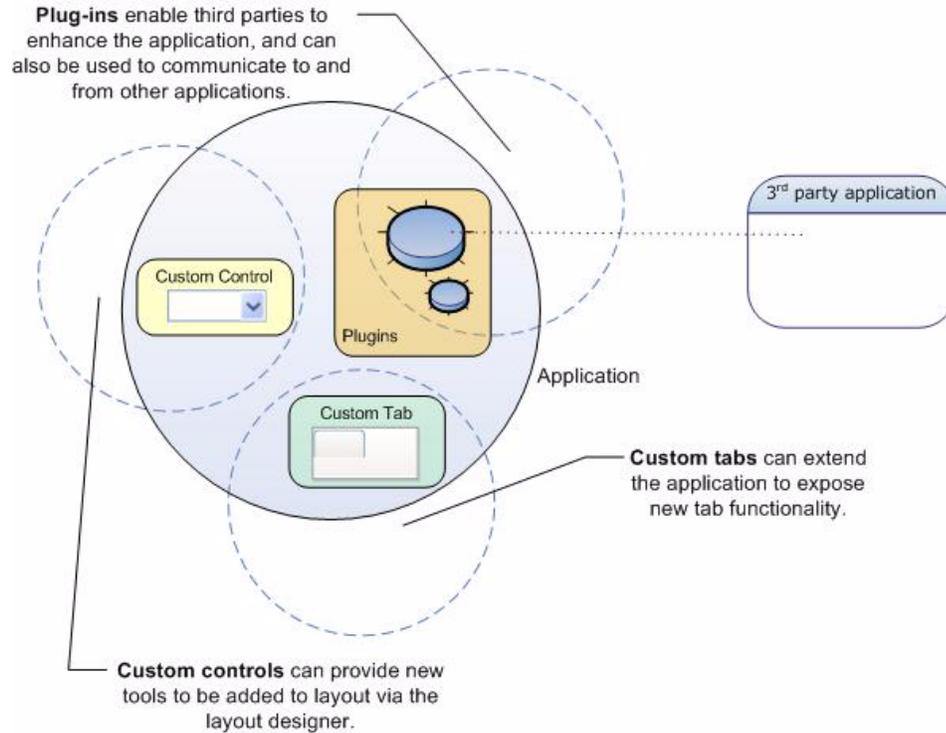
Consuming the Framework

The Framework can be consumed when third parties need to integrate with ACT! and when no interaction with the Application or User Interface is needed. Applications and Windows® services can consume the Framework to access data, automate functionality, and provide back-end services. Web applications and services can consume the Framework to provide client applications or back-end solutions across network boundaries. Trusted devices can also synchronize using the Framework.



Extending the Application

The Application has several extensibility points, including plugins, custom controls, and custom tabs. Third parties can use these separately to provide new functionality or together to create more complex solutions.



Plugins

Plugins enable third parties to extend the application behaviorally and/or visually. Plugins can also serve as gateways to other applications or services that need live interaction with the Application. As in other applications, plugins in ACT! are given context in the hosting application when they are loaded. Plugins can access all of the Application (and Framework). Typically, plugins will subscribe and react to events in the Application and Framework to perform some specialized functionality.

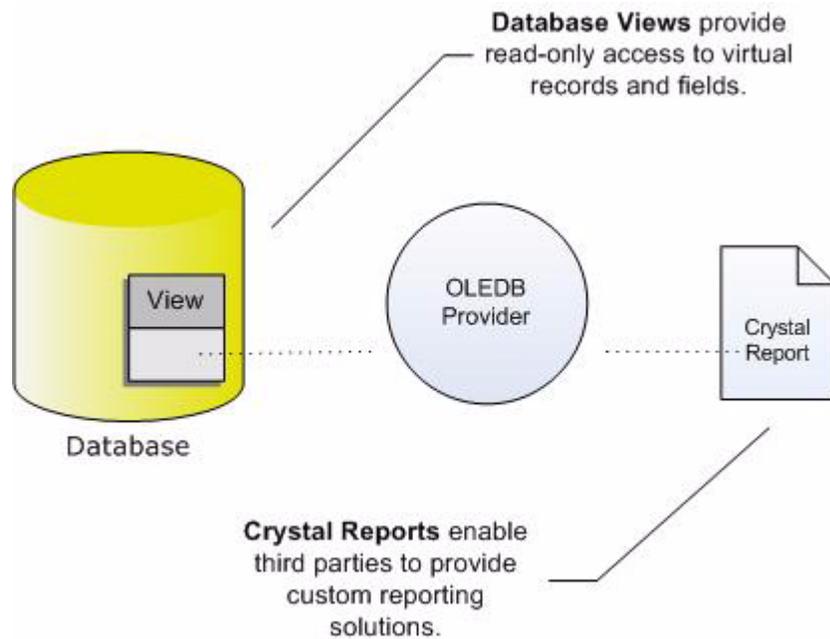
Custom Controls

Third parties can use custom controls to extend the Application's designable views. These include the Contact, Group, and Company detail views. Custom controls also can support rich design-time behavior and integrate with the Layout Designer.

Custom Tabs

Third parties can add custom tabs to provide new ways to view data, for example, in detail views of the application.

ACT! includes an OLEDB Provider, which enables read-only access to Database views. This is the lowest form of data access, since it circumvents the Framework. It can be used to generate custom reports with tools such as Crystal Reports. Security is maintained using the OLEDB provider.



Chapter 2

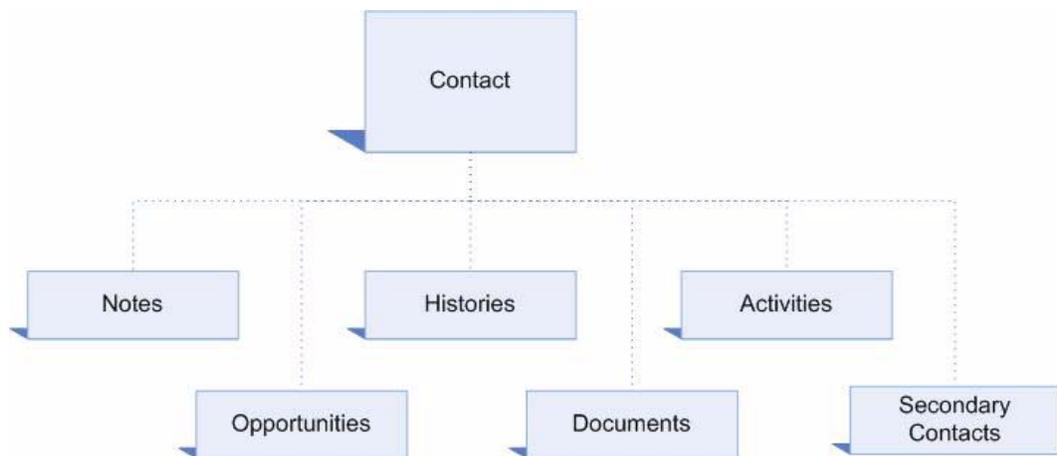
Entities and Relationships

Entities

ACT! consists of primary data or entities and extended data or entities. Primary data includes contacts, groups, companies, and opportunities. Extended data includes notes, histories, activities, secondary contacts, and documents. This chapter provides a brief overview of each of these entities.

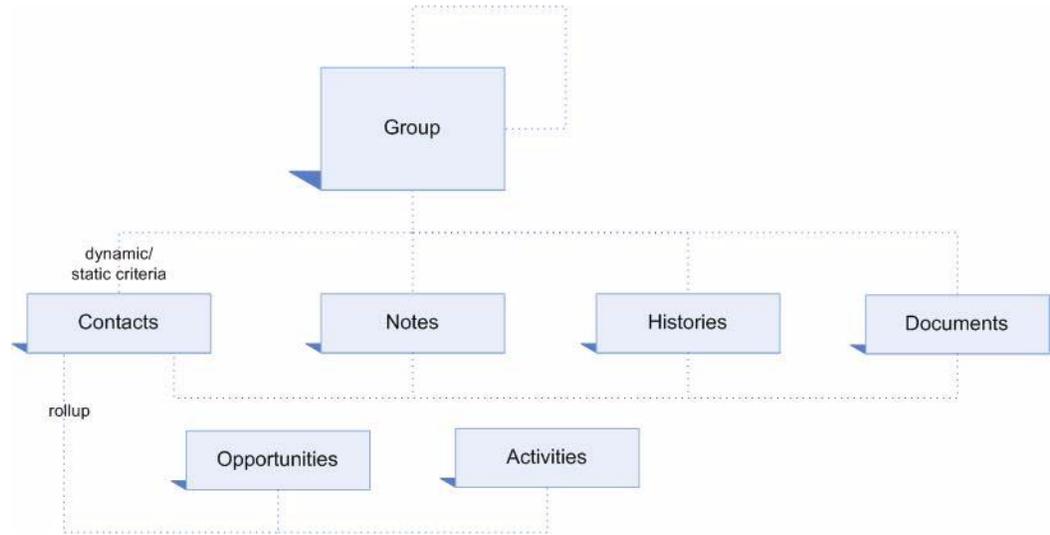
Contacts

Contacts are a first-class entity. Any kind of extended data can be associated with a contact, including notes, histories, activities, opportunities, documents, and secondary contacts.



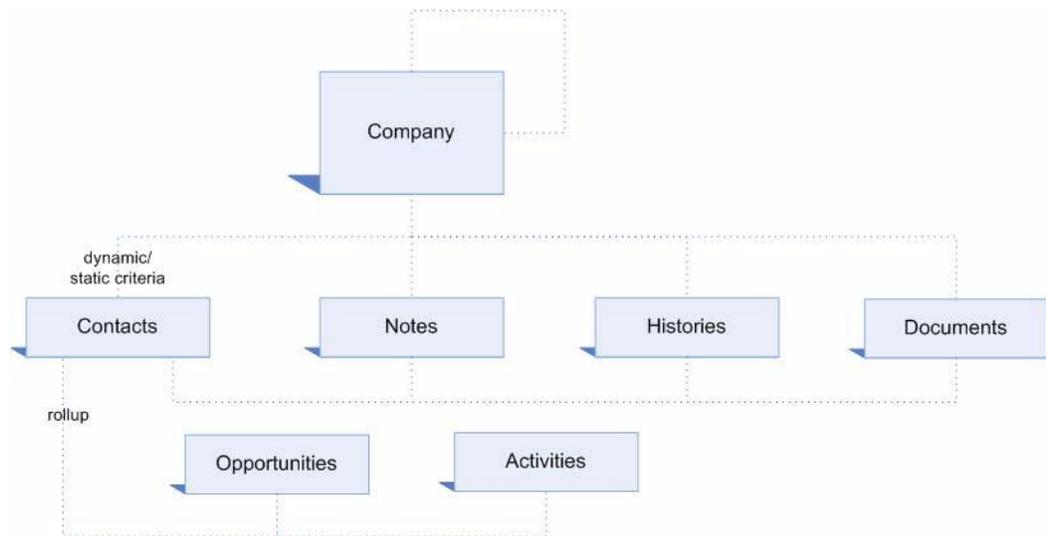
Groups

Groups are dynamic or static sets of contacts. Groups can have their own notes, histories, and documents. Opportunities and activities can be rolled up, so that users can access any of the items associated with contacts in a group. Groups can also contain subgroups.



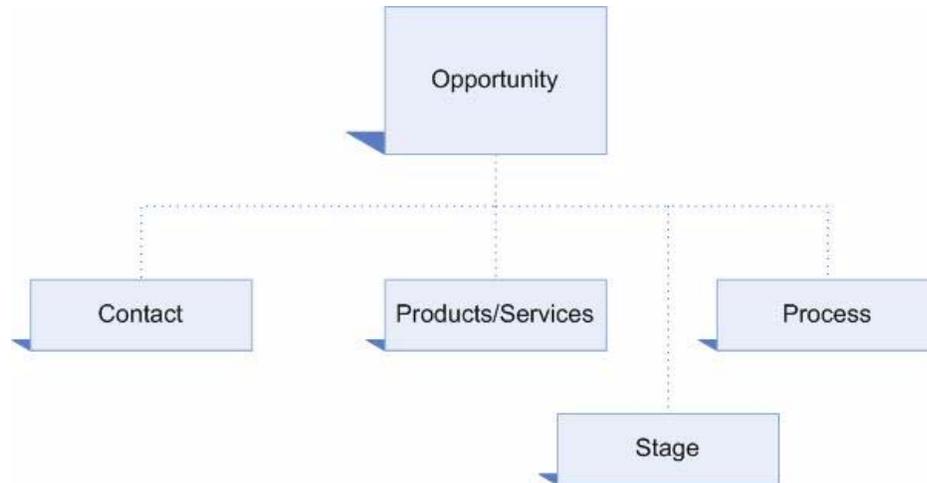
Companies

Companies are similar to groups. They can have their own notes, histories, and documents, and they can roll up opportunities and activities as well. Companies can also have sub-companies, known as divisions.



Opportunities

Opportunities are associated with one contact, one process, and one stage. Opportunities can have many products and services.



Chapter 3

The Framework Object Model

This chapter gives an overview of the Framework object model.

The ActFramework class

ActFramework, found in Act.Framework.dll, is the root Framework class. It is the entry point to all ACT! core functionality.

Getting started

To use ActFramework, you must be authenticated as an ACT! user and log in as described in the following:

```
ActFramework framework = new ActFramework();  
framework.LogOn("CHuffman", "password", "localhost", "MyDatabase");
```

Managers

ActFramework exposes Managers via properties. Managers are gateways to feature- or entity-related functionality. For example, a ContactManager is accessible for the Contacts property, which is responsible for contact-related operations.

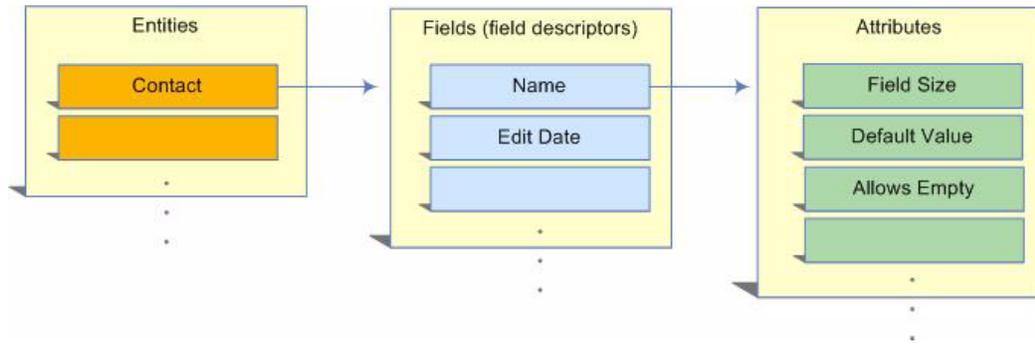
Metadata

Much of the structure that defines extended data classes, such as Note, History, and Activity, is well defined and unalterable. The data surrounding these is available via members of the data's respective classes. However, you can change the topology of primary entities, such as Contact, Group, Company, and Opportunities. You do this via Define Fields (using the application) or via the DatabaseManager (using the Framework). These entities are largely metadata driven. Determining their landscape and data is a process of discovery.

The Framework exposes such an entity's fields via field descriptors. Specifically, entities have specialized types deriving from DBFieldDescriptor. Most of the functionality is based on a .NET native type, the PropertyDescriptor. A PropertyDescriptor is a virtual representation of a property. PropertyDescriptor objects can depict the topology of some classes, which may or may not differ from their real properties. Typically, PropertyDescriptor objects are retrieved via reflection, to dynamically discover the properties of some class. This virtualization can also be used when databinding a list to a grid (by implementing IList). PropertyDescriptor objects also contain an AttributeCollection, which represents any attributes or declarative metadata of the Property.

The Framework extends the capability of a PropertyDescriptor to represent the virtualization of a record's field; more specifically, to dynamically discover an entity's fields. Likewise, the Framework leverages Attribute objects on the PropertyDescriptor (exposed via the AttributeCollection on the Attributes property) to provide attributes of that field (such as size,

mask, or default value). These attributes typically depend on the ACTType of the field (e.g., mask is not applicable to picture fields).



A `PropertyDescriptor` defines a property or field in many ways. It defines the type of the property via `PropertyType` and whether it is read-only via `IsReadOnly`. It assigns the field name via `DisplayName`. `DBFieldDescriptor` extends this to provide `ACTType`, which represents the ACT! type (such as uppercase or phone number). Values can be retrieved from and updated to entities using the `GetValue` and `SetValue` methods, respectively.

Unlike typical retrieval of `PropertyDescriptor` objects via reflection, the Framework enables retrieving `DBFieldDescriptor` objects via an entity's Manager. For consistency and databinding, the ActFramework entity Managers expose all fields as if they were metadata-driven. Thus, all entity managers implement an interface `ISupportMetaData`. This interface allows retrieval of metadata for an entity and filtering of the type of metadata by `Type` or `ACTType`.

See [“Using Framework metadata” on page 21](#) for a sample of how field descriptors can be used.

You are not limited to field descriptors to get or set data in entities. You can also fetch and update field data using the `FieldCollection` indexer on mutable entities. However, using field descriptors is the preferred method of retrieving and setting data. Field descriptors cache needed metadata, so reusing a field descriptor to access or change data on multiple entities (e.g., looping over entities and getting a field value) performs far better than using the field collection on the entity, which has to look up the metadata each time.

Entity lists

All entities can be retrieved via lists (collections). Like metadata, lists are retrieved through their respective entity Manager. Each entity Manager may have specialized parameters, such as filter criteria, that are used to retrieve a list. However, all must at least support retrieving a list passing `SortCriteria`. `SortCriteria` consists of the `PropertyDescriptor` (see [“Metadata”](#) in this chapter), specifying the field, and the `ListSortDirection`, specifying the direction on which to sort (ascending or descending). Some lists, such as mutable entity lists, support sorting on multiple fields.

Entity lists can be bound to any .NET-aware grid controls, which will use the lists `ITypedList` implementation to get the `PropertyDescriptor` objects (see [“Metadata”](#) in this chapter). The `PropertyDescriptor` objects are used to name the columns and get data for each row. Grid controls also will use the entity list's `IBindingList` implementation to sort, search, and react to list notifications, such as when a new item is added.

See [“Getting a contact list” on page 22](#) for a sample.

All entity lists, except for `ActivityList`, fetch data on demand and cache the data as a way to scale to large quantities. As a side effect, data may become stale. To refresh data, invoke the `Refresh` method. Also, be mindful that iterating over or accessing items in the list may cause data to be fetched from the database.

Working with data

You can create and delete entities from their Managers. You can create and delete primary entities, including contacts, groups, and companies, via their lists, using `IBindingList.AddNew()`, and `IList.Remove()` methods. You can retrieve extended data for a primary entity via the Manager of the extended data. For example, the `NoteManager` has `GetNotes` overloads to pass in a contact or a company.

Within a list, you can find items using the `Find()` method, which takes a field descriptor (see [“Metadata”](#) in this chapter) and a value. You can use a lookup to get a list of primary entities that matches a particular criteria. You perform lookups via the `LookupManager`. Lookups are essentially list criteria; each criteria is made up of a logical operator (AND/OR), a field, an operator valid for that field, and a value.

Chapter 4

The Application Object Model

This chapter provides an overview of the Application object model.

The ActApplication class

ActApplication, found in Act.UI.dll, is the parent application class and is the entry point to all User Interface functionality. The Application typically is not created and accessible directly, unlike the Framework, but third parties can acquire its context via extensibility points, such as plugins, custom controls, and custom tabs.

UI Managers

ActApplication exposes UI Managers via properties. Like ActFramework's Managers, the ActApplication's Managers are gateways to feature- or entity-related functionality.

Plugins

Plugins, as previously mentioned, are given context by the Application (and Framework via the Application). The Application serves as a loader and host for plugins, which can then react to events, customize or extend the application, and communicate with other applications.

To become an ACT! plugin, a type must:

- Implement the IPlugin interface (defined in Act.UI.dll).
- Reside in an assembly in the Plugins directory under the ACT! application directory.

The IPlugin interface is a simple interface. It provides an entry point for the application to hand itself to the plugin and a way to notify the plugin when the application is unloaded. After the plugin is loaded, and the ActApplication is provided via the OnLoad method, the plugin is responsible for reacting to events appropriately. For example, the plugin would have to react in a manner appropriate to the context of the ActFramework's AfterLogon and BeforeLogoff events. The plugin would also have to react in a manner appropriate to the context of the BeforeDatabaseLock and AfterDatabaseLock events.

Views

Views are the main User Interface panes in the application. Views, other than calendaring, typically provide detail or lists. Views can be enumerated via the ViewManager. The ActApplication notifies listeners, via the CurrentViewChanged event, when a different view is shown. You can retrieve the current view via the CurrentView property of the ActApplication. Views can be changed and shown via entity UI managers. For example, UIContactManager.ShowDetailView() changes the view.

Application state

The Application contains information about the state of current entities and lists (such as the current contact and current contact list). You can access this information via the `ApplicationState` property in `ActApplication`. Events on the application also exist in the form `XXXChanging` and `XXXChanged`, which notify consumers of entity and entity list changes.

Menus and toolbars

`Act.UI.Core.dll` contains all types and functionality related to menus and toolbars. The `Explorer` property on the `ActApplication` object is the main entry point for accessing, adding, and removing toolbars. Plugins can use this to add a new menu item and to perform an operation when the item is selected.

Custom controls

Custom controls can provide new ways to visualize data and interact with the application in designable views. You can make custom controls available as part of a toolbox tool in the Layout Designer and add them to layouts to enhance the contacts, groups, and companies detail views.

To become a custom control, a type must:

- Implement `IComponent` (for example, by deriving from `Component` or `Control`).
- Be marked with the `CustomControlAttribute` attribute (defined in `Act.Shared.ComponenModel.dll`).
- Reside in an assembly in the `Tools` directory under the ACT! application directory.

This enables end users to access the control by right-clicking on the Layout Designer toolbox and selecting the `Customize` menu. If selected, the custom control is available in the “Custom” category of the toolbox, for use in the Layout Designer.

Once the custom control is placed on a layout, and the layout is saved, that custom control must be installed on the client machine in order for the end user to use that layout.

Using .NET design-time attributes and types

The Layout Designer supports standard .NET design-time related attributes and types; most of this exceeds the scope of this document. However, minimal design-time related functionality is documented here to explain basic design-time interaction with the Layout Designer:

CategoryAttribute Mark a property with this attribute to control the category that will display in the Properties window in the Layout Designer.

DescriptionAttribute Mark a property with this attribute to control the description of the property that will display in the Properties window in the Layout Designer.

Using design-time instruments

The Layout Designer supports design-time instruments such as `TypeConverter` and `UITypeEditor`. You can use these objects to control the behavior of the properties in the Properties window in the Layout Designer.

Using .NET serialization techniques

The Layout Designer also leverages standard .NET techniques for serialization. Custom controls can leverage these to control which properties get serialized in layouts and how this is done:

DefaultValueAttribute Mark a property with this attribute to skip serialization of a property whose value has not changed from the default.

DesignerSerializationVisibility Use this attribute to skip serialization of a property or to serialize the contents of a property (such as a collection).

ShouldSerializeXXX method Use this method, where XXX is the name of a property, to enable a control to programmatically manage whether or not a property should be serialized.

Using ACT! design-time attributes, interfaces, and types

You can use ACT!-related design-time attributes, interfaces, and types in the creation of custom controls:

LayoutToolboxFriendlyNameAttribute Mark a type with this attribute to enable controls to have a friendly name in the designer (other than its type). Provide a public static string `LayoutFriendlyName` property to return the name.

TableAttribute Mark a type with this attribute to enable controls to participate in Tab and Enter Stop functionality in the Layout Designer.

ToolboxBitmapAttribute Mark a type with this attribute to enable controls to specify a custom icon that will display in the toolbox next to the control names.

EmptyTypeConverter Mark a type with a `TypeConverterAttribute` to enable controls to completely hide their properties in the Properties window in the Layout Designer.

LayoutControlDesigner Mark a type with a `DesignerAttribute` to enable the right-click "Edit Properties" menu in the Layout Designer.

ICustomClipboardSupport Implement this interface for controls that interact with ACT!'s clipboard functionality (cut/copy/paste/delete/undo).

SpellCheckableSupportAttribute and ISupportSpellCheck Use these controls to support spell checking.

LayoutSingletonComponentAttribute Mark a type with this attribute to specify that a control can exist only once on a layout.

Binding custom controls to a field

You can bind custom controls to a field, similar to binding a `Field` or `Memo` control in the Layout Designer. You must do the following to implement a bound custom control:

- Implement `IXXXListBoundControl`, where XXX is `Contact`, `Group`, or `Company`. This forces the control to provide a list component property, which is how it will get the context of the current entity. Once the list is set, you can attach the control to the `PositionChanged` and `ItemChanged` events on the `CurrencyManager` (standard .NET databinding) to be notified when the current entity changes. See SDK samples.
- Implement `IXXXFieldBoundControl`, where XXX is `Contact`, `Group`, or `Company`. This forces the control to provide a field descriptor property ("[Metadata](#)" on page 13), so the control can get and set values in the entity when it is updated. See SDK samples.

- Implement `IUpdateableComponent`. This tells the control when to update changes on the underlying entity. See SDK samples.

Custom Tab

You can enrich the Application by creating a custom tab, which may contain its own controls and interface. You must use a plugin to create a custom tab. The plugin adds a new tab to a layout after the layout loads. Typically, a plugin will attach to the `LayoutChanged` event on the `ApplicationState` object and add a tab using the `UILayoutDesignerManager.AddTabToCurrentLayout()` method. This method takes a `.NET TabPage` as a parameter. The plugin must create the tab and add controls to its collection. See SDK samples.

Chapter 5

Sample Code

This chapter contains two samples of code: the first shows the use of Framework metadata; the second shows getting a contact list.

Using Framework metadata

The following is an example of using Framework metadata. We are printing the field display name of any contact string fields that can be edited, do not allow empty values, and do not have any default value (thus, string fields are 'blank' when we create a new contact) :

```
// filter to just return string fields on a Contact
ContactFieldDescriptor[] fields =
framework.Contacts.GetContactFieldDescriptors(new Type[] {typeof(string)});

// we're going to look for editable fields that don't allow empty values
// and don't have default values, so we'll need these attribute types
Type allowsEmptyType = typeof(AllowEmptyFieldAttribute);
Type defaultValueType = typeof(DefaultFieldValueAttribute);

// initialize our attributes
AllowEmptyFieldAttribute allowsEmptyFieldAttr = null;
DefaultFieldValueAttribute defaultFieldAttr = null;

AttributeCollection attributes = null;
ContactFieldDescriptor contactField = null;
for (int i=0;i<fields.Length; i++)
{
    contactField = fields[i];

    // make sure we can modify this field
    if ( !contactField.IsReadOnly )
    {
        attributes = contactField.Attributes;

        // check if we don't all empty values
        allowsEmptyFieldAttr = attributes[allowsEmptyType] as
AllowEmptyFieldAttribute;
        if ( allowsEmptyFieldAttr != null && !allowsEmptyFieldAttr.AllowEmpty )
        {
            // now check to see we don't have a default value
            defaultFieldAttr = attributes[defaultValueType] as
DefaultFieldValueAttribute;
            if ( defaultFieldAttr == null || defaultFieldAttr.DefaultValue == null
            )
            {
```

```
        // we found one
        Console.WriteLine(contactField.DisplayName);
    }
}
}
```

Getting a contact list

The following sample shows how to get a contact list.

```
// get the company field descriptor
DBFieldDescriptor companyField =
framework.Contacts.GetFieldDescriptor("TBL_CONTACT.COMPANYNAME", true);

// get contacts I have access to, sorted by company
ContactList contacts = framework.Contacts.GetContacts(
new SortCriteria[]{new SortCriteria(companyField,
ListSortDirection.Ascending)});
```

Index

Symbols

.NET design-time attributes and types 18
.NET Framework 1
.NET serialization techniques 19

A

ACT! design-time attributes, interfaces, and types 19
ACT! platform 1
ACT! version support 3
Act.UI.Core.dll 18
Act.UI.dll 17
ActApplication class 17
ActFramework class 13
ActFramework Managers 13
Activities 9
add-on components 2
Application 5
Application state 18
Application tier 1
assemblies 2

B

Business logic tier 1

C

CategoryAttribute 18
Collections 14
Companies 9, 10
components
 add-on 2
 third-party 2
Contacts 9
Crystal Reports 7
Custom controls 6, 18
 binding to a field 19
Custom tab
 creating using a plugin 20
Custom tabs 6, 7

D

data
 extended 9
 primary 9
 working with 15
Data logic tier 1
DefaultValueAttribute 19
DescriptionAttribute 18
DesignerSerializationVisibility 19
design-time
 attributes 19
 interfaces 19
 types 19
Design-time components 1

Design-time related instruments 18
Divisions 10
DLL's
 add-on 2
 dependant 2

E

EmptyTypeConverter 19
entities 9
entity lists 14
 binding to grid controls 14
exception handling 3
Extended data 9
Extended entities 9

F

Field descriptors 13
FieldCollection indexer 14
Focuments 9
Framework 5
Framework metadata example 21
Framework tier 1

G

Groups 9, 10

H

Histories 9

I

ICustomClipboardSupport 19
Implement IUpdateableComponent 20
Implement IXXXFieldBoundControl 19
Implement IXXXListBoundControl 19
IPlugin interface 3, 17
ISupportSpellCheck 19

L

Layout Designer 18
LayoutControlDesigner 19
LayoutSingletonComponentAttribute 19
LayoutToolboxFriendlyNameAttribute 19
Lists 14
logging in to ActFramework 13

M

Managers
 ActFramework 13
 UI 17
Menus 1, 18
Metadata 13

N

Notes 9

O

OLEDB Provider 7
Opportunities 9, 11

P

- Platform tiers 1
- Plugins 6, 17
 - adding menu items 18
 - third-party DLL's 2
- Presentation logic tier 1
- Primary data 9
- Primary entities 9
- PropertyDescriptor 13

S

- Secondary contacts 9
- security, with OLEDB provider 7
- serialization 19
- ShouldSerializeXXX method 19
- SortCriteria 14
- SpellCheckableSupportAttribute 19
- state, of the application 18
- Subgroups 10

- supported assemblies 2

T

- TabableAttribute 19
- tabs, adding custom 7, 20
- third-party components 2
- Tier
 - Application 1
 - Database 1
 - Framework 1
- Toolbars 1, 18
- ToolboxBitmapAttribute 19

U

- UI Managers 17

V

- version support 3
- Views 1, 17
 - extending with custom controls 6