Getting Started with ATF

Revision History

This guide provides information about getting started with ATF, including installation and use.

Version	Revision Date	Author(s)	Comments
1.0	14-May-2013	Gary Staas	ATF version 3.5
1.1	31-July-2013	Gary Staas	ATF version 3.6. Remove Visual Studio 2008 support.

Public.

Table of Contents

Revision History 2

About This Guide 4

How to Use This Document 4

ATF Document Set 4

Who Should Use This Guide 4

Supported Tools 5

Migration From Earlier Releases 5

Resources 5

Typographical Conventions 6

1 **Introduction to ATF 8**

Features and Benefits 8

Very High Level View of ATF 8

2 **Installing ATF 10**

Package Manager 10

Installing 10

3 **Managed Extensibility Framework (MEF) 12**

Advantages 12

ATF Components 12

4 **ATF Frameworks 14**

Application Shell Framework 14

Document Framework 15

Context Framework 17

Property Editing Framework 19

Instancing Framework 20

Adaptation Framework 21

Document Object Model (DOM) Framework 22

Commands 22

5 **ATF structure 24**

Frameworks 24

Namespaces 24

Functional Areas 25

Assemblies 25

6 **ATF Samples 27**

Descriptions 27

Components in Samples 27

Building Samples 28

7 **Building an Application Based on an Existing Sample 29**

8 **Next Steps 30**

About This Guide

This guide introduces you to ATF and gets you started using ATF to create rich Windows client applications and game-related tools.

How to Use This Document

Read this document first, before the other ATF documents described in "ATF Document Set". Follow the guide's directions to install ATF in "Installing ATF".

The guide contains these chapters:

- "About this Guide" (this chapter) tells you about this guide and other ATF documentation, what you need to know to work with ATF tools, and informs you about available resources.
- "Introduction to ATF" tells about ATF's benefits and describes ATF at a very high level.
- "Installing ATF" shows how to install ATF and describes what you get.
- "Managed Extensibility Framework (MEF)" discusses why MEF is useful and how ATF uses MEF to create its own components.
- "ATF Frameworks" describes the main frameworks inside ATF.
- "ATF structure" provides some ways to view ATF to get insight into its structure.
- "ATF Samples" tells what the samples do and how to build them, and describes some components created by samples.
- "Building an Application Based on an Existing Sample" outlines building an application, starting with an ATF sample.
- "Next Steps" guides you in learning more about ATF.

ATF Document Set

This is the first volume in the ATF document set, described in Table 1.

Table 1 ATF Document Set

Volume	Version	Comments
Getting Started with ATF	3.5	This document: Introduces ATF concepts and ATF samples.
ATF Programmer's Guide	3.5	Shows using ATF to create applications and tools. (To be done)
ATF Programmer's Guide: Document Object Model (DOM)	3.5	Shows how to use the ATF DOM for application data.
ATF Glossary	3.5	Glossary of ATF technical terms. (To be done)
ATF Refactor User Guide	3.5	Guide to using the ATF Refactor tool.

In addition, complete API class and method reference documentation for ATF is available as comments in the source code itself and as part of Visual Studio's Intellisense.

The ATF documentation page on SHIP contains the latest versions of the documentation. The ATF distribution also includes all the documentation in its `Docs` folder.

You can also check for updates on the ATF project page.

Who Should Use This Guide

This document is directed to game programmers and tools programmers.

To get the most out of using ATF, programmers need the following:

- Experience in C# and .NET development. Framework development experience is helpful.
- Visual Studio 2012 or 2010.
- Access to *SHIP*, the *SHared Information Portal* for Sony Computer Entertainment (SCE) worldwide game development.

Supported Tools

ATF supports the following tools:

- Visual Studio 2012 and 2010. Visual Studio 2010 and 2012 projects are interchangeable.
- All of the core code in the foundation compiles with C# 3.0. Over time, there will be additional support for C# 4.0 covariance and contravariance, using the `in` and `out` keywords, but a C# 3.0 equivalent will be available.

Migration From Earlier Releases

The ATF 2.8 components, data models, and so on, are supported in ATF 3. You can continue to build your tools as usual, while taking advantage of new components and features that ATF 3 provides.

There are many class and method name changes from ATF 2.8 to ATF 3. The `AtfRefactor` tool automates the first step in the migration process by updating those class and method names that have well-defined mapping rules and that affect very few lines in the original source code. See the *Atf Refactor User Guide* for more information.

Resources

Many resources are available for ATF.

Samples

The ATF distribution includes a full set of complete, running sample applications that demonstrate various aspects of the ATF architecture. These applications reside in the ATF\Samples folder. Documentation for these tutorials is available from the ATF Code Samples page on SHIP. Build the samples using the Samples solution in `\components\wws_atf\Samples`.

SHIP Resources

SHIP contains several pages about ATF:

- ATF Project Home
  This ATF page has team contact information and links to video tutorials, news, forums, downloads, the bug tracker, and various wikis.
- ATF documentation
  Download documentation files here.
- ATF Code Samples
  Samples' description.
- ATF Programming Guidelines
  Programming and code style guidelines followed by the Authoring Tools Framework team. Although you are not developing ATF itself, these are useful development guidelines.

If you have any trouble viewing these pages, send an email to support@ship.scea.com to request access.

SCE Developer Community Resources

In addition to providing useful resources, SHIP is home to a vibrant SCE developer community that includes the ATF team. Visit SHIP to contact the ATF team, read blogs, request features, report bugs, and discuss topics of mutual interest with other SCE developers.

Forums

Have a question? Found a bug? Need information that isn't in the documentation? On the Authoring Tools Framework Forum, you can follow the discussions or post a new thread about your hot topic. The ATF team frequently monitors the forum, so a team member may be the first responder to your query.

Reporting Bugs and Requesting Features

If you want to report a bug or request a feature, visit Trackers for bugs and feature requests.

If you don't see your issue there, post a new thread on the ATF Forum.

Contacting the ATF Team

To contact the ATF development team, see the Owner on ATF project home. You can raise issues to the entire team with the ATF Forum.

Other Reading

Some other publications provide useful information:

- *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries (2nd Edition)* by Krzysztof Cwalina and Brad Abrams. This is an excellent book whose guidelines ATF follows in developing a reusable extensible framework. MSDN has much of the book's content as well as the naming guidelines.
- *Programming C# 5.0, Building Windows 8, Web, and Desktop Applications for the .NET 4.5 Framework* by Ian Griffiths, O'Reilly Media.

Typographical Conventions

This document follows the typographical conventions in Table 2.

Table 2 Document typographical conventions

Font	Used for	Examples
monospace	Paths and file names	`wws_atf\Samples\CircuitEditor`

	Scripts	`function ShouldRevive()`
		`  return m_shouldRevive`
		`end`

	Code elements and code	`public Command(string description)`
		`  {`
		`  m_description = description;`
		`  }`

	Formatted text, such as XML	`<?xml version="1.0" encoding="utf-8" standalone="yes"?>`
		`<StringLocalizationTable>`
		`  <StringItem id="Cut" context="" translation="Cut" />`

bold	User interface elements	**Add** button

Font	Used for	Examples
	Menu item and other paths	**Build > Build Solution**
italic	New technical terms	*Document Object Model*
	Emphasis	Do *not* do this.
	Document titles	*Getting Started with ATF*
bold italic	Special emphasis	***MEF Catalog***
blue	URLs (external)	ATF project home
	Links inside a document	"Application Shell Framework"
quoted	Exact text	"ActiproSoftare SyntaxEditor"
	Names	"Add Layer"

1 Introduction to ATF

The Authoring Tools Framework (ATF) is a set of C#/.NET components that enable game tool developers to build rich Windows client applications and game-related tools.

Features and Benefits

ATF offers these benefits:

- ATF provides flexibility and adaptability in creating applications, using components that you can mix and match, from grid property editors to components that provide standard sets of commands.
- ATF contains many useful controls, such as tree controls, a property grid that can embed custom controls, a grid control that can embed custom controls, diagramming controls, and useful dialog controls.
- You can develop many kinds of applications with the rich set of components and tools provided by ATF. Among others, ATF has been used to create sound modeling and audio bank tools, level editors, character animation blending tools, scripting language debuggers, and timeline/sequence tools.
- Developers can focus on providing the unique values of their application, rather than having to redevelop the GUI elements ATF provides.
- Tools built with ATF have a common GUI and behavior, enabling ATF tool users and designers to quickly come up to speed in their use. This can save time in coding as well as in end-user documentation, training, and support.
- As an ATF tool developer, you can write new tools to assist with game development, either from scratch, by starting with one of ATF's sample applications, or by modifying existing tools and underlying schema.
- ATF consists of parts that typically reference each other through C# interfaces, which makes it easy to customize or replace them.
- The ATF Document Object Model (DOM) supports both XML and conventional data representations to enable support of any kind of application data store, including managed data stores.
- ATF comes with a rich set of samples that illustrate using ATF. These samples show ATF elements of particular interest to game developers, and some of these applications can serve (and have served) as starting points for full featured tools.

Very High Level View of ATF

ATF is a software framework for developing applications that are especially useful to game developers. ATF can also be regarded as an extension of the .NET framework, because it is a set of C#/.NET components. ATF has followed design guidelines to provide a unified programming model with .NET to enhance developer productivity.

ATF is oriented toward developing the kind of applications game developers can use with the controls and components it provides. For instance, it offers a statechart renderer to use in developing an editor that shows states and transitions, as illustrated in the State Chart Editor sample.

Some ATF classes are constructed as *Managed Extensibility Framework (MEF) components*, and this allows incorporating them into an application very easily. These components offer capabilities common to many applications, such as adding a **File** menu with its standard commands, providing standard file dialogs, or adding a palette of objects to a UI, so you don't have to develop these pieces again. For more information, see "Managed Extensibility Framework (MEF)".

Applications typically allow for creating and editing data, and the following data-related functionality is usually necessary: keeping multiple views of the data in sync, undo/redo, cut/copy/paste, and persistence in a file. The Document Object Model (DOM) is a simple hierarchical in-memory database that

provides a mechanism for observing changes in the data, and can be used to provide these necessary features. The DOM is very similar to an in-memory XML file, and classes are provided to persist application data in XML files. The DOM structure can be defined using schema files. However, the DOM and XML-support are completely independent, and one is not required for the other. For information, see "Document Object Model (DOM) Framework" and "Document Framework".

Applications often support viewing and editing multiple documents at once, each with their own selection set and their own undo/redo history, for example. ATF calls these independent groups of data *contexts*. In most applications, a context is associated with a single document that is loaded into memory, but documents and contexts are separate. A single physical document can contain multiple contexts and a single context might know about data in multiple documents. ATF provides a framework for handling contexts. Editors and various tools and commands typically make use of the currently active context. For more details, see "Context Framework".

Adaptation is a technique used in ATF to allow one type of C# object to look like another, without using inheritance. If the DOM is used, adaptation is especially important, because it allows the basic database class, `DomNode`, to look like a more convenient application-specific class throughout your code. This also allows the data representation to be independent of "business logic". In terms of software engineering design patterns, adaptation consists of the Adapter and Decorator pattern. For more information, see "Adaptation Framework".

You can develop applications using both Windows Forms (WinForms) and Windows Presentation Foundation (WPF), as shown in the Win Forms App and Wpf App samples. For more information on these samples, see "ATF Samples".

2 Installing ATF

This chapter shows how to install ATF and the resulting files you get. ATF is part of the TNT Game Technology SDK.

In this section, "component" refers to a general software component.

Package Manager

Use Package Manager to install ATF. You can download it at Package Manager download. For general information about Package Manager, see WWS SDK Package Manager.

Installing

To install the full release, run Package Manager as follows:

(1) Create a folder in which to install ATF, such as `c:\atfsdk`.

(2) Open a Windows Command Prompt window.

(3) Enter this command:

```
wwspm install wws_atf -path C:\atfsdk
```

(4) The following prompt appears:

```
wwspm v0.7.9, built Feb 21 2013 14:52:58
No WWS SDK installation found.  Would you like to create a new one at
"c:\atfsdk" ?
[yn]:
```

(5) Enter "y" and press Return. Package Manager finds the ATF component and displays the following message:

```
WWS SDK installation found at "C:\atfsdk".
Analyzing components...
Components to install:
  [wws_atf,3.5.0.14]

Would you like to proceed? [yn]:
```

(6) Enter "y" and press Return. Package Manager installs the ATF component and lists it:

```
Retrieving components...
Deploying component [wws_atf,3.5.0.14] to "c:\atfsdk"...
```

The version number of the ATF component is shown.

The installation contains the following folders:

- `bin\wws_atf`

  Pre-built third party and other required DLLs and ATF sample applications' executables are in the `Release.vs2010` folder.

- `components\wws_atf`

  Readme and Release Note files, as well as these folders:

  - `DevTools`: Source and `.sln` files for building various tools, such as the `AtfRefactor` and `DomGen` tools.
  - `Docs`: ATF documentation. For a documentation list, see "ATF Document Set".
  - `Framework`: ATF source in folders corresponding to Visual Studio projects.
  - `Legacy`: Legacy documentation, source, and samples, pertaining to earlier ATF versions.

- `Samples`: ATF samples including `.sln` and `.csproj` files and full source. For details, see "ATF Samples".
- `Test`: Source code including `.cmd`, `.sln`, and `.csproj` files for various test programs.
- `ThirdParty`: Licensing and other information, DLLs and/or installers for the third party software used in ATF.

- `lib`

  Static library DLLs for ATF components, with subfolders for each platform.

3 Managed Extensibility Framework (MEF)

ATF makes extensive use of Microsoft's *Managed Extensibility Framework (MEF)*, which allows an application to easily add or remove *components*. A component is either an object or a C# type that is added to a special MEF container, so that the component can discover and use other components without having a hard-dependency between them.

Microsoft's MEF documentation calls these components *extensions* and *parts*; this document and the other ATF documents use the term *component* (or *MEF component* if required for clarity).

ATF contains many components to easily provide capabilities common to many applications, so components are an integral part of ATF. Many of these components can be used independently of MEF.

Because of the importance of components in ATF, they are discussed before ATF, per se.

For a quick introduction, see 5 Minute Tutorial on Managed Extensibility Framework (MEF). For a high level how to guide, see the Microsoft blog MEF for Beginner. For a detailed reference, see Managed Extensibility Framework on the MSDN Web site.

Advantages

ATF uses MEF because its plugin architecture offers many advantages:

- MEF provides a nice way to enable interdependent application components to discover each other at runtime.
- It supports many scenarios, supports lazy loading, and has good support for circular dependencies.
- MEF is non-intrusive: it does not require that you implement an interface or derive from a base class.
- It is easy to port to MEF.
- MEF is standard with the release of .NET 4.0 and later. It is included in the ATF SDK `ThirdParty` directory for .NET 3.5 projects.

ATF Components

ATF MEF components are C# classes decorated with attributes so components can discover each other. These components provide a rich set of capabilities, so you can concentrate on developing the unique functionality of your application—instead of redeveloping component's facilities common to many applications.

You don't need to instantiate or directly access these component classes. Instead, you simply list them in the MEF `TypeCatalog` in your `Main()` function and perform some standard initialization, as shown in most of the sample applications.

After initialization, you may not need to do anything to make use of ATF components' capabilities: some components require no additional code to function. For instance, the `StandardFileExitCommand` component adds the **File > Exit** menu item that closes the application, and it requires no additional code to do this.

Other components require the application to provide some functionality. For example, the `StandardFileCommands` component adds standard **File** menu commands for **New**, **Open**, **Save**, **SaveAs**, and **Close**, as well as displaying dialogs for opening and saving files. The application, in turn, provides its own component containing code to handle these file commands: processing opened file data, displaying it, and so forth. In other words, the application performs its own specialized functions for each menu command.

Some of these components work with each other, so all must be present to function properly. The `CommandService` component, for instance, handles commands in menus and toolbars. If you

include `StandardFileExitCommand` but leave `CommandService` out, the **File > Exit** menu item does not appear. In fact, your application would have no menus at all (unless you added them some other way). If a component requires other components in order to function properly, this is noted in the component's documentation.

As noted, you don't have to do anything to get components to discover each other. MEF provides the framework to allow components to work together to accomplish their tasks.

Some ATF components are fundamental and are used in nearly all applications, such as the ones described in "Application Shell Framework". These and other components are discussed in the chapter "ATF Frameworks".

You can also create new components and add them to the MEF `TypeCatalog` to allow MEF to integrate them into your application. Your new component classes need to conform to certain requirements to work with MEF, such as providing certain attributes and constructors with the appropriate signatures. As a guide to creating new components, several samples create and use components; see "Components in Samples" for a description of these components.

4 ATF Frameworks

This chapter describes the frameworks and elements underlying ATF, discussing common components, classes, and interfaces.

Note that all ATF components can be modified or replaced altogether with your own components if the standard component does not meet your needs.

Application Shell Framework

The *Application Shell Framework* consists of the core application services and interfaces needed for applications with a GUI:

- `ControlHostService`: provides a dock in which to hold controls, such as menus and toolbars. Its interface is `IControlHostService`. The docking control allows moving windows around on the dock, and the window configuration can be saved.
- `CommandService`: displays currently available application commands to the user in menus and toolbars and provides keyboard shortcuts. Its interface is `ICommandService`.
- `StatusService`: adds a status bar at the bottom of the main form. Its interface is `IStatusService`.
- `SettingsService`: manages user-editable settings (preferences) and the persistence of these application settings. Its interface is `ISettingsService`.

These services are all components, so you don't need to configure them. Using these components saves you the trouble of developing these common capabilities yourself.

Figure 1 shows the Application Shell Framework's components.

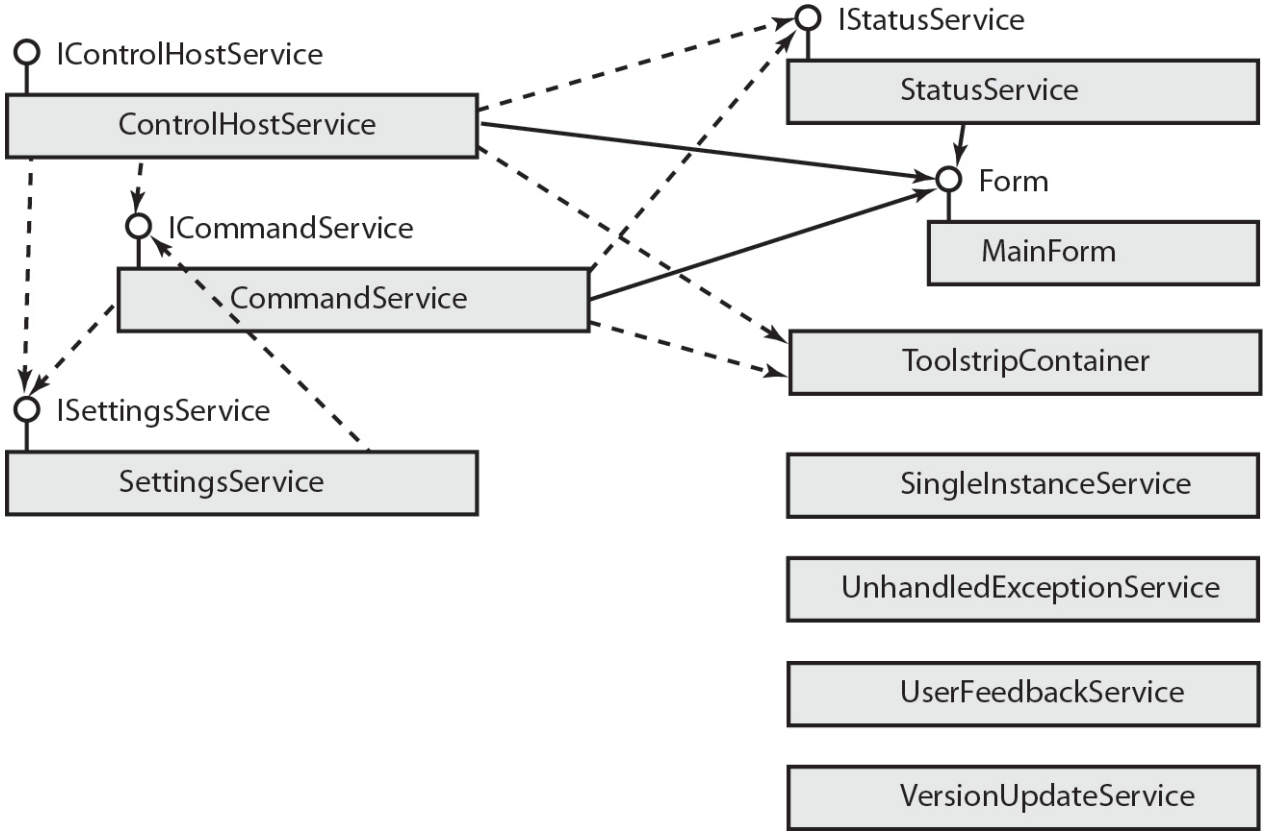
As the diagram indicates, you can add other components to the application to implement single-instance applications (`SingleInstanceService`), unhandled exception recovery (`UnhandledExceptionService`), user feedback (`UserFeedbackService`), and automatic version update (`VersionUpdateService`). All these components are independent and can be omitted or replaced.

As an example of customization, the figure also shows that you can add a `System.Windows.Forms.ToolStripContainer` to present command toolbars to the user.

Although this figure shows a `MainForm`, the Application Shell framework can also be used with Windows Presentation Foundation (WPF) applications. There are versions of these components and interfaces for both WinForms and WPF. `SettingsService` does not interact with a GUI, so it can be used with both WinForms and WPF.

All these components are in the `Sce.Atf.Applications` namespace.

Figure 1 Application Shell Framework

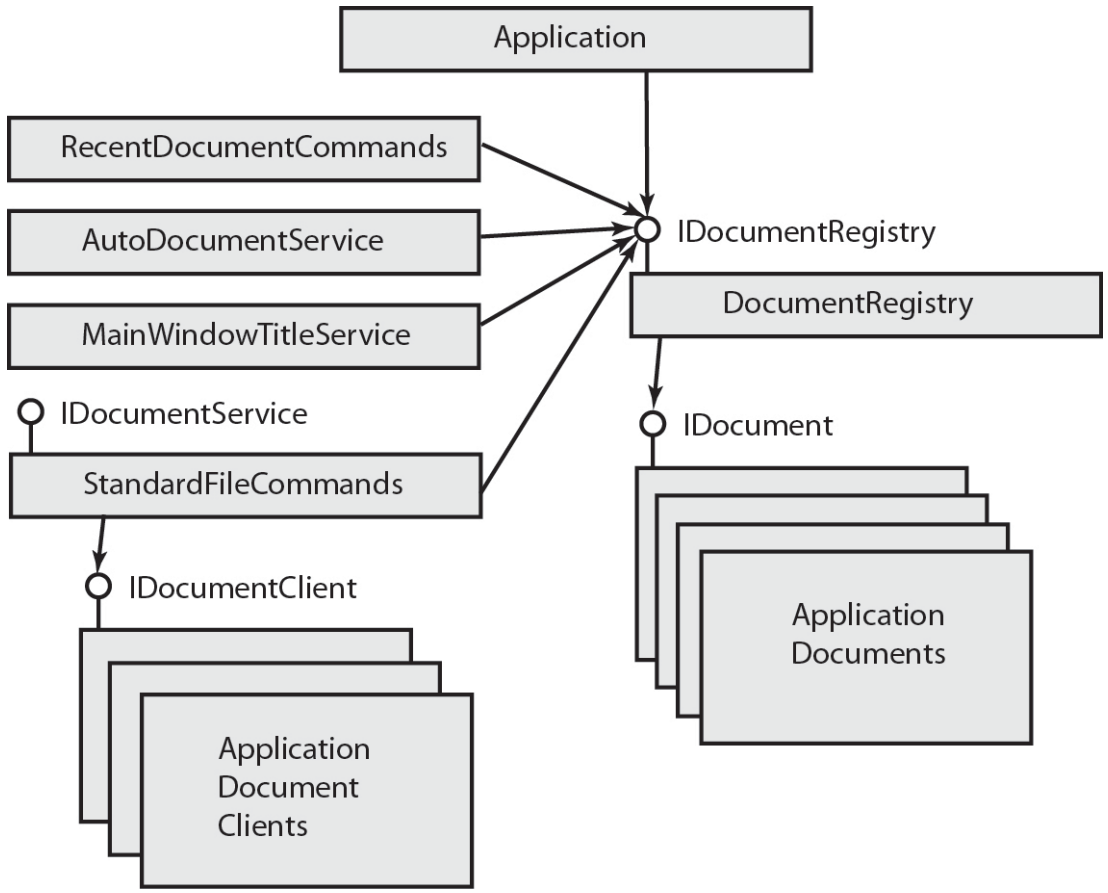


Document Framework

A document is a file that an application opens and a user can edit in some fashion. A document's contents could vary from unformatted text to graphics in a scene that a level editor might create.

The *Document Framework* enables components in an application to track documents that a user is working on and which document is currently active. It includes interfaces and components for working with application documents, as seen in Figure 2.

Figure 2 Document Framework



The Document Framework contains the following interfaces, all in the `Sce.Atf.Applications` namespace, except for `IDocument`:

- `IDocument`: the interface for application documents, which provides read-only and dirty properties. This is in the `Sce.Atf` namespace.
- `IDocumentService`: the interface for document services, such as Open, Close, Save, and Save As, and to determine whether or not a document has a title. Notifications allow components to track what the user is doing with the document. Any application managing documents uses `IDocumentService`.
- `IDocumentRegistry`: the interface to the document registry. The document registry holds documents, provides notifications when a document is added or removed, tracks the most recently active document and the open document contexts, and filters by document type.
- `IDocumentClient`: opens, closes, and displays open documents. Applications should implement a document client for each document type that they can read or write, because this client works with the actual document data.

The Document Framework components (services) are all in the `Sce.Atf.Applications` namespace:

- `StandardFileCommands`: implements **File** menu commands that modify the document registry: **New**, **Open**, **Save**, **SaveAs**, **Save All**, and **Close**. **New** and **Open** commands are created for each `IDocumentClient` component in the application. Since `StandardFileCommands` implements `IDocumentService`, `StandardFileCommands` is easy to replace if needed.
- `AutoDocumentService`: opens documents from the previous application session or creates a new empty document when an application starts.
- `RecentDocumentCommands`: adds recently opened documents to the application's **File > Recent Documents** submenu.

- `DocumentRegistry`: tracks the open documents in an application. It can retrieve the active document or most recently active document of a given type. Since all components access the interface `IDocumentRegistry`, you can replace the ATF `DocumentRegistry` with a customized version.
- `MainWindowTitleService`: updates the main dialog's title to reflect the current document and its state, i.e., modified or not.

Context Framework

The *Context Framework* tracks and works with application contexts.

A *context* is a "logical view" of some kind of application data that is presented to a user for viewing or editing. ATF has a variety of contexts, supporting various capabilities. A context can be very general, working with different kinds of data.

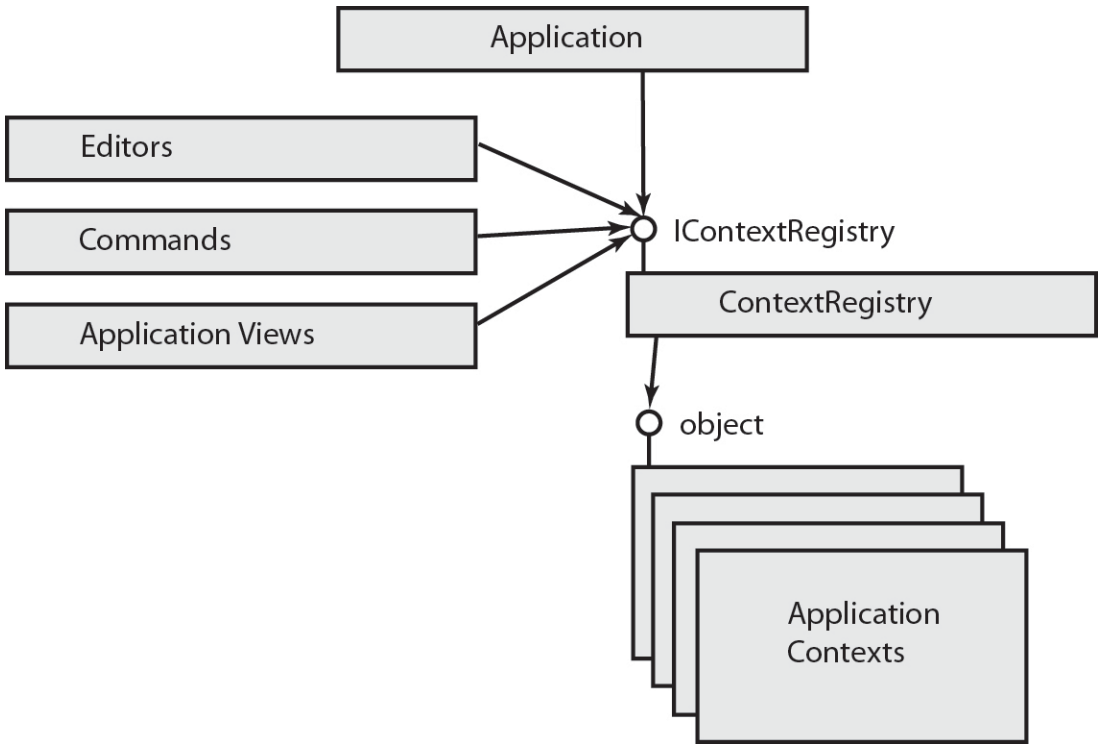
Application components need context information to track what the user is doing, so that they perform the correct operation on the correct object. Application components can use the Context Framework to track documents currently in the system, check whether a path has changed, update a tab (from inactive to active, for example), undo the last command, keep track of the currently selected objects, and so on.

A *context registry* tracks the active contexts in the application. ATF's `ContextRegistry` component implements the `IContextRegistry` interface. `IContextRegistry` maintains active and open contexts, so you can obtain the active context or the most recently active context. `IContextRegistry` also has events to track context changes. `IContextRegistry` makes it easy for applications to track what the user is working on and what is currently active. For example, commands can use `IContextRegistry` to check whether or not their commands are doable.

All ATF components access the registry using `IContextRegistry`, so the ATF `ContextRegistry` can be replaced with a customized context registry.

An application may include Editor, Command, and Application View modules, which all need context information. These modules can access the Context Framework with `IContextRegistry`, as shown in Figure 3.

Figure 3 Context Framework



Each context type has an associated interface indicating what it can do. Context interfaces are designed to perform common application tasks, such as getting the current selection or determining when data has changed. They are designed to be easy to implement and are specific to a particular task.

The main context interfaces are:

- `IViewingContext`: For contexts where items can be viewed. Its methods check whether objects are or can be made visible, or are framed or can be framed in the current view. Note that this context could operate on many different kinds of data; this is true of other contexts as well.
- `ILayoutContext`: For contexts that support positioning and sizing items. It contains methods for getting and setting item bounding rectangles and for determining which item bounds can be set.
- `IValidationContext`: Useful for marking when the user begins and ends logical commands. Validators can wait to check constraints until after an ending event, so the validator can allow invalid states as data changes, but before validation ends, it can raise an exception if a constraint is not satisfied. Observers of the data can respond efficiently: instead of responding to every change in data, they can wait for validation and then update in one operation. Operations can also be cancelled.
- `ISelectionContext`: Distills what it means to have a selection in a context. It provides methods for getting the objects in a selection, setting the selection, and getting the last selection. It also provides filtering: you can get the last selected object of a certain type. It has events detecting before and after a selection changes. It has an efficient method for determining if a selection contains an object.
- `IPropertyEditingContext`: Allow property editing components to get properties and items with properties, enabling property editing controls to display objects and their properties.
- `IObservableContext`: Allow observers of list and tree data to track changes. If an application finds that the active context implements `IObservableContext`, it can implement an editable view that stays in sync with the data. If applications do not find `IObservableContext`, they can assume the view is read-only and still consume data.

- `ITransactionContext`: Useful for implementing transactions on data. `ITransactionContext` simplifies editor design. It allows transactions to be cancelled at any time before calling the `End()` method. Different modules can contribute to an operation without having to know about each other, and if an exception occurs in one module, the whole transaction can be rolled back.
- `IInstancingContext`: For contexts that can instance objects; instancing requires the ability to copy, insert, and delete items. This interface integrates well with the Windows clipboard, because it uses `System.Windows.IDataObject`, which is used by the Windows Cut, Copy, Paste, Delete, and drag and drop commands. `IInstancingContext` should be used by any code that wants to create new instances, such as "Group"/"Ungroup" commands.
- `ILockingContext`: Support read-only documents and lockable objects. `ILockingContext` determines whether an object is locked, so the object cannot be changed or deleted.
- `IHistoryContext`: Enable implementing "Undo"/"Redo" commands. It tracks what has changed since the document was last saved.

`ContextRegistry` and `IContextRegistry` and most of the context interfaces mentioned here are in the `Sce.Atf.Applications` namespace. `IValidationContext`, `IObservableContext`, `ITransactionContext`, and `ILockingContext` are in `Sce.Atf`.

Property Editing Framework

The *Property Editing Framework* works with contexts that allow properties to be edited by controls. ATF offers a variety of *property editor* controls to edit object properties of various types.

If a context allows property editing, it implements one of these interfaces in the Property Editing Framework:

- `IPropertyEditingContext`: interface for contexts in which properties can be edited by controls.
- `ISelectionContext`: interface for selection contexts to get the current selection and be notified when the selection changes. (Also in the Context Framework.)

The following are the Property Editing Framework components to edit properties (all in `Sce.Atf.Applications`):

- `PropertyEditor`: edit object values and attributes using the ATF `PropertyGrid`, which is a two-column editing control.
- `GridPropertyEditor`: edit object values and attributes using the ATF `GridControl`, a spreadsheet-style editing control.
- `PropertyEditingCommands`: provide property editing commands in a context menu that can be used inside property editor controls like `PropertyGrid`.

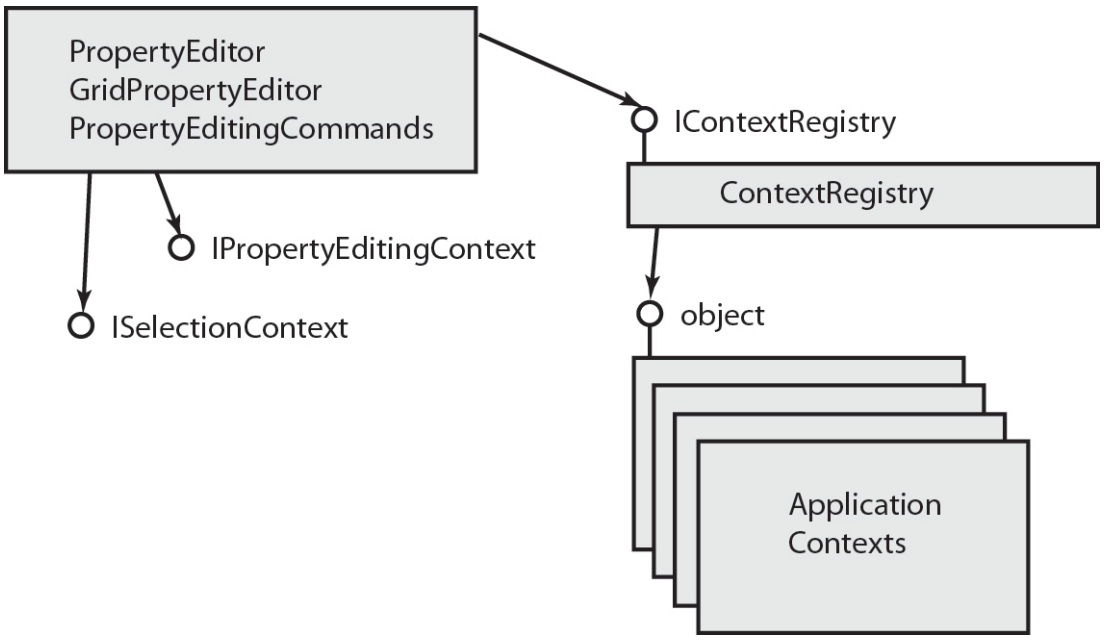
Property Editing Framework components use the Context Framework to get the current context.

ATF property editors first check if the active context implements `IPropertyEditingContext`. If not, they determine if the `ISelectionContext` interface is implemented, and, if so, operate in standard .NET fashion, using `ICustomTypeDescriptor` or reflection to get object properties.

`PropertyEditingCommands` operates in a similar way. It provides the property operation commands **Reset All** and **Reset Current** in a context menu that appears when the user right-clicks a property editing control. `PropertyEditingCommands` can be used with ATF property editors or with custom property editors.

As seen in Figure 4, the Property Editing Framework components access the Context Framework using the `IContextRegistry` interface. The Context Framework can be replaced with a custom framework.

Figure 4 Property Editing Framework



Instancing Framework

The *Instancing Framework* works with object instances that can be edited, i.e., copied, inserted, or deleted. It enables inserting objects, such as typing text onto a page, pasting from a clipboard, or dropping a circuit element onto a canvas. The Instancing Framework allows making copies or clones of objects and deleting instances.

The Instancing Framework's key interface is `IInstancingContext`. It provides `CanCopy()` and `Copy()`, `CanInsert()` and `Insert()`, `CanDelete()` and `Delete()` methods. A context should implement this interface if it allows editing objects.

The Instancing Framework's main component is `StandardEditCommands` (in the `Sce.Atf.Applications` namespace), which provides editing commands.

Two parts of ATF use the Instancing Framework:

- `StandardEditCommands` uses `IInstancingContext` to implement copy, cut, paste, or delete operations.
- Drag and drop operations use the Instancing Framework to insert objects into a context.

Miscellaneous commands, such as grouping commands, also use the Instancing Framework to delete all the objects in a group or insert something else.

Figure 5 Instancing Framework

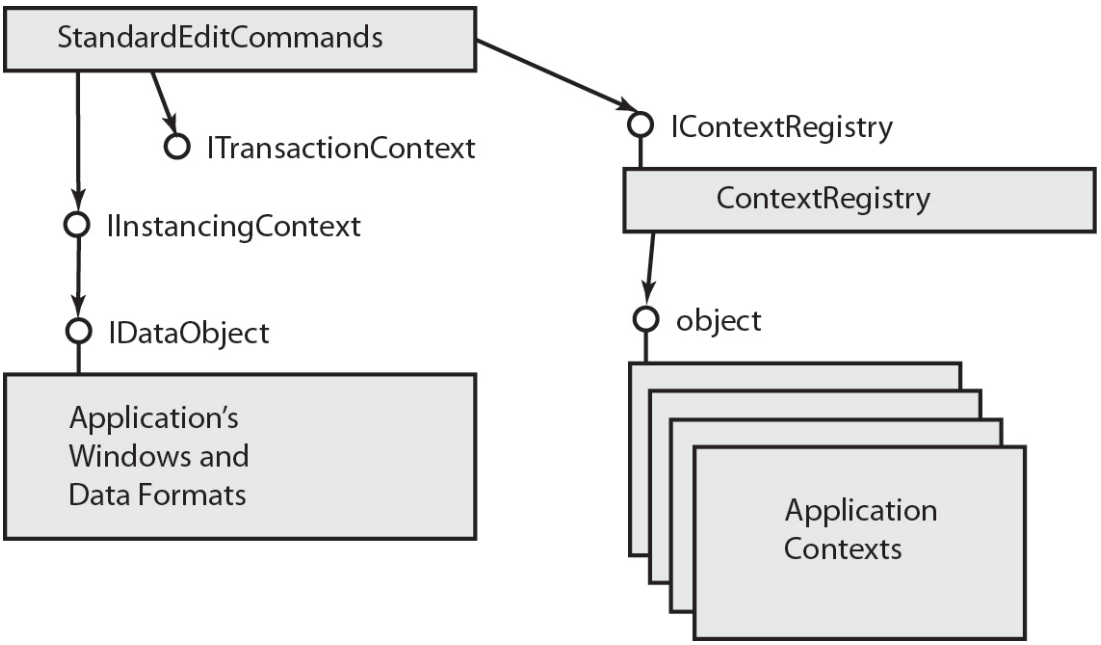


Figure 5 shows how the Instancing Framework's component `StandardEditCommands` references the Context Framework to access context information, such as the active context. The Context Framework can be replaced with a custom framework.

The `StandardEditCommands` component determines whether the active context implements the `IInstancingContext` interface. `StandardEditCommands` provides the **Edit** menu commands **Cut**, **Copy**, **Paste**, and **Delete**, using the `IInstancingContext` and (if available) `ITransactionContext` interfaces to perform the operations.

`IInstancingContext` uses the `System.Windows.Forms.IDataObject` interface, allowing integration with the Windows clipboard, so that data can be both exported from and imported into ATF applications from other applications. Drag and drop operations also use the `IDataObject` interface.

Adaptation Framework

The *Adaptation Framework* allows supporting different kinds of data models and managed data stores, such as a DOM or other CLR objects, by converting objects to other types. An *adapter* for `A` converts something of a different type than `A` to the same type as `A`. Adapters implement the `IAdapter` interface.

The adaptable interface, `IAdaptable`, obtains adapters to other types. You don't have to implement the interface on a particular type. You can use `IAdaptable` to extend type casting. This works for CLR types and DOMs.

`AdaptableControl`, and its Direct2D version `D2dAdaptableControl`, are base controls with adapters. An adapted control can simulate other controls, such as a circuit control. `AdaptableControl` and `D2dAdaptableControl` have the following advantages:

- These controls can be decorated with adapters to add new behavior. You can add functionality from outside the control without using inheritance. You can implement all the new functionality, such as label editing, in the adapter.
- To add a new feature, just add an adapter—you don't have to derive from a standard control class. This allows you to easily augment standard controls such as `CanvasControl`, `GraphControl`, `CircuitControl`, `StatechartControl`, `TimelineControl`, `CurveControl`, `TreeControl`, and `PropertyGrid`.

`AdapterCreator<T>` is a useful class that produces adapters of the given type `T`.

The Adaptation Framework's interfaces and controls are in the `Sce.Atf.Adaptation` and `Sce.Atf.Controls.Adaptable` namespaces.

Document Object Model (DOM) Framework

The *Document Object Model (DOM) Framework* provides a powerful, flexible, and extensible mechanism for loading, storing, validating, and managing changes to large amounts of application or game data, independently of application code. ATF does not require that applications use a DOM, but a DOM is an excellent way to represent application data.

Application data is stored in a tree of DOM nodes, with each DOM node having attributes and child DOM nodes. The DOM node attributes store primitive types of data, like integers and strings, or arrays of these types. The attributes and children available on a particular DOM node depend on the DOM node's type. Typically DOM node types are defined in the XML Schema Definition (XSD) language, but types can be defined in other languages as well.

The ATF DOM differs from conventional XML DOM implementations by converting document data to primitive type values and arrays, which makes it much more suitable for representing data like large triangle meshes. In addition, if your application's data can be stored as an XML document, you gain a lot by using ATF's DOM and Schema/XML support.

The DOM architecture also includes the concept of *DOM node adapters* (DOM adapters, for short), which enable you to define a C# API for your DOM data—separating game and application data from the code that operates on it.

The ATF DOM offers these advantages:

- Simple managed data store design.
- Easy and powerful DOM data adaptation, with notification events for each DOM node.
- Optional DOM node adapters for unique naming, reference tracking, implementing transactions, and name lookup.
- DOM node adapters to decouple the data storage (in DOM nodes) from how data is used.

The `Sce.Atf.Dom` namespace defines the ATF Document Object Model. Classes in this namespace include features to:

- Load and register XML schema type definition files that define the application's data types.
- Use schema metadata objects to define DOM data types.
- Create and manage repositories and collections of DOM objects, which contain the actual application data.
- Load and store DOM data, either in XML or in any format you define. The ATF DOM provides built-in support for XML data storage and retrieval.
- Implement and register DOM node adapters to create dynamic object APIs to the underlying DOM node objects.
- Define and manage DOM annotations in the schema to add custom properties or additional information to data types. For example, you can add an annotation to indicate the control to use to edit a data type or to specify a rendering proxy.
- Add constraints to DOM data to update a set of output data based on an input, such as a user action. For example, update a transformation matrix each time a 3D object is scaled or rotated.

See the *ATF Programmers Guide: Document Object Model (DOM)* for details on using the ATF DOM to manage application data.

Commands

An application usually needs to add custom commands for its menus, buttons, and other controls.

The `CommandService` component (or a custom component like it) is a service used in ATF to handle commands in menus and toolbars.

`CommandService` implements the `ICommandService` interface, which contains methods to register and unregister commands, display context menus, and process keyboard shortcuts.

The `CommandInfo` class provides information about a command, such as user interface name and keyboard shortcut. This class also sets up information for common commands, such as the standard **File** and **Edit** menu item commands.

`ICommandClient` is the interface you implement so that `CommandService` calls your command code back when the user clicks on your command's menu item or icon in a tool strip. `ICommandClient` also determines whether a command can be performed.

Several ATF components add sets of commands to an application:

- `StandardFileCommands`: add standard **File** commands.
- `StandardSelectionCommands`: add standard selection commands.
- `StandardShowCommands`: add standard **Show** menu item commands: **Show Selected**, **Hide Selected**, **Show Last Hidden**, **Show All**, and **Isolate**.
- `PropertyEditingCommands`: provide property editing commands that can be used in property editor controls, such as a property grid.

These components' code demonstrates how to add commands to an application.

The components and interfaces here are in the `Sce.Atf.Applications` namespace.

5 ATF structure

This chapter discusses several ways of visualizing ATF's structure. Use this chapter to help you find the facilities in ATF you need to code various parts of your application.

Frameworks

ATF is a framework that contains frameworks:

- "Adaptation Framework": convert objects to other types to support different kinds of data models and managed data stores, such as a DOM.
- "Application Shell Framework": add core application services needed for applications with a GUI.
- "Commands": create custom commands for application menus, buttons, and other controls.
- "Context Framework": track and work with application contexts.
- "Document Framework": track documents a user is working on.
- "Document Object Model (DOM) Framework": load, store, validate, and manage changes to application data, independently of application code.
- "Instancing Framework": work with object instances being edited.
- "Property Editing Framework": edit properties with controls.

For details, see the "ATF Frameworks" chapter.

Namespaces

ATF is organized under a single namespace, `Sce.Atf`, which has numerous nested namespaces.

Each namespace has an emphasis, although this is not rigid. Some of the key namespaces are described below; those marked with * are especially important.

- `*Sce.Atf`: Provide a variety of components, utilities, and interfaces. For example, it has general purpose file dialogs, loggers, file utilities, GDI utilities, Windows interoperability functions, LINQ (Language Integrated Query) classes, and math utilities. It contains major interfaces, such as `IDocument`, `IInitializable`, and the context interfaces `IValidationContext`, `IObservableContext`, `ITransactionContext`, and `ILockingContext`.
- `*Sce.Atf.Adaptation`: Classes that specialize in adapting objects to other types. It contains the interfaces `IAdaptable` and the key extension method `As<T>`. The "Adaptation Framework" resides in this and the `Sce.Atf.Controls.Adaptable` namespace, so these namespaces provide adaptation support in ATF.
- `*Sce.Atf.Applications`: This is the key namespace. Nearly all the components and interfaces described in the "ATF Frameworks" chapter reside here. (The exceptions are "Adaptation Framework" and "Document Object Model (DOM) Framework".) This namespace provides a great variety of services, including many components you can easily incorporate into applications.
- `Sce.Atf.Controls`: Many of ATF's controls are here, including special purpose dialogs and various tree related controls. This namespace provides many controls not in standard .NET Windows Forms, including controls for editing numbers and file paths, a canvas, and tree controls. `Sce.Atf.Controls.Adaptable` and `Sce.Atf.Controls.Adaptable.Graphs` also contain many controls.
- `*Sce.Atf.Controls.Adaptable`: Controls with adapters, including the fundamental `AdaptableControl`. The "Adaptation Framework" resides in this and the `Sce.Atf.Adaptation` namespace, so these namespaces provide adaption support in ATF.

- `Sce.Atf.Controls.Adaptable.Graphs`: Many classes that work with graphs, some controls using Direct2D.
- `Sce.Atf.Controls.PropertyEditing`: Property editor controls and associated classes to edit many kinds of data.
- `*Sce.Atf.Dom`: Classes that allow you to use a DOM for application data, including the very important `DomNode`, `DomNodeAdapter`, and `DomNodeType`. The "Document Object Model (DOM) Framework" is defined here.
- `Sce.Atf.Rendering`: Facilities for rendering 3D graphics. These are used by the Level Editor tool in the WWS SDK.
- `Sce.Atf.Rendering.Dom`: Classes that render data typically stored in a DOM, such as a 3D Scene. These are used by the Level Editor tool in the WWS SDK.
- `*Sce.Atf.Wpf`: This and its nested namespaces provide WPF facilities.
- `Sce.Atf.Wpf.Applications`: Variety of classes, components, and utilities for WPF. This namespace provides the "Application Shell Framework" components used for WPF.
- `Sce.Atf.Wpf.Behaviors`: Classes to adapt WPF behavior to ATF.
- `Sce.Atf.Wpf.Models`: Provide services to manage controls that use the Model-View-Controller (MVC) design pattern in WPF. With the MVC control framework, you attach data controls to an underlying data model (usually the DOM) through adapter client classes that you define. The adapter client exposes, filters, and presents the data model in a way the control understands.

Functional Areas

To work in various functional areas, see the namespaces and frameworks in Table 3.

Table 3 Functional Areas, Namespaces, and Frameworks

Functional Area	Namespaces	Framework
Adaptation	`Sce.Atf.Adaptation`	"Adaptation Framework"
Adaptable controls	`Sce.Atf.Controls.Adaptable` `Sce.Atf.Controls.Adaptable.Graphs`	"Adaptation Framework"
Commands	`Sce.Atf.Applications`	"Commands"
Components	`Sce.Atf.Applications`	"Application Shell Framework" for core application services
Controls	`Sce.Atf.Controls`	
Documents	`Sce.Atf` `Sce.Atf.Applications`	"Document Framework"
DOM	`Sce.Atf.Dom`	"Document Object Model (DOM) Framework"
Property editing	`Sce.Atf.Controls.PropertyEditing`	"Property Editing Framework"
3D rendering	`Sce.Atf.Rendering` `Sce.Atf.Rendering.Dom`	

Assemblies

ATF is divided into the following assemblies:

- `Atf.Atgi`: support for ATGI asset description files.
- `Atf.Collada`: support for Collada asset description files.

- `Atf.Core`: core application services, including components.
- `Atf.Gui.OpenGL`: OpenGL support.
- `Atf.Gui`: additional GUI support.
- `Atf.Gui.WinForms`: primary WinForms application support.
- `Atf.Gui.Wpf`: WPF application support.
- `Atf.IronPython`: Python services.
- `Atf.Perforce`: Perforce services.
- `Atf.SyntaxEditorControl`: syntax editor controls

These assemblies vary considerably in their number of constituent classes.

6 ATF Samples

This chapter describes ATF's samples, which help you understand and use ATF. It lists some components created by samples and tells how to build samples.

Samples reside in the folder `\components\wws_atf\Samples`.

For detailed information on samples, see the ATF Code Samples wiki page and the individual sample pages.

Descriptions

This section briefly describes each sample and links to its wiki page:

- Circuit Editor: Editor for circuits, consisting of modules with input and output pins and connections between them.
- Code Editor: Code editor that uses the *ActiproSoftare SyntaxEditor* to provide an editing control.
- Diagram Editor: Combination of Circuit, Finite State Machine, and State Chart editors into a single application to show how multiple editors can share an application shell and editor components.
- DOM Tree Editor: Editor that operates on simple user interface definition files.
- DOM Property Editor: Application that demonstrates property editing, with a large variety of property editing controls and property descriptors. The values of the properties are stored in the DOM (Document Object Model).
- File Explorer: Simple Windows Explorer-like application to view the contents of a disk.
- Fsm Editor: Simple finite state machine editor.
- Model Viewer: Model viewer showing how to use ATF classes to load ATGI and Collada models and render them using OpenGL.
- Simple DOM Editor: Editor demonstrating use of the Document Object Model (DOM), including defining a data model.
- Simple DOM No XML Editor: Similar to Simple DOM Editor, but does not use XML.
- State Chart Editor: Simple editor for statecharts.
- Target Manager: Using the `TargetEnumerationService` to discover, add, configure, and select targets, which are network endpoints, such as TCP/IP addresses, PS3 DevKits or Vita DevKits.
- Timeline Editor: Relatively full-featured timeline editor.
- Tree List Editor: Editor to create and add entries to various kinds of tree lists, including a hierarchical list that displays a selected folder's underlying folders and files.
- Using Direct2D: Drawing with Direct2D and ATF classes that support Direct2D.
- Using Dom: Demonstration of basic DOM use.
- Win Forms App: Basic Windows Forms (WinForms) application.
- Wpf App: Basic WPF application.

Components in Samples

The samples create components of various sorts that demonstrate how to create and use components:

- `DomTypes`: Use the `DomNodeType`, `ChildInfo`, and `AttributeInfo` DOM metadata classes to describe an application's Document Object Model (DOM). Normally these are defined by a schema file. This component is in the Simple DOM No XML Editor sample.
- `Editor`: Create and save event sequence documents; it also registers the event sequence `ListView` controls with the hosting service. Circuit Editor.

- `EventListEditor`: Track event sequence contexts and also handle drag and drop and right-click context menus for the `ListView` controls that display event sequences. Simple DOM Editor.
- `GroupingCommands`: Define circuit-specific commands for group and ungroup. Grouping takes modules and the connections between them and turns them into a single element that is equivalent. Circuit Editor.
- `LayeringCommands`: Component to add an "Add Layer" command to an application. Command is accessible only by right-click (context menu). Circuit Editor.
- `MasteringCommands`: Component that defines circuit-specific commands for "Master" and "Unmaster" commands. Mastering creates a new type of circuit element from a subcircuit. Any changes to this master element affect all its instances. Circuit Editor.
- `ModelViewer`: View a 3D model. Model Viewer.
- `ModulePlugin`: Add module types to the editor. This class adds sample audio modules. Circuit Editor.
- `PaletteClient`: Populate a palette with types, such as basic Finite State Machine (FSM) types. Diagram Editor, DOM Tree Editor, Fsm Editor, Simple DOM Editor, Simple DOM No XML Editor, and State Chart Editor.
- `RenderView`: Render a 3D scene from the active document. Model Viewer.
- `ResourceListEditor`: Register a "slave" `ListView` control to display and edit resources that belong to the most recently selected event. It handles drag and drop and right-click context menus for the `ListView` control. Simple DOM Editor and Simple DOM No XML Editor.
- `SchemaLoader`: Load the user interface schema, register data extensions on the DOM types, and annotate the types with display information and `PropertyDescriptor`s. Circuit Editor, Diagram Editor, DOM Tree Editor, Fsm Editor, Simple DOM Editor, State Chart Editor, Timeline Editor, Win Forms App, and Wpf App.
- `SourceControlContext`: Implement a context for source control. Code Editor.
- `TreeLister`: Display a tree view of user interface data. Use the context registry to track the active UI data as documents are opened and closed. DOM Tree Editor.

Building Samples

Pre-built sample executables are in the `bin\wws_atf\Release.vs2010` folder.

To build samples, you need either Visual Studio 2012 or 2010.

You can build samples with two different solutions:

- `Samples.vs2010.sln`, in `components\wws_atf\Samples\`.
- `Everything.vs2010.sln`, in `components\wws_atf\Test\`. "Everything" includes Legacy ATF (also known as ATF 2.8) and unit tests and the `DomGen` development tool. `DomGen` creates static C# classes that contain `DomNodeType`, `AttributeInfo`, and `ChildInfo` metadata objects that make accessing DOM node data fast and less error-prone, by avoiding code that uses string literals from a schema file.

These solutions include all ATF source code, as well as the samples' code.

Each sample has its own project in the solution. The Win Forms App and Wpf App samples are under the `WinGui` folder in Visual Studio's **Solution Explorer** and share common code.

7 Building an Application Based on an Existing Sample

The simplest way to create an ATF-based application is to modify one of the existing samples.

Some suggested steps:

(1) Pick the sample that most closely matches your new application's features. Look at the ATF Code Samples and their descriptions and the ATF Technology and Sample App Matrix on the ATF wiki to help you pick the most suitable sample.

(2) Copy the sample project and the `Samples.sln` file. Open the solution and eliminate the projects you don't need. Make any other changes you need, such as fixing up references to the `\Framework` projects.

(3) Determine which components you want to use. Add or remove components to or from the MEF `TypeCatalog` in `Program.cs`. For information on MEF, see "Managed Extensibility Framework (MEF)".

(4) Remove code you don't need by removing files from the solution or code in files. You can also do this later.

(5) Add code from other samples that you want for your application. You can also do this later.

(6) If you are going to use a DOM for your application data, edit the schema file, if there is one. Consider what types of data your application needs and change the schema to incorporate these types. For DOM information, see "Document Object Model (DOM) Framework" and the *ATF Programmer's Guide: Document Object Model (DOM)*.

(7) Build the application, making needed changes until it builds.

(8) Try to run the application, fixing problems until it runs.

(9) Add code from other samples. Remove code you don't need.

(10) Repeat steps 7 to 9 until the application builds and runs again.

(11) Test whatever features the application currently has and get these working.

(12) Add the application's unique features, using the appropriate ATF classes and components, starting with the most important features. For a guide on where to look for which ATF classes and components to use, see the "ATF structure" chapter, the "Functional Areas" of Table 3 in particular.

(13) The best way to add some features may entail creating components of your own. Look at the samples' examples of components, such as those described in "Components in Samples". You may want to copy some of these components and adapt them to your application. Connect other code into your new component, as needed.

(14) Make the added features work.

(15) Continue development and debugging in steps 12 to 14 until all the application's features work.

8 Next Steps

This section suggests steps to take in learning more about ATF.

- Study the *ATF Programmer's Guide* to learn the basics of programming with ATF.
- If you are going to use a DOM, study the *ATF Programmer's Guide: Document Object Model (DOM)* to learn how to use ATF's DOM.
- If there are ATF terms unfamiliar to you, look them up in the *ATF Glossary*.
- Read the ATF Programming Guidelines to learn the programming and code style guidelines followed by the Authoring Tools Framework team. Even though you are not developing ATF itself, these are useful development guidelines. It is especially useful to follow these guidelines in components you write, because these are essentially extending ATF and .NET.
- Look at the ATF Code Samples and their descriptions and the ATF Technology and Sample App Matrix on the ATF wiki to see what the samples offer.
- Build samples of interest and run them, as described in "Building Samples". Examine sample source, starting with the `Program.cs` file. In the `Main()` function, note how components are added to the MEF `TypeCatalog`. Take a look at some of the components' code. Read other code in the samples.
- Read appropriate selections from the publications listed in "Other Reading" to deepen your understanding of framework development and C#.