POSReady: POS for .NET Overview

By Gordon H. Smith, Embedded MVP

Microsoft® Point of Service (POS) for .NET v1.12 is a class library that enables POS developers to apply Microsoft .NET technologies in their products. POS for .NET provides a straightforward and consistent interface and classes that enable .NET applications to interact with POS peripheral devices, such as line displays or receipt printers. POS for .NET is part of the unified POS industry standard for writing applications and POS drivers that interface with POS peripherals devices. Going beyond the standard features, it includes support for Windows® Plug-and-Play functionality, device simulators, as well as Windows Management Instrumentation (WMI).

Prior to the introduction of POS for .NET, retail-oriented devices were implemented as OPOS (OLE for Point of Sale) controls. The OPOS model had two deliverables for each hardware device. One was a control object that was the interface delivered as a COM ActiveX object that the application used to control the device. The other was a service object that provided a device-specific interface to the hardware device and communicated with the control object. Collectively, these are shown in Figure 1 as Legacy SOs. This style of service object was often difficult to write and did not often benefit from Plug-and-Play support, which made administration of such service objects challenging. The POS for .NET environment incorporates support for these legacy service objects via the Legacy Interop System.

Figure 1 illustrates how POS for .NET uses service objects and operating system infrastructure to provide programmatic access of POS devices to POS applications. A brief discussion of the various components follows.

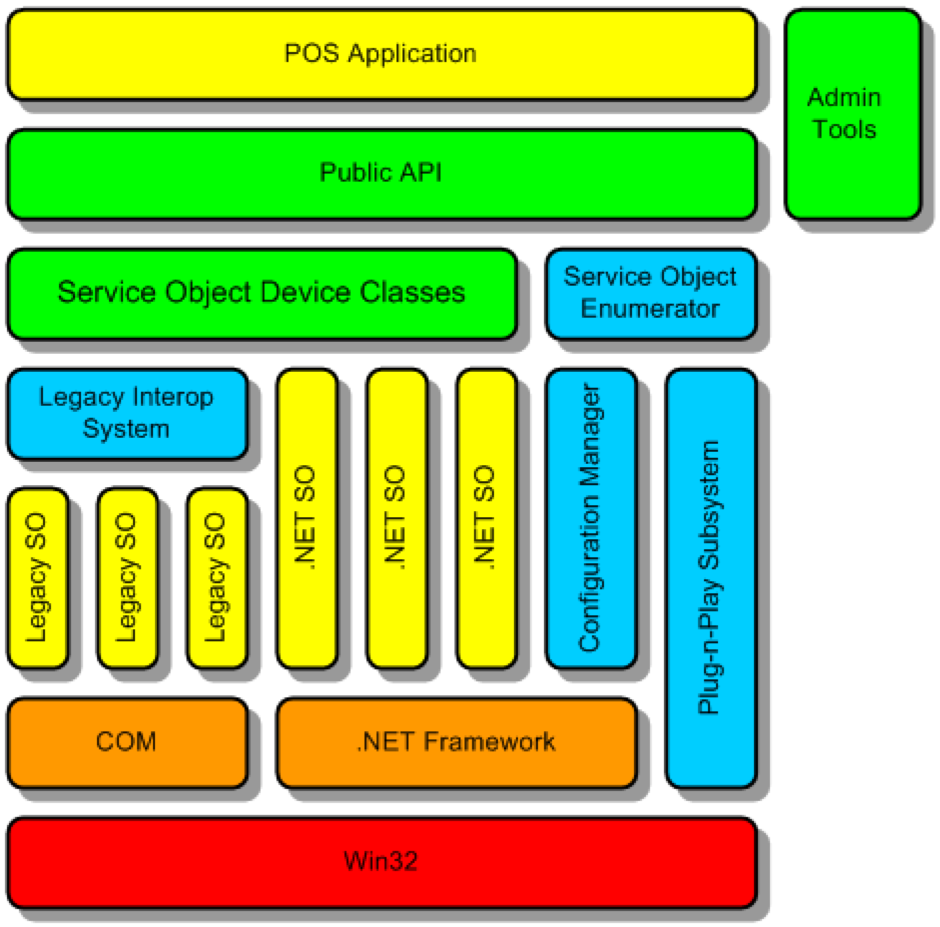

Figure 1. POS for .NET architecture

In POS for .NET, the POS for .NET PosExplorer object replaces the OPOS control objects, eliminating a typical service object/control object mismatch issue in OPOS. A POS for .NET service object, shown in Figure 1 as .NET SO, handles the control of the device and implements interfaces as specified in the Unified Point of Service (UPOS) standard. These POS for .NET service objects provide the appropriate hooks for retail application developers to read and write to the device as well as react to the addition and removal of devices at runtime. The PosExplorer object, shown in Figure 1 as the service object enumerator, gives applications the ability to discover supported POS devices at runtime, which greatly eases the configuration of systems.

POS for .NET supports service object device classes for all thirty-six UPOS device categories. These classes provide a common implementation of much of the logical interaction with POS devices. POS for .NET supports Interface classes for all UPOS devices categories. The Interface classes provide entry points as specified in the UPOS specification, but with minimal functionality. Inheriting from each Interface class is a Basic class. Each Basic class provides support for opening, claiming, and enabling the device, as well as device statistics, and the management of delivering events to the application. Additionally, each Basic class contains a set of inherited and protected methods that the service object can implement. Additionally, fully functional Base classes are provided for a number of device categories that expand their corresponding Basic class with device-specific members provide more specific support for the nine primary UPOS device types. For further details, see http://msdn.microsoft.com/en-us/library/bb411798(WinEmbedded.11).aspx.

The fundamental interface to the operating system is the Win32® API, which is a common interface to all Windows operating systems. Unmanaged (or native) code uses this layer to control Windows resources such as files, graphics, networking, threads, peripheral input/output (I/O), and the registry. Application developers who leverage POS for .NET are isolated from this layer and do not need to consider it in their designs.

Building on top of the Win32 API are Component Object Model (COM) interfaces. The COM model is a legacy programming model that enables interprocess communication and dynamic object creation. You can reuse created COM objects without knowing about their implementation because COM provides a well-defined interface separate from the implementation. The .NET Framework common language runtime is also built on top of the Win32 API. The .NET Framework represents a programming model and provides far more capability to developers than COM, such as memory management, strict type safety, and thread management. To learn more about the .NET Framework, see http://msdn.microsoft.com/en-us/netframework/default.aspx. Again, Application developers who leverage POS for .NET are isolated from this layer.

Built into Windows operating systems is the Plug-and-Play subsystem, whose purpose is to instantiate support for hardware devices at runtime. This subsystem provides the benefit of not having to explicitly configure and load device drivers in case of hardware additions or subtractions. When a bus identifies an unknown device, the Plug-and-Play subsystem searches for compatible device drivers and loads them automatically. Application developers who use POS for .NET are able to be notified of Plug-and-Play events by simply subscribing to the event from the POSExplorer class.

POS for .NET exposes a public API that you can use to create instances of the various service objects and enable the POS Application to control devices. Additionally, the API enables retail developers to build more resilient applications that are capable of brand-agnostic hardware support, and that react intelligently when peripherals are added or removed at runtime.