Chapter 18

# Developing Solutions for Intel® Active Management Technology

*Goto, n.: A programming tool that exists to allow structured programmers to complain about unstructured programmers.*

—Ray Simard

In this chapter, we cover the development and architecture decisions involved in building Intel® vPro™ technology solutions. Individuals, software vendors, and organizations can all decide to build their very own software that makes use of Intel Active Management Technology (Intel AMT), or add Intel AMT support onto existing software. In all cases, there are a set of common decisions that can have a large impact the cost, feature set and level of integration or the final product.

As with all technologies, developers face a learning curve to get up to speed and ready to build new software, so we want look at all of the options first, before starting work on software development.

## Complete Reuse

For many people just getting started with Intel vPro technology, having deployed just a few computers so far and considering adding Intel vPro support to an existing software, probably the easiest way to go is to take in

existing software practically as-is. Intel provides the open source Manageability Developer Tool Kit (DTK), which includes many sample tools that work with Intel vPro technology. Two of the tools in the DTK are especially made for integration into existing software. These are the Manageability Commander Tool and the Manageability Terminal Tool. Both of these can be invoked from the command line to manage a targeted computer:

```
"Manageability Commander Tool.exe" -h:<hostname>
-u:<username> -p:<passwork> [-tls]
```

This command line will invoke the Commander tool to manage a single computer with Intel vPro technology. The user interface is simplified a little because only one computer is managed. The user can't, for example, add or remove a computer. It is pretty easy to add an option to launch the Commander tool for another software package, and when doing it this way, the list of managed computers and credentials is not kept in the Commander tool, but rather by the calling software. As a simple first solution, existing management software should add the option to invoke the Commander tool when the user selects a computer with Intel vPro technology.

All of the software in the Manageability DTK is provided as open source. As a result, developers can freely change or re-brand the code to best suit their needs.

The Commander tool can deal with most of the Intel AMT features including support for Serial-over-LAN and IDE-Redirect. Invoking directly the Commander tool is by far the fastest way to add Intel vPro support to existing software. It is limited to running on Microsoft Windows using the .NET framework, but otherwise performs rather well.

## Supporting Serial-over-LAN

Another tool in the DTK with the approximately the same launch options as the Commander tool is the Manageability Terminal.

```
"ManageabilityTerminalTool.exe" -h:<hostname>
-u:<username> -p:<password> -t:(title)
```

The Terminal tool can be launched to target a computer. The calling software can also optionally specify the title of the window that is displayed on top of the terminal. This way to launch the terminal tool also allows the terminal to support remote control and IDE-Redirect. This is by far the easiest way to add Serial-over-LAN support into an existing application.

### Selecting a Terminal

There are other approaches to supporting serial-over-LAN. Some vendors have opted to use existing terminals such as Putty, Microsoft Telnet, or Microsoft Hyperterm, which is included with Microsoft Windows XP and optionally available on Microsoft Vista as a supplemental download. In order to do this in the past, software solutions would use the IMRSDK.dll to connect to a managed computer and forward the traffic back and forth to a local TCP socket. As a result we have the traffic flow shown in Figure 18.1.

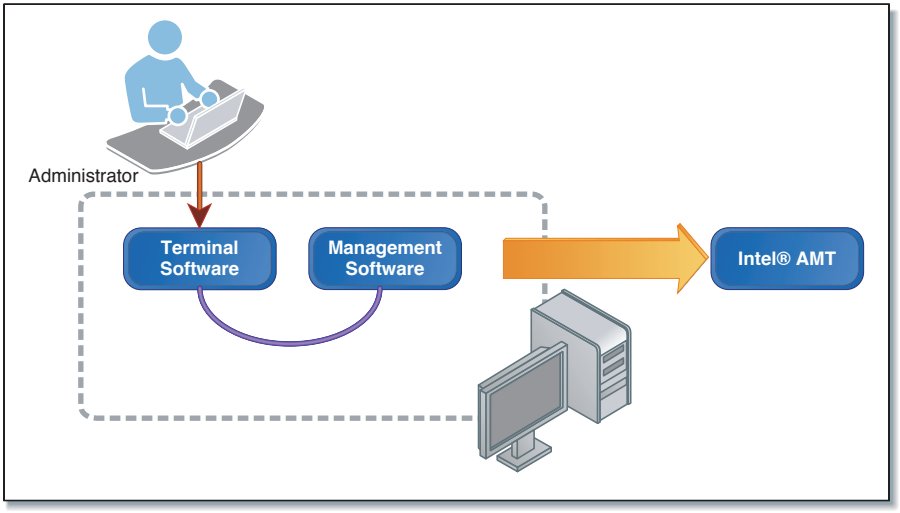


**Figure 18.1** Management Software Traffic Flow When Using Separate Terminal Software

At this point, any VT100 terminal software could be used as long as it's capable of connecting to a local TCP port. One example is to invoke the Microsoft Windows Telnet tool like this:

```
Telnet.exe localhost 12345
```

Where `12345` is the local port on which the management software is listening. Even the Manageability Terminal Tool included in the DTK can be used in this role since it can also be invoked just like the Microsoft Telnet tool.

```
"ManageabilityTerminalTool.exe" <hostname> <port> (window
title)
```

Developers will have to build the IMRSDK.dll to TCP replay on their own and the Intel AMT SDK can help. Once completed, this solution as the benefit of running on Linux. Developers must be advised that just any terminal will not do; terminals should support 25 display lines. Many terminals intended for modem usages support only 24 lines. Also, telnet applications don't make good VT100 terminals; they are somewhat compatible with VT100, but it's not perfect. Putty for example supports both Telnet and RAW modes, and the RAW mode would be preferable.

It's also important to note that the IMRSDK library is only provided in binary form with no source code for Microsoft Windows and Linux, in both 32-bit and 64-bit versions. Developers must make sure that this library can run on their platform before starting to build a terminal solution.

In general, redirecting traffic to an existing terminal is not really a great solution, it's probably best to use the DTK terminal if possible because it's custom built for Intel vPro. Also, the DTK's terminal will support F1 to F12 keys for various BIOS, and so on. So it's generally more user friendly.

Another option is to use the terminal control that is in the DTK. This Microsoft .NET control can be dropped into an existing .NET application, giving great compatibility while letting developers have flexibility about how it can be best integrated into an existing application. The terminal control can be added to an existing form, making the resulting application more integrated.

If a developer opts to build his own terminal from the ground up, it's generally recommended to take a look at the DTK's terminal as a starting point. Developers can learn from it and use it to avoid many time consuming mistakes.

## Selecting a Software Stack

On a practical level, developers will have to select a software stack for their Intel vPro development or build their own. Figure 18.2 shows three commonly used Intel AMT source bases.

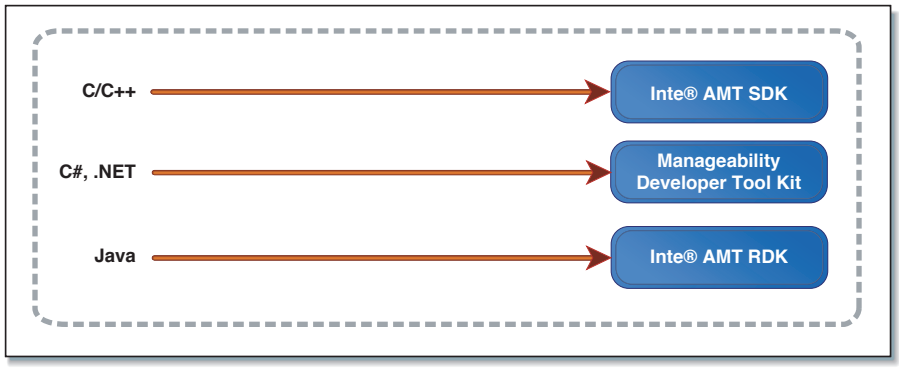


**Figure 18.2** Available Development Stacks for Various Programming Languages

Figure 18.2 is a bit of an oversimplification. All developers, regardless of the programming language they use, should get to know the Intel AMT SDK since it's the official reference for everything else. Both the Manageability Tool Kit and the Intel AMT RDK where built with the Intel AMT SDK as the starting point.

In general, if the target software is Microsoft .NET–based, developers should look at the DTK's Manageability Stack.dll. It includes wrapper classes for practically all of the Intel AMT features with many DTK tools service as samples on top of this stack.

For C/C++ software development, developers will likely have to start from the Intel AMT SDK. This SDK provides detailed documentation but all samples are very low level and so, much work has to be done to use all of the features. Still, it's the basis for all other Intel vPro software and is the official software development kit for Intel AMT in addition to being the most up-to-date with latest platform features.

For Java development, Intel provides the Reference Developer Kit (RDK). This Java reference code is not maintained as much as the SDK and DTK, but it still serves as a good start for Java developers. Even if the RDK is Java, it won't run on all platforms since it does make native calls to IMRSDK for Serial-over-LAN and IDE-Redirect support.

### Selecting a WS-Management Stack

Developers using the Manageability Developer Tool Kit will find that it includes its own C# built WS-Management stack that was custom made to work with Intel AMT. For developers not using the DTK's code, there are two other well known solutions for supporting WS-Management. Figure 18.3 shows the WS-Management stacks that are commonly available.

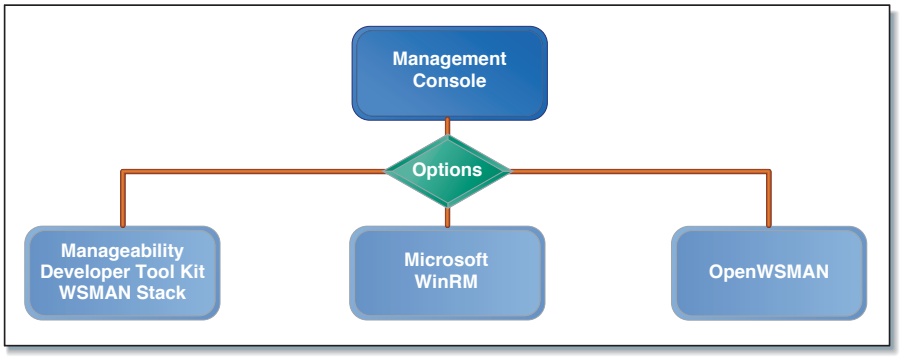


**Figure 18.3** Management Consoles Have a Choice of Three WS-Management Stacks

In the Manageability Developer Tool Kit, the WS-Management stack is fully written in C# and works on top of the HTTP client that is built into .NET. Since it's built with Intel vPro in mind, it works very well, but does not claim to be a general purpose WS-Management stack. It is also built to be fast and work correctly with the Intel vPro Enabled Gateway. For developers using .NET, this is clearly the WS-Management stack that is recommended.

Starting with Microsoft Vista, Microsoft includes a WS-Management stack with the operating system. Microsoft WinRM can also be installed as a freely available package on Microsoft.com. All of the Intel AMT SDK samples based on WS-Management use WinRM and so, the Intel AMT SDK

provides plenty of sample code for using WinRM. Even the C# samples in the SDK use WinRM, and so work differently from the C# code in the DTK. Microsoft WinRM is widely used but has a few major disadvantages:

Microsoft WinRM must be configured in advance. When using Microsoft WinRM, the user must first configure and start WinRM. Instructions on how to do this are included in the Intel AMT SDK.

Microsoft Windows XP users must download and install an extra package. If not automated, this can be a time-consuming process.

WinRM is also slow. Because each WS-Management call required a new HTTP connection, making many consecutive WS-Management calls can be rather slow. OpenWSMAN and the DTK WS-Management stack don't have this problem and users will notice a significant improvement in call performance.

WinRM can't ignore un-trusted certificates. It is sometimes useful to connect to a TLS enabled Intel AMT computer even if the certificate on the Intel AMT computer is not trusted by the console. For example, this is practical if the certificate must be renewed.

Lastly, this stack may not work well with Intel vPro Enabled Gateway. Because the developer can't setup a different proxy for each HTTP session, instead it must be set system-wide, possibly disrupting other applications on the same computer.

For all these reasons, even if WinRM is heavily used with all of the Intel AMT SDK samples, it is a stack that should be avoided when possible.

Lastly, OpenWSMAN is likely the best solution for C/C++ developers. It's available at no cost, it's well supported, and it works well with Intel AMT.

Other WS-Management stacks are available for JAVA and other languages. In the case of Java, the Intel AMT RDK does not support WS-Management, only SOAP and so does not yet use a WS-Management stack of its own.

## Using the WS-Management Translator

Since Intel AMT started out using SOAP and is moving over to WS-Management, new manageability solutions should focus on providing excellent support through WS-Management. Still, Intel AMT enabled computers before version 3.0 do not have support for WS-Management and to help with backward compatibility, Intel provides a freely available WS-Management translator that allows WS-Management–only solutions to communicate with SOAP-only Intel AMT computers.

Except for developers using the Management Developer Tool Kit, which supports both WS-Management and SOAP in the same stack, all other developers should consider focusing on WS-Management support first, with native SOAP support second or using the WS-Management translator for legacy support.

## Using the Manageability DTK Stack

The Manageability Developer Tool Kit (DTK) is more than a set of reference tools; it also includes a usable Microsoft .NET stack built in C#. The DTK stack has many benefits such as automatic detection of TLS and WS-Management and support for all versions of Intel AMT, all the way back to Intel AMT 1.0.

The DTK includes two major DLLs that most of the DTK tools make use of:

**Manageability Stack.dll** Includes all of the .NET classes needed to connect to and manage an Intel AMT computer. The stack supports all of the features demonstrated by all the tools and supports remote and local (LMS) connection, TLS support, WS-Management support and much more. This stack contains only one user visible form for debugging. Any application built on top of this stack can cause the stack to show this debug form, making it easier to see what the stack is doing.

**Manageability Controls.dll** In the DTK, all common Intel AMT forms are located in this controls DLL. This includes the VT100 terminal, and common forms for editing certificates, circuit breakers, and much more. Because most of these controls have a look and feel that is unique to the DTK, many developers may opt to use the DTK stack as-is, but change

the forms from the Manageability Controls.dll to best match their own application. Probably by far the most popular control in this DLL is the VT100 terminal, one of the only terminals custom built specifically for Intel AMT serial-over-LAN.

A really good way to get started with the DTK stack is to take a look at the ManageabilityCmd.exe sample that is available as part of the DTK's source code. The code sample is about two pages long and demonstrates the basics of how to use the stack to connect to an Intel AMT computer and perform management commands on it.

```
// In the main method
AmtSystem computer =
  new AmtSystem(hostname, 16992, username, password,
false, true);
computer.AutoFetchCache = false;
computer.OnStateChanged +=
  new AmtSystem.ObjectStateHandler(SystemStateChangedHandl
er);
computer.Connect();


// Event sing method
private static void SystemStateChangedHandler(AmtSystem
computer)
{
    switch (computer.State)
    {
        case AmtSystemObjState.Connecting:
            Console.Out.WriteLine("Connecting.");
            break;
        case AmtSystemObjState.Disconnected:
            Console.Out.WriteLine("Disconnected.");
            break;
        case AmtSystemObjState.Connected:
            Console.Out.WriteLine("Connected.");
            break;
    }
}
```

In this code sample, an object of class `AmtSystem` is created and setup for managing a target hostname with a given username and password. The specified port "16992" indicates that this computer is probably not setup with TLS, but the stack may automatically change the port to 16993 if TLS is detected.

The `AutoFetchCache` property is set to false so that the stack does not start to load up all of the settings of the computer upon connection. If set to true, the stack will pre-load all of the computer's settings making subsequent calls to get Intel AMT parameters much faster. This caching feature was added to the DTK stack to speed up user interfaces.

Finally, the Connect and Disconnect methods can be called on the `AmtSystem` object to connect and disconnect from the Intel AMT computer. Since calls are made over HTTP, the stack's concept of connection is really made up, at any given time the stack many not be truly connected to the computer, but the concept of connection was created for simplicity. Both Connect() and Disconnect() methods on the stack are non-blocking and will return immediately. An application must catch the state change event to get notified of the connection.

At any time, the DTK stack will update the connection state from "disconnected" to "connecting" to "connected" and back to "disconnected" by updating the `State` property and firing the `OnStateChanged` event. Figure 18.4 shows the possible transitions between these states. Only when an object is in "Connected" state can management operations be called.

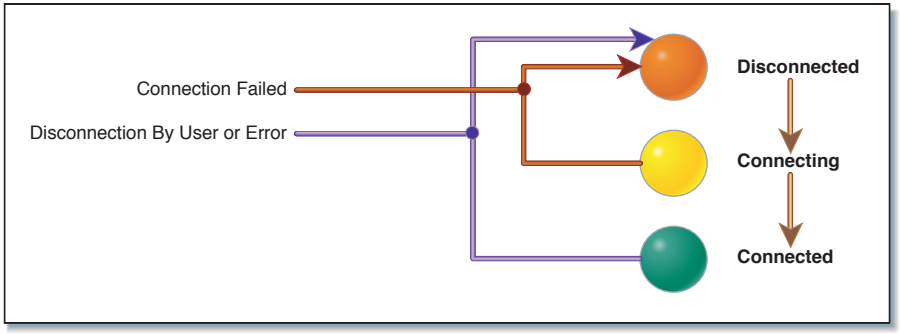


**Figure 18.4** Connection State of the Manageability DTK Stack

Using the connection and state system of the DTK stack, an application can easily keep track of the state of each connection. It's generally recommended in this model to call Connect() or Disconnect() and only update the user interface once the state of the object as changed and the event is fired. This way, a user interface only needs to update at a single place in the code.

### Manageability Stack Services

Once the DTK stack is in connected state, it's time to perform management operations. To do this, the stack offers a set of sub-objects, each representing a set of features, as shown in Table 18.1.

**Table 18.1** DTK C# Sub-objects for Performing Management Operations when the DTK Stack Is in the Connected State

| | |
|---|---|
| Info | SecurityAdmin |
| Remote | CircuitBreaker |
| Network | Wireless |
| Assets | NetworkAccessControl |
| Events | NetworkAccessControlAdmin |
| Redirection | RemoteAccessAdmin |
| Storage | NetworkTime |
| Watchdog | AuditLog |
| WatchdogLocal | |

Once connected, a developer can simple use any of these services by calling methods like this:

```
computer.Assets.GetBios();
computer.CircuitBreaker.GetCircuitBreakerFilters();
computer.Redirection.GetIderSessionLog();
computer.Storage.GetStorageAttributes();
computer.Wireless.GetWirelessCapabilities();
computer.Watchdog.GetAgents();
```

These are only some of the many methods that can be called on services. There is one important point to remember: if a service is not available on a given computer, the service object will be null. For example on Intel AMT 1.0, there is no support for Circuit Breakers (Intel® System Defense) and so the "CircuitBreaker" services will be null. As a precaution, a developer can perform a test:

```
if (computer.CircuitBreaker != null)
{
    // Supported
    computer.CircuitBreaker.GetCircuitBreakerFilters();
}
else
{
    // Not supported
}
```

Another good example of this test is when the DTK stack is connected locally via LMS, in this case, most of the services are not available but some services are uniquely available when connected locally.

```
if (computer.Watchdog != null)
{
    // Remote connection
    computer.Watchdog.GetAgents();
}
if (computer.WatchdogLocal != null)
{
    // Local connection
    computer.WatchdogLocal.GetWatchdogs();
}
```

In this example, the stack can be connected locally or remotely and in both cases, one of the two services will usually be accessible. Speaking of connecting the stack locally using LMS, the DTK stack is built to attempt to automatically detect local connections, so this code will work:

```
// In the main method
AmtSystem computer =
  new AmtSystem("localhost", 16992, username, password,
false, true);
computer.LocalConnection = true;
computer.Connect();
```

The "localhost" string will cause the stack to connect to the local LMS and will automatically detect that only local services are present and functional. We can optionally set the LocalConnection property to true ahead of invoking the Connect() method, this will give a hint to the stack that a local connection is expected and accelerate the connection process. We can check local connectivity after connection with the following code:

```
if (computer.LocalConnection == true)
{
    // Local LMS connection
}
else
{
    // Remote network connection
}
```

There is one other case where giving a hint to the stack prior to connection can significantly accelerate the speed of connection. This is when we expect to connect to the TLS enabled Intel AMT computer.

```
// In the main method
AmtSystem computer =
  new AmtSystem(hostname, 16993, username, password,
false, true);
computer.UseTls = true;
computer.Connect();
```

In this case, both the port and the UseTls property are used to give hints to the stack that a TLS connection is expected. After connection is established, both Port and UseTls properties can be read again to see what value was actually used. If the TLS setting is not correct, the stack will automatically try again but there is usually a 5- to 10-second penalty for the detection process to occur.

Developers building their own Intel AMT stack may be interested in the process of auto-detecting Intel AMT Version, TLS, LMS and WS-Management. There is no formal way to do this and Intel AMT was certainly not designed to make the process easy. The "TryConnect()" method in AmtSystem.cs performs this nearly magical task. Care was taken for this process to work even as early back as Intel AMT 1.0 and so the methodology will seem very odd.

Once connected, the version of Intel AMT is located in the `CoreVersion` property of the `AmtSystem` object. This value is really the first piece of data the stack attempts to retrieve from Intel AMT after which it will adapt itself to work correctly.

### Certificate Operations

Developers will notice quickly that it is important to be able to operate on certificates in order to provision or use many of the features of Intel AMT. When building the DTK, it became important to create, sign, validate, and conduct other certificate operations. The DTK stack provides a class called `CertificateOperations` that contains only static methods. These methods use the OpenSSL executable and some built-in Microsoft .NET methods to perform these operations.

With Microsoft .NET 2.0, there is not built-in support for a certificate authority. In addition, with Intel AMT 1.0 and 2.0, pushing a certificate into Intel AMT involved a bit of magic. While it's beyond the scope of this book to go into details, developers should look at the `CertificateOperations` class for insight into how the DTK handles certificates.

The DTK stack also makes heavy use of the Microsoft Windows personal certificate store. If Intel AMT is set up to use mutual-authentication, the DTK stack will automatically try each certificate with the correct usage flags in the personal certificate store in trying to connect to an Intel AMT computer.

### Kerberos Support

Starting with Intel AMT 2.0, Intel AMT has support for Kerberos user authentication. Some provisioning tools support it and the DTK stack is capable of logging into an Intel AMT computer with Kerberos. This feature is somewhat hidden. To log using the current account, just leave the username and password blank:

```
// In the main method
AmtSystem computer =
  new AmtSystem(hostname, 16993, "", "", false, true);
computer.UseTls = true;
computer.Connect();
```

The stack also supports using any Kerberos username and password, simply add a "\" in the username with the format "domain\username" like:

```
// In the main method
AmtSystem computer =
  new AmtSystem(hostname, 16993, "domain\user", "pass",
false, true);
computer.UseTls = true;
computer.Connect();
```

The presence of the "\" in the username field will automatically cause the DTK stack to use Kerberos authentication. One word of warning about using Kerberos and Serial-over-LAN and IDE Redirect: because of a limitation in the IMRSDK.dll which does not allow the Kerberos credentials to be passed into the library, Kerberos will only work with the locally logged-in user. Leaving both username and password blank will make Serial-over-LAN and IDE-Redirect work since both the stack and IMRSDK will use the local user account.

## Summary

Before starting a new software project, or before adding Intel vPro support to an existing project, important decisions must be taken. Considering the programming language, type of solution and cost, many options are available. This chapter covered much of the high level information needed to make proper design decisions.

We also covered the basics of using the Manageability Developer Tool Kit stack, a community-supported open source stack that is available freely to developers.