



# Intel<sup>®</sup> Technology Journal

Interoperable Home Infrastructure

**Remote I/O:  
Freeing the Experience From the  
Platform with UPnP\* Architecture**

# Remote I/O: Freeing the Experience from the Platform with UPnP\* Architecture

Mark R. Walker, Desktop Platform Group, Intel Corporation  
Jim Edwards, Desktop Platform Group, Intel Corporation  
Michael Jeronimo, Desktop Platform Group, Intel Corporation.  
John G. Ritchie, Desktop Platform Group, Intel Corporation  
Ylian Saint-Hilaire, Desktop Platform Group, Intel Corporation

Index words: UPnP technology, remote, interface, discovery, protocol

## ABSTRACT

Increasing adoption rates of home networking solutions raises the possibility of using the Personal Computer (PC) to remotely extend its power and capabilities into all areas of the home. A new specification to be developed under the auspices of the Universal Plug and Play (UPnP\*) Forum will allow home applications to be controlled and experienced with remote, network-connected devices. When adopted, this standard will enable the development of whole new classes of software applications that power new user experiences within the digital home. In this paper, we present two usage scenarios that illustrate new capabilities that would be enabled with this proposed standard. We discuss the foundational architecture of UPnP technology for home networking solutions and highlight its efficacy as a basis for constructing new, remote Input/Output (I/O) software services standards. Finally, we describe the technical challenges of the proposed standard and contrast this standard with existing and comparatively incomplete schemes for remote desktop and similar functionality.

## INTRODUCTION

Recent reductions in the installation cost and advances in the average bandwidth of wired and wireless networking solutions have raised the possibility of employing the home network to remotely extend the considerable processing power of a fixed-location home PC into all regions of the home. As a result, a new standardization effort recently approved by the Universal Plug and Play (UPnP\*) Forum [1] will focus on developing new software services for remote I/O across the home network.

Remote I/O is a technology under development based on the UPnP device architecture. It moves the point of user interaction away from an application running on a specific device, such as a PC or a CE (Consumer Electronic) device, to one or more remote I/O devices. The remote I/O device supplies input and output services such as mouse, keyboard, and display that together comprise the user interface. Applications run on a host elsewhere on the home network and are matched with compatible I/O devices. Applications may take on different user interface characteristics depending on the I/O devices being used. Furthermore, the application user interface can migrate across I/O devices as the user moves about the home.

Remote I/O supports interactions between PC and CE devices by introducing location independence and tailored I/O devices to the home network. Applications can connect to wireless I/O devices anywhere in the home, freeing the user interaction from the location of the application. For example, a user might prefer to read the news in a comfortable chair in the family room instead of using the desktop PC in the den. In addition, I/O devices can be tailored to user activity. For example, a display for reading the news should be handheld, comfortable to hold, and allow the user to adjust visual settings such as colors and contrast. These two properties of remote I/O devices will improve the quality of user experiences in the digital home.

When in place, this proposed standard will effectively establish a UPnP remote I/O framework, allowing a fixed-location PC to serve user interfaces to a multitude of wired and wireless remote devices in the home. Under this new framework, remote User Interface (UI) devices will possess the “out-of-the-box” ability to discover and execute applications that can be run remotely. Correspondingly, these applications running on the PC would possess the out-of-the-box ability to discover remote UI devices and

---

\* All other brands and names are the property of their respective owners.

project user interfaces onto them. By using this new middleware standard to drive remote UI devices, the PC-application user experience will be “freed” from the home PC platform.

## User Scenarios

The following scenarios illustrate specific technical concepts of the proposed UPnP remote I/O architecture. It is important to note that there are many other compelling scenarios, especially around entertainment and communications, which could also benefit from remote I/O.

### Scenario 1: Remote Control of Home Automation Functions

This is an illustration of the remote I/O standard-endowed ability of remote devices to discover and execute remote-enabled home applications.

Dave and Kathy’s house has a home automation system. The system consists of some application software running on the home PC and a wire termination panel installed in the pantry. The termination panel is in turn connected to a large collection of interior light switches and dimmers, the thermostat, switches for outdoor lighting, the security system, and the lawn watering system. The PC implements the UPnP remote I/O standard on the home network. The termination panel is also connected to the home network, but communicates only with the PC and is not a UPnP device.

Kathy picks up the newly purchased universal wireless home remote from the kitchen table and sees an icon corresponding to the home-automation system in the activity display that she hadn’t noticed previously. Touching the corresponding softkey, Kathy is surprised to see icons corresponding to all of the home-automation subsystems in the display. She selects the interior lighting system and watches the display change into a group of icons representing switches and dimmers for the entire house. She touches the on/off softkey corresponding to the light switch for the den and hears muffled, annoyed shouting coming from the den at the other end of the house. Dave is apparently still in the den using the PC.

### Scenario 2: Integration of Remote User Interface Devices with a Home Security System

This is an illustration of the remote I/O standard-endowed ability of remote-enabled applications to discover and project user interfaces onto remote devices.

Continuing the previous scenario, Dave and Kathy’s home-automation system is also connected to their security system. Dave configured the security system to use his wireless PDA and the connected TVs in the house

as alarm enunciator and response devices, in addition to the alarm bell connected directly to the wire termination panel. The PDA and the TVs implement the UPnP remote I/O standard.

Late one evening, the family is suddenly awakened by the alarm. Dave and Kathy jump out of bed and immediately direct their attention to the TV. The TV displays text informing them that there is a smoke alarm active (in the garage), the best evacuation route from the master bedroom (down the front staircase and out the front door), and where to meet (near the oak tree in the front yard). As they dash from the room, Dave grabs his wireless PDA on the nightstand. The PDA screen displays some of the same text that appeared on the TV, but also has a menu displaying a choice of emergency responses. Dave uses the stylus to touch “Call Fire Department” as he reaches the front door. Their teenage son Rick meets Dave and Kathy on the lawn. The net-connected TV in Rick’s room had instructed him to take the back staircase to the patio door in order to avoid the most likely location of the fire. In less than three minutes, a firefighter team arrives on the scene and determines that the smoke is emanating from the breaker panel in the garage. One firefighter uses a handheld extinguisher on the source of the smoke. The entire incident is over in less than twenty minutes. The damage to the house is minimal, and all family members are safe and sound.

## General Requirements of the Remote I/O Framework

The above two scenarios would be possible only with some type of a preexisting framework for remote I/O. General requirements of the hypothetical framework underlying the above usage scenario descriptions include the following:

1. *Out-of-the-box device and service discovery capability.* The UPnP remote I/O framework must allow applications to discover remote-capable devices and initiate remote UI sessions. Correspondingly, remote devices must also have a way to discover and initiate UI sessions with remote-enabled applications.
2. *Scaling to match device capabilities.* Successfully remoting UIs to very thin devices would be made possible by endowing the application server with the ability, under the UPnP remote I/O framework, to adapt the UI and data transmission scheme according to the capabilities of the targeted remote device.
3. *Simplicity.* For remoting to very thin UI devices, it is important to keep the UPnP remote I/O framework as simple as possible. Ideally, the data transmission

protocol should require very little state on the client other than a frame buffer. The application server should additionally be responsible for maintaining its state independent of the remote device.

## HIGH-LEVEL REMOTE I/O ARCHITECTURE

### UPnP\* 1.0 Architecture Overview

The UPnP Device Architecture V1.0 specification [2] provides a solid foundation for building UPnP remote I/O capabilities into local area networks. UPnP technology effectively establishes a middleware standard for out-of-the-box, cross-vendor discovery, description, control, eventing, and presentation.

UPnP Device Architecture V1.0 defines protocols for communication between *control points* and *devices*. Control points are essentially software applications and are the active components of UPnP architecture. Devices are physical or logical entities, enumerated via simple XML descriptions and containing APIs referred to as *services*. Physical devices may host multiple logical devices, and each device may host multiple services.

Messages are transported over UPnP networks via HTTP over UDP/IP or TCP/IP. The supported message formats are Simple Service Discovery Protocol (SSDP), General Event Notification Architecture (GENA), and Simple Object Access Protocol (SOAP). IP addresses are assigned by a DHCP server, or auto-assigned when no server is present. There are four basic types of UPnP network activity:

1. *Discovery*. When a device is added to the network, the device is allowed to “advertise” its presence on the network using SSDP. When a control point is added, the control point is similarly allowed to generate a multicast search for devices. In either case, the message exchange consists of a brief description of the device that includes the UPnP device type, the device ID, and a URL to the full device description.
2. *Description*. A control point obtains more information about a specific device by retrieving the full description from the URL with HTTP GET. The full description is composed of a device description and a service description. The device and service descriptions are XML documents and are constructed by the device vendor with the aid of the device and

service template schemas. The service description contains details of the hosted API commands, called *actions*, along with parameters, called *arguments*.

3. *Control*. A control point accomplishes device control by invoking actions on the service control URL and by polling for service state variables. UPnP architecture employs the SOAP remote procedure call scheme to deliver control messages and return results. Services keep state tables updated so that control points can obtain meaningful values. When state variables change, events are broadcast over the home IP network to all interested control points.
4. *Eventing*. Control points may receive notification of specific state variable changes by forwarding a subscription message containing a delivery URL to the selected service. The service maintains a list of URLs corresponding to subscribing control points. Events are expressed in XML and forwarded to subscribers via GENA-extended HTTP messages.

### Overview of Proposed Architecture of Remote I/O

Like previous UPnP working-committee-developed standards, the UPnP remote I/O standard specification will build upon the basic architecture behaviors listed above and will specify new devices, services, control actions, and event types necessary to realize the usage scenarios.

Features of the proposed UPnP remote I/O framework needed to expose remote-capable *applications* include the following:

- An application registry service that lists applications that implement remote I/O support would be required. The registry would be enumerated by an UPnP remote I/O application server device and visible to all other UPnP devices on the home network. At least one UPnP application server device would be associated with each physical application server on the network.
- Each application listed in the registry would expose a list of input and output attributes in standard form. Typical output attributes of a single application would include video output, static display output in unformatted text or XHTML form, and display update events. Typical input attributes would include named UI events.

Features of the proposed UPnP remote I/O framework needed to describe capabilities of remote *devices* include the following:

- Device input and output services enumerated by UPnP remote I/O devices and visible to all other devices on the home network would be required. At

---

\* Other brands and names are the property of their respective owners.

least one UPnP remote device would be associated with each physical remote UI device on the network.

- Each UPnP remote I/O device would expose input and output services along with device and service-specific properties. Typical input services would include video frame buffers, as well as text and bitmap display. Device-specific properties of frame buffers enumerated by UPnP remote I/O would include frame size, pixel resolution, and supported data transmission protocols. Typical output services would include events bound to physical device actions like mouse & key clicks.

Other than providing support for named events, the lower-level details of the session-specific data transmission protocol selected for use between the application and the remote UI device are out-of-scope with respect to the UPnP remote I/O framework. While UPnP remote I/O provides a framework for enumerating vendor-preferred protocols that may be supported by a given remote UI device, it will not require specific data transmission schemes for standard compliance, nor will it attempt to define new ones.

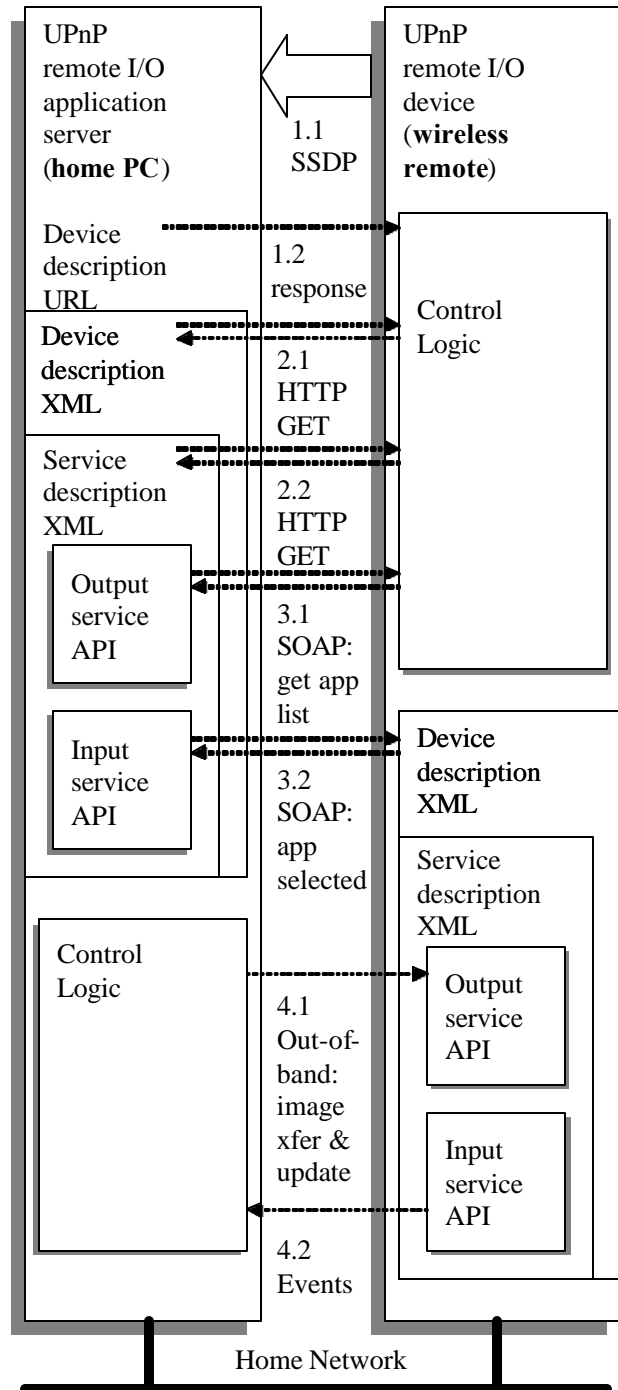
### Analysis of Usage Scenarios

We can now analyze the usage scenarios against the proposed remote I/O framework described in the previous section.

Scenario 1 is an example of a device initiating the discovery of and establishing a remote I/O session with a remote-capable application. The wireless remote in this case hosts a logical remote I/O device with input and output services, as well as control point functions. Figure 1 contains a graphic illustration of the following sequence:

1. *Discovery.* When Kathy powers up the wireless remote, the local control point logic on the remote queries the network for all application-server devices with an SSDP multicast (Figure 1). The application-server PC hosting the home automation application responds to the remote with a URL for retrieving the root of the XML application-server device description.
2. *Description.* Using HTTP GET, the remote device acquires the detailed XML description of the application server device. Another URL pointing to the XML description of the specific remote I/O services hosted by the application server is contained in the device description XML.

The wireless remote device automatically employs the URL pointing to the service description with HTTP GET to acquire the detailed XML listing of the



**Figure 1: UPnP remote I/O communications sequence between application server and wireless remote UI device for Scenario 1**

services hosted by the application server. The control logic on the remote parses the service XML and finds both remote I/O input and output services. The input and output service XML document lists API-like “actions” that can be called on the

application server by the remote device, including an action that returns a listing of all remote-enabled applications accessible through this application server.

3. *Control.* The wireless remote automatically issues a SOAP packet to the application server containing the API action that returns the remote-enabled application list. The XML list is processed for UI display. Kathy then selects the lighting control application causing the wireless remote to issue another SOAP packet containing another requested action on the application server. The parameter list of this particular action contains not only the selected lighting control application, but also the XML UPnP remote I/O device and UPnP remote I/O service description of the wireless remote itself.

The application server receives the SOAP packet containing the requested application along with the remote device description. At this point, control transfers from the remote device to the application server, as the application server establishes a UPnP remote I/O session with the wireless remote. The control logic on the application server matches a group of named events required by the application UPnP remote I/O input service with the softkey description contained in the XML UPnP remote I/O description of the wireless device.

4. *Out-of-band data transfer protocol and eventing.* An appropriately scaled image (possibly a bitmap) is forwarded from the application to the remote frame buffer. As Kathy touches the individual softkeys, corresponding command events are relayed to the subscribing input service on the application server and interpreted by the lighting control application.

Scenario 2 is an example of a remote-capable application initiating the discovery of compatible remote UI devices. The application server in this example hosts a logical application server device and registry service along with some control point functions.

1. *Discovery.* At installation time, using a discovery procedure identical to the one described in Scenario 1, the UPnP remote I/O application server hosting the home security system discovered all UPnP remote I/O devices and corresponding services. The security application established a table of remote I/O devices and their physical locations within the home.
2. *Description.* Prior to the alarm, the UPnP remote I/O application server hosting the home security system acquired the detailed device and service descriptions

from each of the UPnP remote I/O devices responding to the discovery multicast.

3. *Control.* The above information was used to pre-configure the control action requests generated by the home security application requesting to set up sessions with each of the remote devices, as well as to pre-scale displays appropriately for each of the UPnP remote I/O devices.
4. *Out-of-band data transfer protocol and eventing.* When the smoke alarm activates, all pre-configured remote I/O sessions are instantly activated. Bitmaps with explanation texts are forwarded to each display device using the data transmission protocol appropriate for each device. The wireless PDA that Dave grabs from the nightstand implements input as well as displays functions. When Dave uses his stylus to touch the screen location corresponding to “call fire department,” a command event is generated by the remote I/O output service on the wireless PDA. Dave’s intent is interpreted by the home security application and an alert signal is sent to Dave and Kathy’s security monitoring service, which in turn alert the local emergency services.

## EXISTING, RELATED TECHNOLOGIES AND STANDARDS

Existing remote I/O schemes fall into two broad categories:

- High-level methods for PC “desktop” remote User Interface (UI)
- Low-level data transport protocols for remote devices

Remote desktop methods are generally designed to allow other interface devices (especially portable tablets) to transparently share desktop application functionality with a specific host machine.

### HTML Presentation Pages

Under UPnP<sup>\*</sup> Architecture V1.0, if a UPnP device has a URL for presentation, then the control point can retrieve an HTML-based UI for controlling and/or viewing device status from this URL, load the page into a browser, and depending on the capabilities of the page, allow a user to control the device and/or view the device status [2]. The page is delivered via HTTP over TCP over IP. To retrieve a presentation page, the control point issues an HTTP GET request to the presentation URL, and the device returns a presentation page.

---

\* Other brands and names are the property of their respective owners.

Unlike the UPnP Device and Service Templates, and standard device and service types, the capabilities of the presentation page are completely specified by the vendor, not the UPnP Forum.

### Microsoft's RDP\*

Microsoft's Remote Desktop Protocol (RDP) [3] is designed to provide remote display and input capabilities over network connections for Windows\*-based applications running on a server. RDP is a protocol that allows for up to 64,000 separate virtual channels carrying device communication and presentation data from the server, as well as encrypted client mouse and keyboard data. RDP provides an extensible base from which to build many more capabilities. Other features of Microsoft RDP include the following:

- bandwidth-reduction measures
- support for roaming disconnect
- support for print redirection
- support for multipoint transmission

### Sun SLIM\*

Sun's SLIM [4] protocol is an experimental remote interface protocol that takes advantage of the fact that the display tends to respond to human input, which is quite slow. Refreshing the display from a local frame buffer and transmitting only pixel updates enable bandwidth savings between server and client. The SLIM client/console is essentially just a frame buffer. The server maintains the full, persistent contents of the frame buffer.

### EIA-775, 775.1

EIA-775 [5] is the DTV 1394 Interface Specification developed by CEA's R4.8 1394 Interface Committee. The low-level EIA-775 protocol provides a two-way bus type connection for high-end audio/video products. The bus architecture allows several devices to send and receive audio, video, and control information to all other devices on the bus. For example, when going from a TV to a DVD, a consumer needs only to insert the disk into the player and hit play. The player will send the right messages to the audio and video displays to set up for stereo, or multi-channel and standard, or wide screen—whatever the case may be. EIA 775.1, the WEB-Enhanced DTV 1394 Interface specification, additionally includes Web browser and other Internet protocols. These allow a source of MPEG service (such as a cable or terrestrial set-top box,

digital VCR or DTV) to utilize the MPEG decoding and display capabilities.

### HAVI DDI

The Home Audio/Video Interoperability (HAVI) Data-Driven Interaction (DDI) protocol [6] defines an API-based scheme for allowing soft entities designated as DDI Controllers to manipulate DDI Target objects. The DDI Controller employs a description of the UI (DDI Data) to be presented to the user, which is obtained from the DDI Target. The HAVI DDI specification describes APIs and objects that are used to establish control sessions between Controllers and Targets.

### Comparisons to UPnP Remote I/O

The essential component seen in virtually all of the technologies listed above is the type of specific, over-the-wire, data transmission protocol for exchanging UI information between an application and a remote UI device. Although these types of low-level communication protocol descriptions are indeed an essential requirement for remoting an application UI, they are insufficient for enabling general interoperability between applications and remote UI devices in a multi-vendor environment. In general, the piece that is missing in each of the above cases is a widely adopted framework for the discovery and enumeration of interoperable applications and UI devices.

### CONCLUSION

The goal of the proposed standard is the establishment of a UPnP\* remote I/O framework, allowing a fixed-location PC to serve user interfaces to a multitude of wired and wireless remote devices in the home. It is anticipated that the use of this new middleware standard to drive remote User Interface (UI) devices will effectively "free" the application user experience from the home PC platform.

### REFERENCES

- [1] Mark R. Walker, John G. Ritchie, Michael Jeronimo, "Remote I/O: A UPnP Forum Working Committee Proposal," October 2002.
- [2] *Universal Plug and Play Device Architecture* V1.0, June 2000.  
[http://www.upnp.org/download/UPnPDA10\\_20000613.htm](http://www.upnp.org/download/UPnPDA10_20000613.htm).
- [3] Remote Desktop Protocol (RDP) Features and Performance.

---

\* Other brands and names are the property of their respective owners.

---

\* Other brands and names are the property of their respective owners.

<http://www.microsoft.com/TechNet/prodtechnol/win2kts/evaluate/featfunc/rdpfpref.asp>

- [4] B.K. Schmidt, M.S. Lam, J.D. Northcutt, "The Interactive Performance of SLIM: A Stateless, Thin-Client Architecture," *Operating Systems Review*, vol. 34, 5, December 1999, pp 32-47.
- [5] DTV 1394 Interface Specification.
- [6] The HAVi Specification: "Specification of the Home Audio/Video Interoperability (HAVi) Architecture," V1.0, January 2000.

## AUTHORS' BIOGRAPHIES

**Mark R. Walker** has worked on numerous projects including PC-based video conferencing, wideband audio compression, and synthetic speech since joining Intel Architecture Labs in 1992. Mark received his Ph.D. degree from Arizona State University in 1991 and is now a member of the Intel Corporation's Solutions Architecture lab in Hillsboro, Oregon. Solutions Architecture is a part of Desktop Platform Architecture, which is an architecture organization within the Desktop Platform Group responsible for defining and developing platforms of the future. His e-mail is [mark.r.walker@intel.com](mailto:mark.r.walker@intel.com).

**Michael Jeronimo** is a software architect with Intel Corporation's Solutions Architecture lab in Hillsboro, Oregon, working on the architecture and technologies for the Digital Home. His interests include software architecture, UML, design patterns, and Internet security. Michael holds a B.A. degree in Computer Science from the University of California at Santa Cruz. His e-mail address is [Michael.Jeronimo@intel.com](mailto:Michael.Jeronimo@intel.com).

**Jim Edwards** is a lead architect and a technology development manager with Intel Corporation's Solutions Architecture lab in Hillsboro, Oregon, working on the architecture and technologies for the Digital Home. His interests include home networking, digital media, and sports cars. Jim holds a B.S. and M.S. degree in Electrical Engineering from Cornell University. His e-mail is [jim.edwards@intel.com](mailto:jim.edwards@intel.com).

**John G. Ritchie** joined Intel Corporation in 1986 and is a Staff Engineer with the Intel Corporation's Solutions Architecture lab in Hillsboro, Oregon. John is currently the co-chair of the UPnP AV working committee and was a principal architect and author of the UPnP AV specifications. John has been involved with other interoperability technologies since 1994 including CE-Bus, Home PnP, Home API, and HAVi. John earned his B.S. degree in Information and Computer Science from the University of California, Irvine in 1983. His e-mail address is [john.g.ritchie@intel.com](mailto:john.g.ritchie@intel.com).

**Ylian Saint-Hilaire** is a software architect/developer working on UPnP/AV solutions. During his career, Ylian has been responsible for the development of network infrastructure and technology in a variety of areas including Intel's IPsec key negotiation technology and network mobility technologies. Ylian is currently a member of the Intel Corporation's Solutions Architecture lab in Hillsboro, Oregon. His e-mail address is [ylian.saint-hilaire@intel.com](mailto:ylian.saint-hilaire@intel.com).

Copyright © Intel Corporation 2002. This publication was downloaded from <http://developer.intel.com/>.

Legal notices at <http://www.intel.com/sites/corporate/tradmarx.htm>

For further information visit:

[developer.intel.com/technology/itj/index.htm](http://developer.intel.com/technology/itj/index.htm)