

Interoperability

*Building Systems Working Together
For The Enterprise*

*XML
APIs
SDKs*

A Hirsch Electronics White Paper



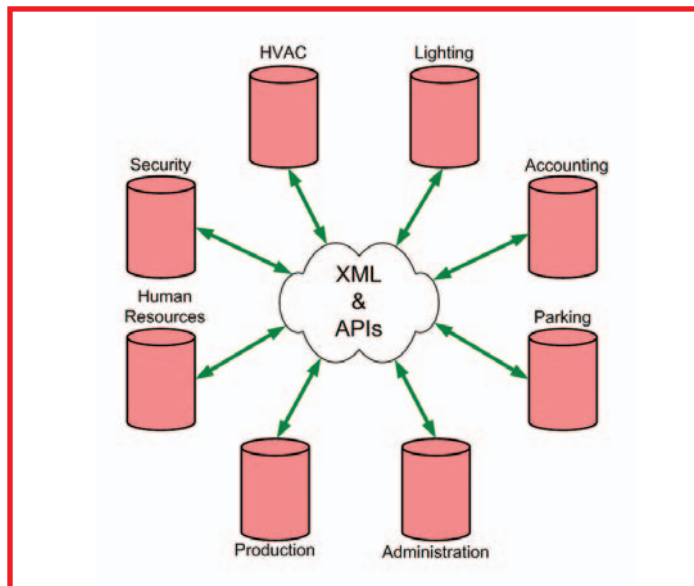
Introduction

Imagine the following scenario. A tenant or employee enters their building after hours by presenting a card to a reader, or entering a code on the keypad, at the main entrance. At that point several things happen. The door unlocks. The HVAC system is notified that the individual's office on the 5th floor needs to have its temperature setpoints changed to normal occupancy values so the individual is comfortable when they arrive. The lighting system is notified to turn on the appropriate lights for the office area on the 5th floor so the individual feels safe. Property management or the accounting department is notified of the exact time when the individual enters and leaves the building, so they can be billed for after-hours energy usage. Everyone benefits.

Today's facility executive wants to purchase systems and components from various manufacturers and have those components work together as one system. They want "open systems" that allow the manufacturers to offer new features and add value without locking the owner into a single vendor's proprietary protocols. In short, they want Interoperability.

Interoperability allows disparate systems to work together to deliver application solutions not possible with separate, proprietary offerings.

However, the enterprise has been plagued with "silos of information," which stand alone. Lots of information resides in separate databases and systems. So, if one is to optimize the enterprise, information needs to be shared. And, we need to share information dynamically.

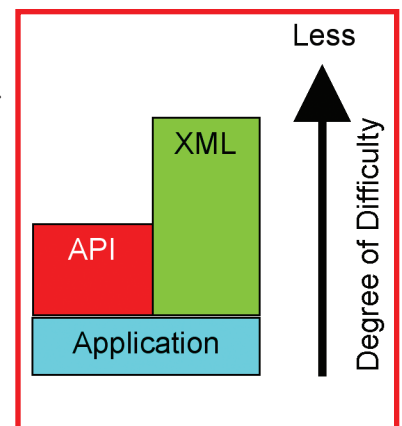


Fortunately, technologies are now being embraced that allow information to be shared between different manufacturers' proprietary application software. These technologies are available to both the application software supplier and to the IT department that provides the information infrastructure to the enterprise. Suppliers who create these technologies to make it easy for their system to interface and interoperate with another's, add value to the enterprise.

These technologies are the API, XML, and web servers.

What is an API?

An Application Programming Interface – or API – is essentially a set of conventions that allows a third party application to interoperate with the target application. The third party program need not know the details or native programming concepts of the target application.



Without an API, it can be virtually impossible for programmers or developers to create an interface that connects to or controls any aspect of the target application. With an API, interoperability is readily available to skilled developers to the extent the manufacturer exposes their functionality. The responsibility for providing an API belongs with the application software vendor.

The SDK

Ideally, a third party programmer or the end user can develop interfaces to a target application without dependence on the supplier. It is incumbent on the application software supplier to provide documentation and sample code to facilitate development of interfaces by others. The API, the documentation, and the sample code collectively make up a Software Development Kit, or SDK. Without an SDK, an API can be difficult or impossible to implement.

What is XML?

eXtensible Markup Language (XML) is fast becoming the worldwide standard for exchanging data because it is human readable, self describing, flexible, platform independent and widely supported.

Many people are familiar with XML's older cousin, HTML (Hyper Text Markup Language), the language of the World Wide Web. Web pages are written in HTML and browsers will read the HTML on virtually any computer. Yet, there are significant differences between XML and HTML. Indeed, XML is taking the web to the next level.

HTML is designed to "display" data and text and to focus on how data and text appear. XML is designed to "describe" data and to focus on delivering the data in a useful format. As a result, XML is playing an increasingly important role in the exchange of a wide variety of data on the Web, within the enterprise, and even on the desktop.

XML is a simple, very flexible text format derived from a language originally designed to meet the challenges of large-scale electronic publishing. It is written in plain text, usually English, making it easy to read, understand, and implement.

XML is also a data format. Like other common data formats (such as CSV, .dbf, .mdb, etc.), XML can be used in programs for calculations, sorts, queries, reports, displays, mergers of data, et cetera.

XML is not a programming language, per se. Nor is it a protocol. By itself, it does nothing. However, when data is stored and communicated in XML's text based format, programmers gain a powerful tool for the development of interoperability between applications.

A Markup Language applies "tags" before and after the data content to define that data. These opening and closing tags are enclosed by the caret or arrow symbols. A closing tag always has a forward slash as follows </closing tag>.

```
<?xml version="1.0" encoding="us-ascii" ?>
<Security XML REPORT="List Doors">
  <Authentication>
    <UserName>Joe Smith</UserName>
    <Password>JS55$$$$</Password>
    <Domain>HVAC</Domain>
    <Workstation>CUP</Workstation>
  </Authentication>
</Security XML>
```

The opening tag, the closing tag and the information in between constitute an "element." These elements are usually organized in an indented, outline style list to establish the relationship between the data and its purpose. An XML file that contains these elements is called a "document."

As shown in the preceding example, XML is written in plain text, and it is self describing. It is not too difficult to understand that an HVAC workstation in the Central Utilities Plant is requesting a list of doors from the Security System and is providing four factor authentication to the Security System in order to receive that information. User name, password, domain, and workstation must be correct to process the request.

Perhaps another application in the enterprise requires five factor authentication. An extra authentication element could be added for that application.

That is the eXtensible part. The benefit of an extensible language is that different applications use only what they need, and you can add on (extend) at anytime without disruption of the existing investment. Vendors and developers are encouraged to continue to add value to their offering without concerns of compatibility with the existing language. Everyone wins.

The flexibility of XML also comes from the ability for a developer to create their own tags. These tags can be used to "self-document" the data and make it easy for third party applications to identify and use the XML data.

Binary Protocols vs. XML

Today, most controllers communicate with devices or other controllers using a binary protocol. So, integration between two vendors means one has to acquire the binary protocol of the other and adapt to their proprietary, ever-changing rules. Industry standard protocols have overcome this problem to some extent.

Binary protocols (based on ones and zeros) have been around for decades. Although computers still talk in ones and zeroes, the gigabit Ethernet speeds available today make it practical to use human readable XML text for intranet and machine-to-machine (M2M) communications of data. After all, the Internet is built upon TCP/IP using HyperText Transport Protocol (HTTP) which is optimized for "text."

More importantly, binary protocols are not extensible. Once a binary protocol is established, it is not easily changed. Whether to describe the data on the magnetic stripe of your credit card or to define communications between controllers, a change would probably mean reissuing all credit cards, and updating or replacing controllers. With XML, another "element" can be added as easily as another item can be added to an outline in a Word document.

Let's take an example of a building with an existing XML enabled access control system. In a standalone mode, a reader transaction might provide the following data for other building and business systems in the enterprise.

```
<Message>
  <Parameters>
    <Door>data</Door>
    <Address>data</Address>
    <Reader>data</Reader>
    <DateTime>data</DateTime>
    <Description>data</Description>
    <EventID>data</EventID>
    <EventType>data</EventType>
  </Parameters>
</Message>
```

However, the facilities department wants to use entry/exit reader information from the anti-passback program to provide better modulation of outside air dampers for indoor air quality. Also, a presidential task force is being established to evaluate actual building occupancy patterns of both employees and visitors to reduce cost by setting better working hours and sharing resources among telecommuters. They want to include information indicating which zone the occupant is traveling from and to. This could easily be added (eXtended) by including two new elements in the XML message as follows:

```
<Message>
  <Parameters>
    <Door>data</Door>
    <Address>data</Address>
    <Reader>data</Reader>
    <DateTime>data</DateTime>
    <Description>data</Description>
    <EventID>data</EventID>
    <EventType>data</EventType>
    <FromZone>data</FromZone>
    <ToZone>data</ToZone>
  </Parameters>
</Message>
```

Only the access control system that implemented the anti-passback control with occupancy counting, the HVAC system, and the presidential task force database application using this information would need to be aware of the "From" and "To" elements added to the XML message. Other non-target applications could retain their legacy interfaces. XML extensibility allows tremendous modifications without incurring any of the re-development costs associated with traditional binary interface protocols.

Transformation of Data using XML

One of the advantages of XML is that any software developer can define their own tags. However, this can also be a problem if two developers use different tags to mean the same thing. For instance one manufacturer might use the tag "<door>" and another might use the tag "<portal>".

Fortunately, there are two solutions. One is the development of industry standards (addressed later). The other solution is to utilize XML's ability to support "Transformations." XML transforms can be used today while formal industry standards are in development.

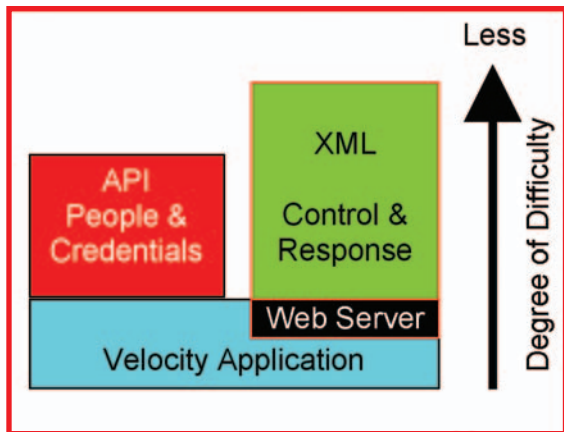
XSLT (eXtensible Stylesheet Language Transformations), which is also conveniently written in XML, provides a way to "map" or create a template that cross-references one developer's conventions to another's. So, if one manufacturer calls an entryway a <door> and another calls it a <portal>, the XSLT transform would make the differences transparent when the two applications exchange data.

Hirsch Embraces Interoperability

Hirsch Electronics has undertaken a new multi-faceted initiative to support interoperability. This initiative is based on the latest version of Hirsch's Velocity Security Management System application software.

It is easiest to understand how Velocity "exposes" its application software by thinking of two programs running on the Velocity application. First, there is an API for People and Credentials. This is the information generally stored in the Velocity server. Secondly, there is a web server and an XML interface for Control and Response functions. Both the API and XML allow interoperability to extend through the application software to the controllers for distributed execution.

An enterprise might use the People and Credentials API to interface the human resources operations with the physical security



application. This allows a new employee's personal and departmental information to be transferred from HR to security in order to automatically assign the new employee credential. The credential is immediately authorized for specific doors and is printed as a photo badge in the HR department to hand to the new hire with their orientation materials. Conversely, terminations are sent from HR to the security system to immediately disable the employee's access privileges everywhere.

All of this is possible because Velocity has an API available to third party applications which efficiently interacts with person and credential records. These records can be added, modified, and deleted. Applications such as visitor management can force a download of visitor credentials to the Velocity controller network for accountability records far superior to illegible sign-in logbooks commonly found at reception desks.

Technically speaking, the Velocity API is a DLL (dynamic link library) based on a programmable COM (Component Object Model) interface in the form of HirschVMO.dll. Using HirschVMO.dll, third parties can programmatically connect to the Velocity SQL database and interact with Person and Credential records without touching the data directly.

The Velocity API and XML have a full SDK to support developers of interoperable interfaces. It is vastly easier to develop an interface to Velocity using an API than to obtain and learn the underlying application code and database structure.

XML provides a different approach to simplifying interface development. This means certain program commands that are desired to be triggered from a third party application can be sent to Velocity through XML. For instance, severe weather alarms could

unlock doors. Conversely, Velocity can send triggers to the elevator equipment to activate certain floor buttons for an individual presenting a card or activating a unique keypad code. Transactional information can notify a nurse's station of which doctors are currently in the building. Alarms and arming status can be posted to a custom display in the command center of a central utilities district.

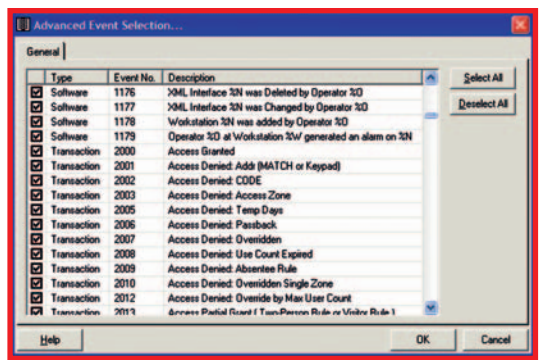
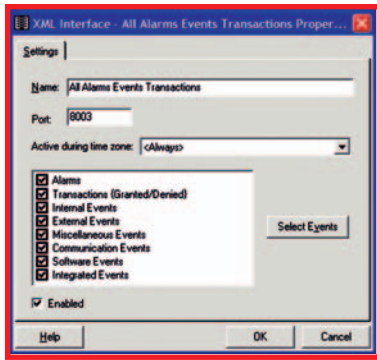
```
<?xmlversion="1.0" encoding="us -ascii" ?>
<Velocity.XML RPC="CommandSet.Execute">
  <Authentication>
    <UserName>Administrator </UserName>
    <Password> 123</Password>
    <Domain> Velocity</Domain>
    <Workstation> Mobile</Workstation>
  </Authentication>
  <Parameters>
    <Identity>5</Identity>
  </Parameters>
</Velocity.XML>
```

Again technically speaking, Velocity includes a web server that allows it to receive authenticated commands from an outside application and also allows Velocity to broadcast selected information to specific outside sources. An XML-RPC (Remote Procedure Call) is used to communicate with the Velocity web server.

Velocity uses its web server to communicate over the internet using HTTP. You will recognize http:// as the prefix of a web page in the address bar of your browser. The Velocity web server recognizes the HTTP "POST" and "GET" functions. A third party application can issue an HTTP POST to the Velocity web server that consists of an XML document that contains the correct information to request a "List" or issue a "Command." The Velocity web server will respond to a properly formatted and authenticated POST by allowing the third party application to GET an XML document that provides the information requested.

Each POST must be properly authenticated by including a name, password, security domain, and workstation identity. In effect, the Velocity web server considers the third party application to be a "client" and enforces the same NT authentication and Active Directory policy as is applied to a Velocity operator on a client terminal. Velocity will manage Port 80, processing POST and GET messages in XML format. By convention, Port 80 transitions firewalls to allow access for web browsers.

- ⚡ CommandSet.Execute
- ⚡ Door.MomentaryAccess
- ⚡ Door.Unlock
- ⚡ Door.Relock
- ⚡ Relay.Trigger
- ⚡ Relay.ForceOn
- ⚡ Relay.ForceOnRelease
- ⚡ Relay.ForceOff
- ⚡ Relay.ForceOffRelease
- ⚡ Relay.LockDown
- ⚡ Relay.LockDownRelease
- ⚡ Relay.LockOpen
- ⚡ Relay.LockOpenRelease
- ⚡ Input.Mask
- ⚡ Input.UnMask
- ⚡ Input.MomentaryMask
- ⚡ Alarm.Acknowledge
- ⚡ Alarm.Clear
- ⚡ ControlZone.Trigger
- ⚡ ControlZone.Mask
- ⚡ ControlZone.UnMask
- ⚡ ControlZone.ForceON
- ⚡ ControlZone.ForceOFF
- ⚡ ControlZone.ForceOnRelease
- ⚡ ControlZone.ForceOFFRelease
- ⚡ ControlZone.LockDown
- ⚡ ControlZone.LockDownRelease
- ⚡ ControlZone.LockOpen
- ⚡ ControlZone.LockOpenRelease
- ⚡ ControlZone.MomentaryMask
- ⚡ ControlZone.CancelEntryDelay
- ⚡ ControlZone.StartExitTimer
- ⚡ ControlZone.MaskAlarmCancelEntryDelay
- ⚡ UnMaskAlarmStartExitTimer
- ⚡ Settings.SetDate
- ⚡ Settings.SetTime
- ⚡ Settings.DownloadConfiguration
- ⚡ Settings.DownloadCredentials



List of commands that Velocity will accept from authenticated third party applications. These are accepted as a POST and return a GET. Together, these commands allow third party applications to remotely perform the key commands that a Velocity operator or controller logic package can perform.

Velocity supports a form of "discovery" by allowing a POST to obtain a list of related items or even a report which is a list of lists. A list will provide the common name of an item as well as its unique identifier assigned by the SQL Server database. This information can be used to issue a specific command to Velocity – such as "Unlock the Lobby Door" – or to develop a transform between two applications.

As an "add on" Server Extension, the Velocity web server will also broadcast serial data over a user defined port via its "XML Writer." The XML Writer allows the customer to choose what specific types of alarms, events, and transactions are sent to which port at what time. This way one or more target applications will receive only the information they need. This technique provides a form of subscription service.

The target application will typically parse and/or transform the data from the Velocity XML Writer so that it is usable locally.

Standards

Today, just about every industry is forming working groups, technical committees, etc. to develop the XML conventions to make it easy to use XML to exchange their industry specific data.

Significant standards activity is occurring in the oBIX (Open Buildings Interface Exchange) technical committee of OASIS (Organization for the Advancement of Structured Information

XML Writer. The top pane assigns a name, a port and an active time zone to the stream of information communicated from Velocity to another application. A second screen shows the extreme granularity available in determining what type of information should be sent out the specified port.

Standards). CABA (Continental Automated Buildings Association) served as an incubator for oBIX before it affiliated with OASIS for a closer association with the IT industry. The ASHRAE (American Society for Heating Refrigeration and Air conditioning Engineers) Standing Standard Project Committee 135, is actively developing an XML interface for their Data Communication Protocol for Building Automation and Control Networks (BACnet). SIA (the Security Industry Association) is also actively developing standards for interoperability.

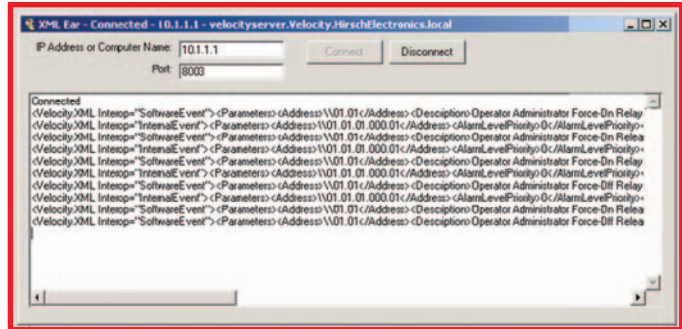
While official standards for each industry will take time to develop, many vendors will start using the preliminary templates and constructs proposed from the committee work prior to that. The demand is high, the benefit is great, and changes are easy to make. It's eXtensible.

Also of note, Microsoft is spending billions of dollars annually on XML and web services because of the business opportunities these tools open up. Their Office 2003 package stores data natively in XML, and Internet Explorer is XML enabled.

```

<xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"
xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
xmlns:rs="urn:schemas-microsoft-com:rowset"
xmlns:z="HRowsetSchema">
<s:Schema id="RowsetSchema">
<s:ElementType name="row" content="eltOnly" rs:CommandTimeout="30">
<s:AttributeType name="DoorID" rs:number="1">
<s:datatype dt:type="int" dt:maxLength="4" rs:precision="10"
rs:fixedLength="true" rs:mayBeNull="false"/>
</s:AttributeType>
<s:AttributeType name="DoorName" rs:number="2" rs:nullable="true"
rs:writeUnknown="true">
<s:datatype dt:type="string" dt:maxLength="32"/>
</s:AttributeType>
<s:AttributeType name="Controller_Name" rs:number="3"
rs:nullable="true" rs:writeUnknown="true">
<s:datatype dt:type="string" dt:maxLength="32"/>
</s:AttributeType>
<s:AttributeType name="Index" rs:number="4" rs:nullable="true"
rs:writeUnknown="true">
<s:datatype dt:type="int" dt:maxLength="2" rs:precision="5"
rs:fixedLength="true"/>
</s:AttributeType>
<s:extends type="rs:rowbase"/>
</s:ElementType>
</s:Schema>
<rs:data>
<z:row DoorID="3" DoorName="Archives" Controller_Name="M8"
Index="3"/>
<z:row DoorID="2" DoorName="Computer Room" Controller_Name="M8"
Index="2"/>
<z:row DoorID="4" DoorName="Loading Dock" Controller_Name="M8"
Index="4"/>
<z:row DoorID="1" DoorName="Main Entrance" Controller_Name="M8"
Index="1"/>
<z:row DoorID="7" DoorName="Parking Gate" Controller_Name="M8"
Index="7"/>
<z:row DoorID="5" DoorName="Pharmacy" Controller_Name="M8"
Index="5"/>
<z:row DoorID="8" DoorName="Research and Development"
Controller_Name="M8" Index="8"/>
<z:row DoorID="6" DoorName="Stockroom" Controller_Name="M8"
Index="6"/>
<z:row DoorID="9" DoorName="Guard Station" Controller_Name="Model 1"
Index="1"/>
</rs:data>
</xml>

```



XML Ear. Hirsch Professional Services developed this program to demonstrate the Velocity XML Writer broadcasting alarms and events. It will paint one line at a time and is reminiscent of the Velocity Event Viewer. The XML Ear is part of the SDK.

In the not-so-distant future, end-users and systems integrators will be able to obtain off the shelf interoperability applications for specific manufacturers. Those applications will provide standard templates and mapping tools to join the silos of information that exist among the various systems within the enterprise. New solutions will emerge that result from interoperability among two or more applications

Velocity is the Interoperability Platform of choice for a security system in the "intelligent enterprise."

Links

www.HirschElectronics.com

www.w3schools.com/xml

www.w3.org/XML

www.oreillynet.com/faqs/list.csp?id_subject=23

www.xml.com

www.oasis-open.org/who



Example of a GET response to a POST request for a list of doors. The response includes XML schema in the upper half of the display. The schema is a Microsoft ADO (Active X Data Object) Record Set for ease of implementation.

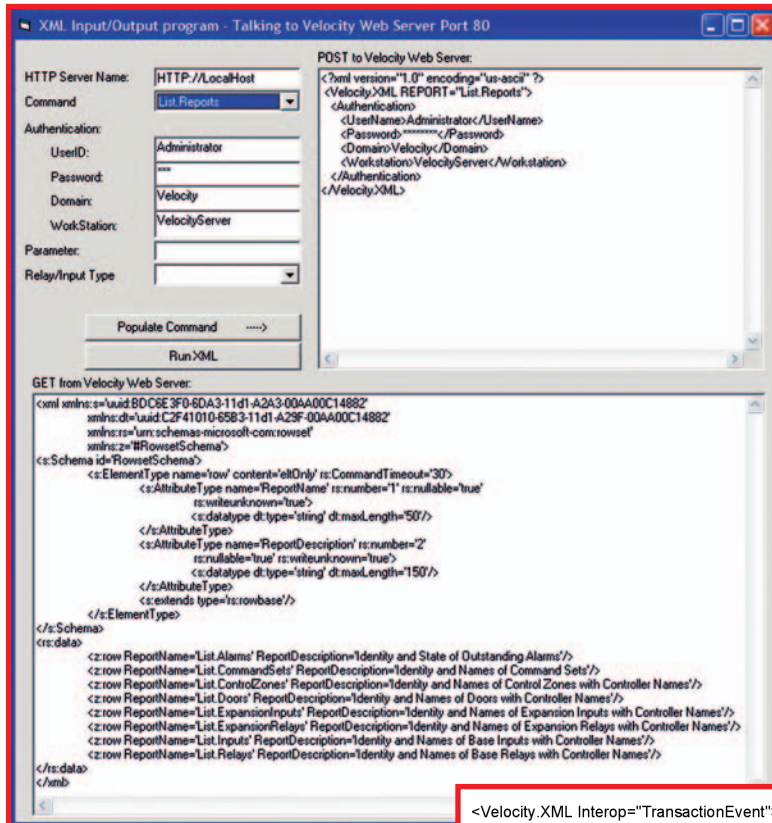
Many software companies are developing new applications, including shareware and freeware, to simplify creating and using XML documents.

Professional Services

Recognizing that many customers may want to contract for implementations of the Hirsch API or XML capability, Hirsch offers a Professional Services Group. They perform consulting, analysis, programming, and development services to assist the customer in linking the Hirsch Velocity system to a third party business or control system capable of interoperability.

Vision

Customers are now free to choose individual building control systems. As long as they choose systems that are truly open – like those based on XML and APIs with SDKs and professional services available – they will enjoy interfaces that are flexible, customer controlled, and yield a strong ROI.



XML Input/Output program. Hirsch Professional Services developed this program to make it easy to demonstrate XML in action. Included in the SDK, it uses a combo (pull down) box to select the XML text from a library and drop it into the POST window. Clicking the Run XML results in the GET pane being populated by the Velocity web server.

Typical serial information associated with an Access Granted transaction received from the XML Writer.

```

<Velocity.XML Interop="TransactionEvent">
  <Parameters>
    <Address>\01.01.01.000.01.SM01</Address>
    <AlarmLevelPriority>0</AlarmLevelPriority>
    <CARD></CARD>
    <ControllerDate Time>5/19/2004 3:17:08 PM</ControllerDate Time>
    <ControllerID>2</ControllerID>
    <Description>Access Granted Sam Spade Pharmacy Exit-Reader 09</Description>
    <Disposition>0</Disposition>
    <DomainID>1</DomainID>
    <DoorOrExpansion>1</DoorOrExpansion>
    <DTAddress>1</DTAddress>
    <EventID>3122</EventID>
    <EventType></EventType>
    <FromZone>0</FromZone>
    <PIN></PIN>
    <PortAddress>1</PortAddress>
    <Reader>9</Reader>
    <ReaderType>192</ReaderType>
    <ReportAsAlarm>False</ReportAsAlarm>
    <ServerDate Time>5/19/2004 3:17:09 PM</ServerDate Time>
    <ServerID>1</ServerID>
    <ToZone>0</ToZone>
    <TransactionType>33</TransactionType>
    <UID1>6</UID1>
    <UID2>3</UID2>
    <VelocityEventType>Digi+Trac Transaction</VelocityEventType>
    <XAddress>0</XAddress>
    <Zone>1</Zone>
  </Parameters>
</Velocity.XML>

```



Specifications are subject to change without notice.
©2004 Hirsch Electronics Corp. All rights reserved.

Global Headquarters

1900-B Carnegie Ave., Santa Ana, CA 92705 USA
Phone: 949-250-8888 Fax: 949-250-7372
Web: www.HirschElectronics.com
E-mail: info@hirschelectronics.com