



Interoperability Development Kit Reference Guide

January 24, 2005

Velocity IDK Reference Guide

IDK001 January, 2005

Revision B

Copyright© 2005 Hirsch Electronics Corporation. All rights reserved. ScramblePad® and ScrambleProx® are registered trademarks of Hirsch Electronics Corporation. DIGI*TRAC™, MATCH™, ScrambleCard™, SCRAMBLE*NET™ (abbreviated S*NET), X*NET, and Velocity™ are all trademarks of Hirsch Electronics Corporation.

Microsoft, MS and MS-DOS are registered trademarks, and Windows, Windows 2000, and Windows XP are trademarks of Microsoft Corporation.

Hirsch Electronics Corporation
1900-B Carnegie Avenue
Santa Ana, CA 92705-5520

Phone:949-250-8888 or 888-809-8880 (toll-free)
Fax:949-250-7372
Web:www.HirschElectronics.com



Getting Help

If you encounter a problem that is not discussed in this guide and you need technical support, do the following:

1. Contact your local dealer or the provider of this product.
2. If your dealer is not available, contact Hirsch Technical Support directly. This can be done in a number of ways:

Mail: Hirsch Electronics Corporation
1900-B Carnegie Avenue
Santa Ana, CA 92705-5520

Attn: Professional Services

Phone: 877-HIRSCHX (877-447-7249) toll-free

Fax: (949) 250-7362

Email: support@HirschElectronics.com

WWW: www.HirschElectronics.com

Whenever you call your local dealer or Hirsch, be sure to have your registration material, serial number and software version number available.

For future reference, record these numbers here.

Serial Number: _____

Version Number: _____

Dealer: _____

Dealer Phone #: _____

CCM Rev/Version #: _____

Contents

Getting Help	iii
Introduction.....	1
Installing IDK.....	5
Hirsch VMO API	7
VMO Class Properties	9
Credential	13
Properties	13
Methods	14
Events	15
Remarks	15
CredentialCollection	16
Properties	16
Methods	16
Events	16
Remarks	16
CredentialTemplate	18
Properties	18
Methods	18
Events	18
Remarks	18
CredentialTemplateCollection	19
Properties	19
Methods	19
Events	19
Remarks	19
ImagePlaceholder	20
Properties	20
Methods	20

Events	20
ImagePlaceholderCollection	21
Properties	21
Methods	21
Events	21
Remarks	21
Person	23
Properties	23
Methods	31
Events	32
Remarks	32
PersonCollection	33
Properties	33
Methods	34
Events	34
Remarks	34
PersonImage	37
Properties	37
Methods	37
Events	38
Remarks	38
PersonImageCollection	39
Properties	39
Methods	39
Events	39
Remarks	39
Selection	41
Properties	41
SelectionCollection	42
Properties	42
Methods	42
Events	42
UserDefinedField	43

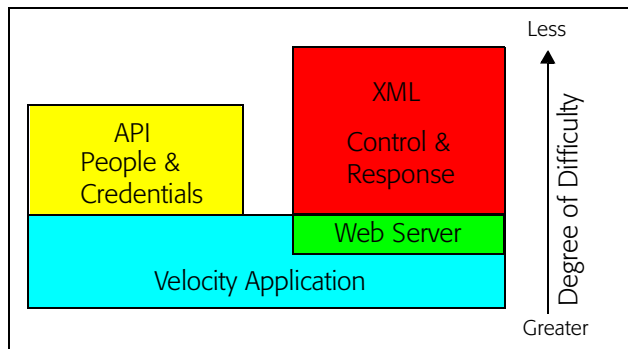
Properties	43
Methods	43
Events	43
Remarks	44
UserDefinedFieldCollection	45
Properties	45
Methods	45
Events	45
Remarks	45
UserDefinedFieldSetup	46
Properties	46
Methods	47
Events	47
Remarks	47
UserDefinedFieldSetupCollection	48
Properties	48
Methods	48
Events	48
Remarks	48
VelocityServer	49
Properties	49
Methods	50
Events	51
Remarks	57
VB Calls Sample	58
XML Interoperability Services.....	61
Remote Procedure Calls	63
Acknowledging and Clearing Alarms.....	63
Acknowledge	64
Clear	64
SQL Statement Execution (Insert, Update, Delete, Query, etc.)	64
Channel Responses	65
Error Handling	65

Discovery	66
Default Supported Lists	67
Receiving Alarms & Events	68
Alarm & Event Messaging.....	68
Non-Alarm Event	68
Alarm.....	69
Supported Remote Procedure Calls.....	70
Communicating with Velocity Web Server	72
Executing Remote Procedure Calls.....	72

Introduction

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. Second, there is a web server and an XML interface for Control and Response functions. Both the API and XML enable interoperability to extend through the application software to the connected controllers for distributed execution.



A company might use the People and Credentials API to interface 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 other orientation materials. Conversely, terminations are sent from HR to Security to immediately disable the employee's access privileges throughout company facilities.

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

Technically, the Velocity API is a DLL (Dynamic Link Library), *HirschVMO.dll*, based on a programmable COM (Component Object Model) interface. Using *HirschVMO.dll*, third parties can programmatically connect to the Velocity SQL database and interact with the 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 far 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 strategy for simplifying interface development. In this approach, commands required to trigger certain Velocity functions from a third-party application can be sent to Velocity via XML. For example, severe weather alarms could unlock doors. Velocity might also send triggers to elevator equipment to activate floor buttons for an individual presenting a card or activating a unique keypad code. Transactional information can notify a nurse's station concerning 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.

Velocity includes a web server that enables it to receive authenticated commands from an outside application and broadcast selected information to specific outside sources. An XML-RPC (Remote Procedure Call) is used to communicate with the Velocity web server. Velocity uses this web server to communicate over the internet using HTTP.

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 containing the correct information to request a List or issue a Command. The Velocity web server then responds to a properly formatted and authenticated POST by allowing the third-party application to GET an XML document that provides the requested information. Each POST must be properly authenticated by including a name, password, security domain, and workstation ID. 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 manages Port 80, processing POST and GET messages in XML format. By convention, Port 80 allows communication across firewalls for web browsers.

Velocity also supports a form of 'discovery' by enabling a POST to obtain a list of related items or even a report which is a list of lists. A list provides the command 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.

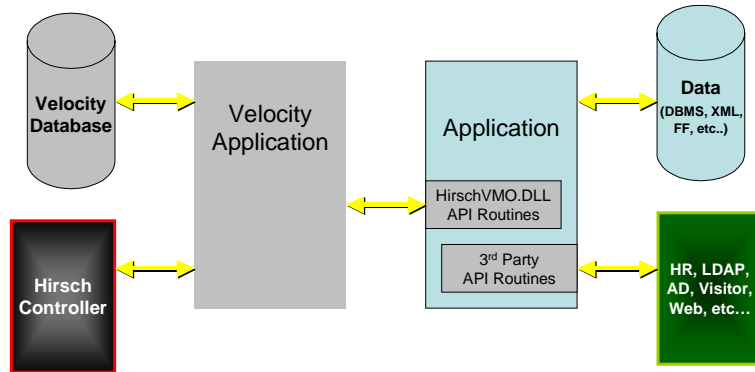
As an add-on server extension, the Velocity web server can also broadcast serial data over a user-defined port via its XML Writer. The XML Writer enables the customer to choose what specific types of alarms, events, and transactions are sent to which port at what time. In this way one or more target applications can receive only the information they need, providing a sort of subscription service. The target application typically parses or transforms the data from the Velocity XML Writer so that is usable locally.

The new Hirsch Velocity IDK consists of two modules:

- Hirsch Velocity Management Object (VMO) API routines
- Velocity XML Interoperability Services

These two modules enable the qualified programmer to customize their application for interaction with Velocity.

As shown in the following diagram, Velocity can now exchange specific types of data using VMO API routines and export data in XML form.



Through HirschVMO.DLL, a programmer can access data on persons and credentials within the Velocity database, reformat that data as required, even modify the data and return it to the Velocity database. Third-party applications, such as human resource or accounting spreadsheets, can then share security information for seamless exchanges of data. In this way, for example, an employee who HR terminates, can have all security privileges simultaneously suspended if the programmer customizes the HR software to interoperate with Velocity through HirschVMO.

For information on installing IDK software, see "Installing IDK," starting on page 5.

For more on HirschVMO, see "Hirsch VMO API" starting on page 7.

Velocity's new XML interoperability services enable a qualified programmer to receive XML data from Velocity through Velocity's XML Writer. This data can then be reinterpreted as required for a third-party application. Since XML Writer outputs raw XML files, the programmer must be familiar with the manipulation of XML: essentially, how to cross-match tags using XML schemas.

For more on this, see "XML Interoperability Services" starting on page 61.

Installing IDK

Installing the Velocity IDK is simple. First, purchase the IDK package from Hirsch Electronics. (For the relevant Hirsch Electronics phone number, refer to page iii of this manual.) The required IDK zip file is contained on a CD Hirsch will ship you.

1. Insert the IDK CD in your CD drive.
2. Copy the *IDK.zip* file to your hard drive.
3. Unpack the zip file.
4. Drill down to *setup.exe*.
5. Double click **setup.exe**.
6. Follow these steps:

At this screen:	Do this:
Welcome	Click Next .
Destination Location	Click Next .
Start Installation	Click Next . The IDK files are installed.
Hirsch IDK Integration has been successfully installed	Click Finish .

The installation is now complete.

The IDK documentation in Acrobat format is copied to the *Program Files\Hirsch Electronics\Velocity\Documentation* folder.

The IDK examples are copied to the *Program Files\Hirsch Electronics\Velocity\Sample Code* folder. These files include:

- FunctionsWrapper.cls
- HirschAPIWrap.dat
- HirschAPIWrap.vbp
- HirschAPIWrap.vbw

You should now have all the files you need to develop programs for Velocity.

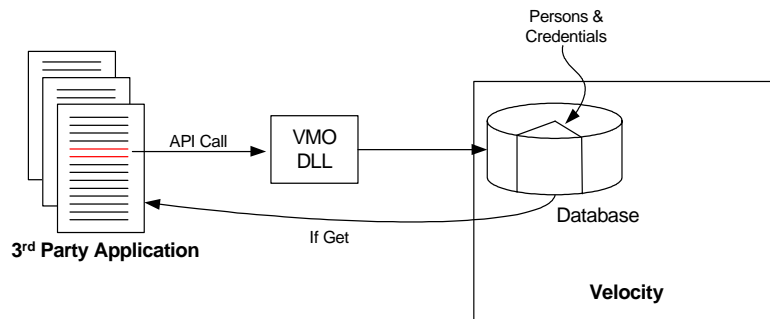
Hirsch VMO API

Velocity Management Object (VMO) provides a programmable COM interface in the form of VMO.DLL to protect internal data integrity yet provide an open architecture to third-party vendors who require an interaction between Velocity and their own systems.

The VMO API includes:

- VMO.DLL that must be installed before you can access Person and Credential records in Velocity.
- A sample COM project, VMOSample.exe, that runs in most development environments and shows how a VMO routine can be written.
- On-line HTML documentation.

Using HirschVMO.DLL, third parties can connect to the Velocity SQL database and interact with the Person and Credential records without touching the data directly. This provides a structured method for adding, updating, or deleting Person and Credential records from programs external to Velocity as shown in this diagram:



The interface provides access to the entire Person database as well as specific Person records on demand. Person records can also be traversed to search Credential records.

The interface also provides a read-only OO interface for User-Defined Field customized names, and read-write for specialized values (lists).

The interface supports defining criteria for traversing Person (plural) records with the use of an end-user definable filter. An embedded filter generator is included to simplify end-designer functionality.

The interface has been tested for Microsoft Visual Basic. COM interfaces are typically supported by other Microsoft Visual Studio products and Borland Delphi, as well as most implementations of Sun Microsystem's Java; however, these implementations will not be tested or guaranteed by Hirsch Electronics.

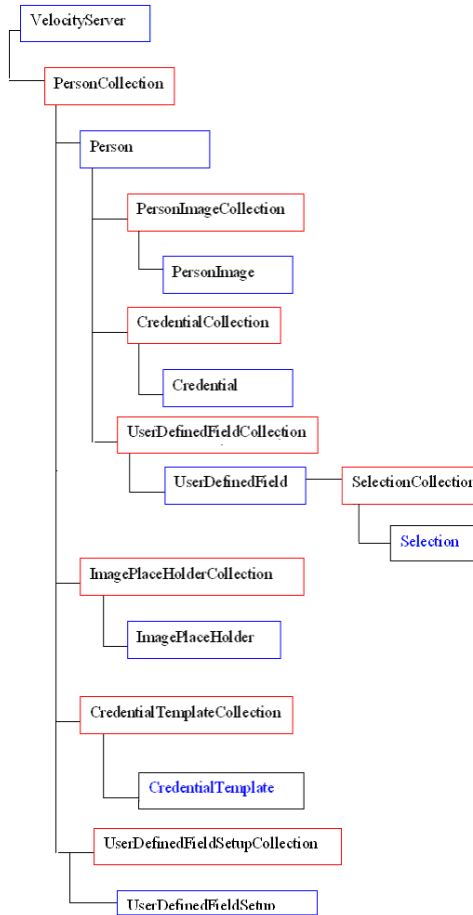
The IDK includes a sample project, VMOSample.exe, that can be opened and studied using most development environments.

VMO Class Properties

The classes comprising this VMO are:

- Credential
- CredentialCollection
- CredentialTemplate
- CredentialTemplateCollection
- ImagePlaceHolder
- ImagePlaceHolderCollection
- Person
- PersonCollection
- PersonImage
- PersonImageCollection
- Selection
- SelectionCollection
- UserDefinedField
- UserDefinedFieldCollection
- UserDefinedFieldSetupCollection
- VelocityServer

The hierarchy of these classes is shown in the following illustration:



As shown above, classes are aligned in a hierarchical fashion with VelocityServer at the top, PersonCollection as an one of the second order classes, and Person, UserDefinedFieldCollection, ImagePlaceHolderCollection, and CredentialTemplateCollection representing nested third-order classes.

In HirschVMO.dll, all classes except `velocityServer` have their Instancing Properties set to `PublicNotCreatable` which means neither the syntax:

```
Dim obj as New ClassName
nor
Dim obj as ClassName
Set obj = New ClassName
```

are supported.

You can only instantiate a class by calling the `Item` property of its parent collection class or calling the `AddNew` method of the collection class if there is one. For example, to obtain the credential object of ID 12 of a Person with ID 16, use:

```
Velocity.Persons(16).Credentials(12)
```

assuming that you have created an `velocityServer` object called `velocity` and successfully called the `Connect` method after it is instantiated.

All collection classes support the `NewEnum` interface which allows enumerating of the collection using this syntax:

```
For each obj in colObject
Next
```

This method, however, can be very processor-intensive, particularly if there are many elements in the collections. For example, if there are thousands of users in the system, enumerating the `Persons` collection could be time-consuming and monopolize processor time. Therefore, only use this technique when it is necessary.

If you only need to obtain one element whose Identity is known, use the collection's `Item` property.



The `Item` property takes an Identity parameter instead of a collection item index.

A practical way to use this SDK is to enumerate a collection once, record the IDs of every element in that collection then use the `Item` property to obtain a single object when needed. Since the `Item` property of the collection classes takes an Identity parameter instead of a collection item index, you cannot enumerate the collection using this method:

```
For Index = 0 to colObject.Count - 1
  Debug.Print colObject(Index).Name
Next
```

Each class and its attending properties are described on the following pages.

Credential

Instancing: Public Not Creatable

MTS Mode: Not an MTS object

Persistence: Not Persistable

Properties

PersonId [Get] - Public (Return Datatype: Long) Read-only

No Parameters

CredentialId [Get] - Public (Return Datatype: Long) Read-only

No Parameters

CardData [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	String

CardData [Get] - Public (Return Datatype: String)

No Parameters

HostDescription [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	String

HostDescription [Get] - Public (Return Datatype: String)

No Parameters

IDF [Get] - Public (Return Datatype: Byte) Read-only

No Parameters

IDFString [Get] - Public (Return Datatype: String)

No Parameters

Returns a descriptive string of the IDF Format

MATCH [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	String

MATCH [Get] - Public (Return Datatype: String)

No Parameters

PIN [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	String

PIN [Get] - Public (Return Datatype: String)

No Parameters

StampNo [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	String

StampNo [Get] - Public (Return Datatype: String)

No Parameters

TemplateId [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	Long

Provide the template ID based on which the credential is created

Methods

GetFunctionSummaryDescription - Public (Return Datatype: String)

No Parameters

returns a descriptive string of the function and function category

LockCredential - Public

No Parameters

UnLockCredential - Public

No Parameters

Update - Public

No Parameters

Events

None

Remarks

Use the Item method of the CredentialCollection object to obtain an existing Credential object, for example:

```
Set oCredential = Velocity.Persons(1).Credentials(1230)
```

To obtain the database identity of a person, you need to enumerate the Credentials Collection of a person at least once then store the CredentialID of each Credential object locally for future reference.

Existing Credentials are read-only. The Update method only saves changes to a Credential object if this is a new Credential obtained from CredentialCollection's AddNew method.

To create a Credential:

1. Obtain a CredentialCollection object of a person.
2. Call the AddNew method of the CredentialCollection object to return a new Credential object.
3. Set the properties of the new Credential object. Most importantly, set the TemplateID property (write-only) so a new credential is created based on the specified Credential template.

If a keypad only template is used, the PIN property must be provided. If a card-only template is used, the MATCH property or CardData property must be provided. If any other IDF format template is used, the combination of the above must be provided.

4. Call the Update method of the Credential object:

```
Dim oCredential as VMO.Credential  
Set oCredential = Velocity.Persons(1).Credentials.AddNew  
'Set the Credential properties  
oCredential .Update
```



Creating or removing a person by using the CredentialCollection requires the operator to have certain privilege.

CredentialCollection

Instancing: Public Not Creatable
MTS Mode: Not an MTS object
Persistence: Not Persistable

Properties

PersonId [Get] - Public (Return Datatype: Long) - Read-only

No Parameters

Count [Get] - Public (Return Datatype: Long)

No Parameters

Item [Get] - Public (Return Datatype: Credential) Default Property

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	CredentialId	Long

Methods

AddNew - Public (Return Datatype: Credential)

No Parameters

Remove - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	CredentialId	Long

Events

None

Remarks

With `CredentialCollection`, you can:

- Get the number of Credentials of a person using the Count property.
- Obtain a Credential object of a person using the Item property.
- Create a Credential for a person based on a template.

- Delete a Credential from a person.

To create a Credential for a person:

1. Obtain a CredentialCollection object of a person.
2. Call the AddNew method of the CredentialCollection object to return a new Credential object.
3. Set the properties of the new Credential object. Most importantly, set the TemplateID property (write-only) so a new credential is created based on the specified Credential template.

If a keypad only template is used, the PIN property must be provided. If a card-only template is used, the MATCH property or CardData property must be provided. If any other IDF format template is used, the combination of the above must be provided.

4. Call the Update method of the Credential object:

```
Dim oCredential as VMO.Credential
Set oCredential = Velocity.Persons(1).Credentials.AddNew
'Set the Credential properties
oCredential .Update
```

To delete a Credential for a person:

1. Use the Remove method of the CredentialCollection object and pass in the CredentialID to remove the targeted Credential.
`Velocity.Persons(1).Credentials.Remove(1230)`
2. When using the Item or Remove method, CredentialCollection supports member identification only using its database Identity. For example:
`Set oCredential = Velocity.Persons(1).Credentials(1230)`
3. To obtain the database identity of a credential, enumerate the Credentials Collection at least once then store the CredentialID of each Credential object locally for future reference.



Creating or removing a person by using the

CredentialCollection requires the operator to have certain privilege.

CredentialTemplate

Instanting: MultiUse
MTS Mode: Not an MTS object
Persistence: Not Persistable

Properties

TemplateId [Get] - Public (Return Datatype: Long) Read-only

No Parameters

IDF [Get] - Public (Return Datatype: Byte) Read-only

No Parameters

IDFString [Get] - Public (Return Datatype: String) Read-only

No Parameters

returns a descriptive string of the IDF Format

**HostDescription [Get] - Public (Return Datatype: String)
Read-only**

No Parameters

Methods

GetFunctionSummaryDescription - Public (Return Datatype: String)

No Parameters

returns a descriptive string of the function and function category

Events

None

Remarks

With the `CredentialTemplate` object, you can read the ID, IDF and description of an existing `CredentialTemplate`. This is read-only.

CredentialTemplateCollection

Instanting: MultiUse
MTS Mode: Not an MTS object
Persistence: Not Persistable

Properties

Count [Get] - Public (Return Datatype: Long)

No Parameters

Item [Get] - Public (Return Datatype: CredentialTemplate) Default Property

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	CredentialId	Long

Methods

None

Events

None

Remarks

With the `CredentialTemplateCollection`, you can:

- Get the number of `CredentialTemplates` in the system using the `Count` property.
- Obtain a `CredentialTemplate` object using the `Item` property.

ImagePlaceholder

Instanting: Public Not Creatable
MTS Mode: Not an MTS object
Persistence: Not Persistable

Properties

Caption [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	String

Caption [Get] - Public (Return Datatype: String)

No Parameters

ImagePlaceholderID [Get] - Public (Return Datatype: Long) Read only

No Parameters

Methods

Update - Public

No Parameters

Events

None

ImagePlaceholderCollection

Instancing: Public Not Creatable
 MTS Mode: Not an MTS object
 Persistence: Not Persistable

Properties

Count [Get] - Public (Return Datatype: Long)

No Parameters

Item [Get] - Public (Return Datatype: ImagePlaceholder) Default Property

Optional	ByVal / ByRef	Variable	Datatype
		ImagePlaceholder ID	Long

Methods

AddNew - Public (Return Datatype: ImagePlaceholder)

No Parameters

Remove - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	ImagePlaceholder ID	Long

Events

None

Remarks

With `ImagePlaceholderCollection`, you can:

- Get the number of Image Place Holders in the system using the `Count` property.
- Obtain an `ImagePlaceholderCollection` object using the `Item` property.
- Create an `ImagePlaceholder`.
- Remove an `ImagePlaceholder`.

To create an ImagePlaceholder:

1. Call the AddNew method of the ImagePlaceholderCollection object to return a new ImagePlaceholder object.
2. Set the properties of the new ImagePlaceholder object.
3. Call the Update method of the ImagePlaceholder object:

```
Dim oImagePlaceholder as VMO.ImagePlaceholder
Set oImagePlaceholder =
    Velocity.Persons.ImagePlaceholders.AddNew
'Set the oImagePlaceholder properties
oImagePlaceholder .Update
```

To remove an ImagePlaceholder:

1. Use the Remove method of the ImagePlaceholderCollection object and pass in the ImagePlaceholderID to remove the targeted ImagePlaceholder.
Velocity.Persons.ImagePlaceholders.Remove(1)
2. When using the Item or Remove method, ImagePlaceholderCollection supports member identification only using its database Identity.

For example:

```
Set oImagePlaceholder =
    Velocity.Persons.ImagePlaceholders(1)
```

3. To obtain the database identity of an ImagePlaceholder, enumerate the ImagePlaceholders Collection at least once then store the ImagePlaceholderID of each ImagePlaceholder object locally for future reference.



Creating or removing a ImagePlaceholder by using the ImagePlaceholderCollection requires the operator to have certain privilege.

Person

Instanting: Public Not Creatable

MTS Mode: Not an MTS object

Persistence: Not Persistable

Properties

PersonId [Get] - Public (Return Datatype: Long)

Database Identity of a person record. Read-only.

No Parameters

FirstName [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

FirstName [Get] - Public (Return Datatype: String)

No Parameters

LastName [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

LastName [Get] - Public (Return Datatype: String)

No Parameters

MiddleName [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

MiddleName [Get] - Public (Return Datatype: String)

No Parameters

Photo [Set] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	Std Picture

Photo [Get] - Public (Return Datatype: Object)

No Parameters

Signature [Set] - Pub

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	Std Picture

Signature [Get] - Public (Return Datatype: StdPicture)

No Parameters

Suffix [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

Suffix [Get] - Public (Return Datatype: String)

No Parameters

Title [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

Title [Get] - Public (Return Datatype: String)

No Parameters

UDF01 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF01 [Get] - Public (Return Datatype: String)

No Parameters

UDF02 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF02 [Get] - Public (Return Datatype: String)

No Parameters

UDF03 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF03 [Get] - Public (Return Datatype: String)

No Parameters

UDF04 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF04 [Get] - Public (Return Datatype: String)

No Parameters

UDF05 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF05 [Get] - Public (Return Datatype: String)

No Parameters

UDF06 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF06 [Get] - Public (Return Datatype: String)

No Parameters

UDF07 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF07 [Get] - Public (Return Datatype: String)

No Parameters

UDF08 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF08 [Get] - Public (Return Datatype: String)

No Parameters

UDF09 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF09 [Get] - Public (Return Datatype: String)

No Parameters

UDF10 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF10 [Get] - Public (Return Datatype: String)

No Parameters

UDF11 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF11 [Get] - Public (Return Datatype: String)

No Parameters

UDF12 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF12 [Get] - Public (Return Datatype: String)

No Parameters

UDF13 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF13 [Get] - Public (Return Datatype: String)

No Parameters

UDF14 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF14 [Get] - Public (Return Datatype: String)

No Parameters

UDF15 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF15 [Get] - Public (Return Datatype: String)

No Parameters

UDF16 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF16 [Get] - Public (Return Datatype: String)

No Parameters

UDF17 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF17 [Get] - Public (Return Datatype: String)

No Parameters

UDF18 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF18 [Get] - Public (Return Datatype: String)

No Parameters

UDF19 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF19 [Get] - Public (Return Datatype: String)

No Parameters

UDF20 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF20 [Get] - Public (Return Datatype: String)

No Parameters

UDF21 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF21 [Get] - Public (Return Datatype: String)

No Parameters

UDF22 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF22 [Get] - Public (Return Datatype: String)

No Parameters

UDF23 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF23 [Get] - Public (Return Datatype: String)

No Parameters

UDF24 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF24 [Get] - Public (Return Datatype: String)

No Parameters

UDF25 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF25 [Get] - Public (Return Datatype: String)

No Parameters

UDF26 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF26 [Get] - Public (Return Datatype: String)

No Parameters

UDF27 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF27 [Get] - Public (Return Datatype: String)

No Parameters

UDF28 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF28 [Get] - Public (Return Datatype: String)

No Parameters

UDF29 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF29 [Get] - Public (Return Datatype: String)

No Parameters

UDF30 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF30 [Get] - Public (Return Datatype: String)

No Parameters

UDF31 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF31 [Get] - Public (Return Datatype: String)

No Parameters

UDF32 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF32 [Get] - Public (Return Datatype: String)

No Parameters

UDF33 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF33 [Get] - Public (Return Datatype: String)

No Parameters

UDF34 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF34 [Get] - Public (Return Datatype: String)

No Parameters

UDF35 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF35 [Get] - Public (Return Datatype: String)

No Parameters

UDF36 [Let] - Public

Optional	ByVal / ByRef	Variable	Datatype
	ByVal	vData	String

UDF36 [Get] - Public (Return Datatype: String)

No Parameters

Methods**LockPerson - Public**

No Parameters

UnLockPerson - Public

No Parameters

Update - Public

No Parameters

Events

None

Remarks

Use the `Item` method of the `PersonCollection` object to obtain an existing `Person` object. For example:

```
Set oPerson = Velocity.Persons(1)
```

To obtain the database identity of a person, enumerate the `Persons` Collection at least once and then store the `PersonID` of each `Person` object locally for future reference.

With the `Person` object, you can:

- Change the `FirstName`, `LastName`, `MiddleName`, `Suffix`, and so on for an existing person.
- Change user defined fields 1-36 for an existing person.
- Change the primary photo and signature for an existing person.

PersonCollection

Instancing: MultiUse
 MTS Mode: Not an MTS object
 Persistence: Not Persistable

Properties

UserDefinedFieldSetups [Get] - Public (Datatype: UserDefinedFieldSetupCollection)

returns a collection of `UserDefinedFieldSetup` objects.

CredentialTemplates [Get] - Public (Datatype: CredentialTemplateCollection)

returns a collection of credential templates

ImagePlaceHolders [Get] - Public (Datatype: ImagePlaceholderCollection)

returns a collection of image place holders

Count [Get] - Public (Return Datatype: Long)

No Parameters

Filter [Let] - Public: A SQL compatible where clause to limit the persons returned.

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	NewFilter	String

Filter [Get] - Public (Return Datatype: String)

No Parameters

Item [Get] - Public (Return Datatype: Person) Default Property

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	PersonId	Long

Methods

Remove - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	PersonId	Long

AddNew - Public (Return Datatype: Person)

No Parameters

Events

None

Remarks

With `PersonCollection`, you can:

- Traverse all the persons.
- Get the number of persons in the system using the `Count` property.
- Obtain a `Person` object using the `Item` property.
- Create a person.
- Remove a person.

To add a new person:

1. Obtain a `PersonCollection` object.
2. Call the `AddNew` method of the `PersonCollection` object to return a new `Person` object.
3. Set the properties of the new `Person` object.
4. Call the `Update` method of the `Person` object

```
Dim Velocity as New VMO.VelocityServer
Dim oPerson as VMO.Person
```

```
Velocity.Connect "ENG-SQL-1"
Set oPerson = Velocity.Persons.AddNew
'Set the person properties
oPerson.Update
Set oPerson = Nothing
Velocity.Disconnect
```

```
Set Velocity = Nothing
```

To remove a person:

Use the `Remove` method of the `PersonCollection` object and pass in the `PersonID` to remove the targeted person.

```
Velocity.Persons.Remove(1)
```

When using the `Item` or `Remove` method, the `PersonCollection` supports member identification only using its database `Identity`. For example:

```
Set oPerson = Velocity.Persons(1)
```

To obtain the database identity of a person, you need to enumerate the `Persons` Collection at least once and then store the `PersonID` of each `Person` object Locally for future reference.



Creating or removing a person by using the `PersonCollection` requires the operator to have certain privilege.

To traverse all persons:

```
Dim VMO As New VMO.VelocityServer           ' Declare a VMO object
Dim Person As Person                       ' Declare a Person object

' Establish a connection to the VMO SQL database
VMO.Connect "ENG-SQL-1"

' We can traverse the entire database if necessary ( browse )
' this is VERY expensive on the CPU time
For Each Person In VMO.Persons

    Debug.Print Person.PersonID
    Debug.Print Person.FirstName
    Debug.Print Person.LastName

Next

' Gracefully close the database connection
VMO.Disconnect

' Destroy our local variables
Set VMO = Nothing
Set Person = Nothing

End Sub
```

To view a specific person:

```
Dim VMO As New VMO.VelocityServer           ' Declare a VMO object
Dim Person As Person                       ' Declare a Person object

Const KEITH_MILLESON As Integer = 3

Picture1.Cls
Picture2.Cls

' Establish a connection to the VMO SQL database
VMO.Connect "ENG-SQL-1"

' Tell the object we want to retrieve a specific Person record.
' In this case ID=3
Set Person = VMO.Persons(KEITH_MILLESON)

' Display the result in the debug window
Debug.Print Person.PersonID
Debug.Print Person.FirstName
Debug.Print Person.LastName

' Display the photo's in picture boxes on the form
Picture1.Picture = Person.Photo
Picture2.Picture = Person.Signature

' Gracefully close the database connection
VMO.Disconnect

' Destroy our local variables
Set VMO = Nothing
Set Person = Nothing

End Sub
```

PersonImage

Instancing: Public Not Creatable

MTS Mode: Not an MTS object

Persistence: Not Persistable

Properties

PersonId [Get] - Public (Return Datatype: Long) Read-only

No Parameters

PlaceholderID [Get] - Public (Return Datatype: Long)

No Parameters

PlaceholderID [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vNewValue	Long

Description [Get] - Public (Return Datatype: String)

No Parameters

Description [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vNewValue	String

Image [Get] - Public (Return Datatype: StdPicture)

No Parameters

Image [Set] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	StdPicture

Methods

Update - Public

No Parameters

Events

None

Remarks

With the `PersonImage` object, you can:

- View the `Description`, `PlaceholderID` and `Image` properties of a `PersonImage`.
- Change the `Description`, `PlaceholderID` and `Image` properties of a `PersonImage`.

To change an additional image for a person:

1. Use the `Item` method of the `PersonImageCollection` object to obtain an existing `PersonImage` object, for example:

```
Set oImage = Velocity.Persons(1).Images(1)
```
2. Set the properties of the `PersonImage` object.
3. Call the `Update` method of the `PersonImage` object:

```
Dim oImage as VMO.PersonImage
Set oImage = Velocity.Persons(1).Images(1)
'Set the oUDF properties
oImage .Update
```



Changing a `PersonImage` requires the operator to have certain privilege.

PersonImageCollection

Instancing: Public Not Creatable

MTS Mode: Not an MTS object

Persistence: Not Persistable

Properties

PersonId [Get] - Public (Return Datatype: Long) Read-only

No Parameters

Count [Get] - Public (Return Datatype: Long)

No Parameters

Item [Get] - Public (Return Datatype: PersonImage)

Optional	ByVal/ByRef	Variable	Datatype
		PlaceholderID	Long

Methods

AddNew - Public (Return Datatype: PersonImage)

No Parameters

Remove - Public

Optional	ByVal/ByRef	Variable	Datatype
		PlaceholderID	Long

Events

None

Remarks

With `PersonImageCollection`, you can:

- Get the number of additional images of a person using the `Count` property.
- Obtain an additional image at a person's image place holder using the `Item` property.

- Create an additional image at a person's image place holder.
- Delete an additional image at a person's image place holder.

To create an additional Image for a person:

1. Obtain a `PersonImageCollection` object of a person.
2. Call the `AddNew` method of the `PersonImageCollection` object to return a new `PersonImage` object.
3. Set the properties of the new `PersonImage` object such as `PlaceholderID`, `Description` and `Image`. `PlaceholderID` must be an existing `ImagePlaceholder`.
4. Call the `Update` method of the `PersonImage` object in this way:

```
Dim oImage as VMO.PersonImage
Set oImage = Velocity.Persons(1).Images.AddNew
'Set the PersonImage properties
oImage.Update
```

To delete an additional Image for a person:

1. Use the `Remove` method of the `PersonImageCollection` object and pass the `PlaceholderID` to remove the targeted additional image at a specific `Image` place holder.

```
Velocity.Persons(1).Images.Remove(1)
```

2. When using the `Item` or `Remove` method, the `PersonImageCollection` collection supports member identification only using the `Image` place holder database `Identity`. For example:

```
Set oImage = Velocity.Persons(1).Images(1)
```

Note that (1) in the preceding example is the `ImagePlaceholderID` for an already defined `Image Place Holder`.

To obtain the database identity of an image place holder, you must enumerate the `ImagePlaceHolders` collection at least once, then store the `ImagePlaceholderID` of each `ImagePlaceholder` object locally for future reference.



Creating or removing a person by using the `CredentialCollection` requires the operator to have certain privilege.

Selection

Instanting: Public Not Creatable
MTS Mode: Not an MTS object
Persistence: Not Persistable

Properties

Value [Let] - Public

This property is used when assigning a value to the property on the left side of an assignment. The basic syntax is:

```
X.Value = 5
```

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	String

Value [Get] - Public (Return Datatype: String)

This property is used when retrieving value of a property, on the right side of an assignment. The basic syntax is:

```
Debug.Print X.Value
```

No Parameters

SelectionCollection

Instanting: Public Not Creatable
MTS Mode: Not an MTS object
Persistence: Not Persistable

Properties

Count [Get] - Public (Return Datatype: Long)

No Parameters

Item [Get] - Public (Return Datatype: Selection) Default property

Optional	ByVal/ByRef	Variable	Datatype
		vntIndexKey	Variant

Methods

None

Events

None

UserDefinedField

Instancing: Public Not Creatable

MTS Mode: Not an MTS object

Persistence: Not Persistable

Properties

Fieldno [Let] - Friend

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	Integer

Fieldno [Get] - Public (Return Datatype: Integer)

No Parameters

PersonId [Get] - Public (Return Datatype: Long)

No Parameters

Value [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	String

Value [Get] - Public (Return Datatype: String)

No Parameters

Methods

Update - Public

Optional	ByVal/ByRef	Variable	Datatype
		Value	String

Events

None

Remarks

With the `UserDefinedField` object, you can assign or get the value of a `UserDefinedField`.

To change a user-defined field:

1. Obtain a `UserDefinedField` object using the `Item` property of the `UserDefinedFieldCollection` object.
2. Set the properties of the `UserDefinedField` object.
3. Call the `Update` method of the `UserDefinedField` object

```
Dim oUDF as VMO.UserDefinedField
Set oUDF = Velocity.Persons(1).UserDefinedFields(1)
'Set the oUDF properties
oUDF.Value = 'new value'
oUDF.Update
```



Changing a `UserDefinedField` requires the operator to have certain privilege.

UserDefinedFieldCollection

Instancing: Public Not Creatable

MTS Mode: Not an MTS object

Persistence: Not Persistable

Properties

Count [Get] - Public (Return Datatype: Long)

No Parameters

Item [Get] - Public (Return Datatype: UserDefinedField)

Optional	ByVal/ByRef	Variable	Datatype
		bytFieldNo	Long

PersonId [Get] - Public (Return Datatype: Long)

No Parameters

Methods

None

Events

None

Remarks

With `UserDefinedFieldCollection`, you can:

- Obtain a `UserDefinedField` using the `Item` property and pass in the Field Number (`FieldNo`) starting at 1.
- Get the number of user defined fields by the `Count` property.

UserDefinedFieldSetup

Instancing: Public Not Creatable

MTS Mode: Not an MTS object

Persistence: Not Persistable

Properties

Caption [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	String

Caption [Get] - Public (Return Datatype: String)

No Parameters

Fieldno [Get] - Public (Return Datatype: Integer)

No Parameters

FieldType [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	vFieldTypes

Public Enum type vFieldTypes

vTextBox = -1
 vDropDown = 0
 vDropDownAsc = 3
 vDropDownDesc = 4
 vDropDownList = 2
 vDropDownListAsc = 5
 vDropDownListDesc = 6

FieldType [Get] - Public (Return Datatype: vFieldTypes)

No Parameters

Required [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vData	Boolean

Required [Get] - Public (Return Datatype: Boolean)

No Parameters

Methods

Update - Public

Optional	ByVal/ByRef	Variable	Datatype
Optional	ByVal	Value	String("")

If you set the user-defined field type to any of the drop-down list types, pass a carriage return delimited string for the default pick list in the combo box.

Events

None

Remarks

With the `UserDefinedFieldSetup` object, you can assign or get the caption, `FieldType` and `Required` properties of a `UserDefinedFieldSetup` object.

To change a user defined field setup:

1. Obtain a `UserDefinedFieldSetup` object using the `Item` property of the `UserDefinedFieldSetupCollection` object.
2. Set the properties of the `UserDefinedField` object.
3. Call the `Update` method of the `UserDefinedField` object in this way:

```
Dim oUDFSetup as VMO.UserDefinedFieldSetup
Set oUDFSetup =
    Velocity.Persons.UserDefinedFieldSetups(1)
'Set the oUDFSetup properties
oUDFSetup.Caption = "Department"
oUDFSetup.FieldType = vDropDownDesc
oUDFSetup.Required = True
oUDFSetup .Update
```



Changing a `UserDefinedFieldSetup` requires the operator to have certain privilege.

UserDefinedFieldSetupCollection

Instantiating: Public Not Creatable

MTS Mode: Not an MTS object

Persistence: Not Persistable

Properties

Count [Get] - Public (Return Datatype: Long)

No Parameters

Item [Get] - Public (Return Datatype: UserDefinedFieldSetup)

Optional	ByVal/ByRef	Variable	Datatype
		bytFieldNo	Long

Methods

ChangeNumberOfUDFS - Public

Optional	ByVal/ByRef	Variable	Datatype
		n	Integer

Events

None

Remarks

With `UserDefinedFieldSetupCollection`, you can:

- Obtain a `UserDefinedFieldSetup` object using the `Item` property and passing in the Field Number (`FieldNo`), starting at 1.
- Get the number of user-defined fields supported by the `Count` property.
- Change the number of user defined fields supported by the `ChangeNumberOfUDFs` method.

A sample of code employing this call is:

```
Dim oColUDFSetup as V'MO.UserDefinedFieldSetupCollection
Set oColUDFSetup =
    Velocity.Persons.UserDefinedFieldSetups
oColUDFSetup.ChangeNumberOfUDFs(72)
```

VelocityServer

Instancing: MultiUse
 MTS Mode: Not an MTS object
 Persistence: Not Persistable

Properties

EnableCustomize [Get] - Public (Return Datatype: Boolean)

No Parameters. Determines if Customized string values are used.
 Default value is **False**.

EnableCustomize [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vNewValue	Boolean

Persons [Get] - Public (Return Datatype: PersonCollection) Read-only

No Parameters

Silent [Get] - Public (Return Datatype: Boolean)

No Parameters. Determines whether an explicit error condition is raised. Default value is **False**.

Silent [Let] - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	vNewValue	Boolean

Methods

Connect - Public

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	RemoteServerName (The server on which Velocity Domain Service is running.)	String
	ByVal	RemoteServerPort (The remote port which the server is set to be listening on.)	String
	ByVal	SingleWorkstationInstall (Whether this machine is running this part of the single workstation installation. If yes, pass True ; if no, pass False .)	Boolean
	ByVal	SQLServerName (SQL Server Name)	String
	ByVal	DatabaseName (Database name)	String

Call `Connect` after instantiating a `VelocityServer` object to establish connection with the Velocity Domain Service. Pass the server name to the `Connect` method.

Disconnect - Public

No Parameters. Clean this up when application shuts down.

Events

OnSuperStatus

Optional	ByVal/ByRef	Variable	Datatype
		BaseRelayStatus	vBaseRelayStatus
		ExpRelayStatus	vExpRelayStatus
		KeypadStatus	vKeypadStatus
		StandardTimeZoneStatus	vSTZStatus
		MasterTimeZoneStatus	vMTZStatus
		GrandMasterTimeZoneStatus	vGTZStatus
		BaseInputStatus	vBaseInputStatus
		ExpInputsStatus1To8	vExpInputStatus1To8
		ExpInputsStatus9To16	vExpInputStatus9To16
		MiscellaneousFlags	vMiscellaneousFlags
		BaseInputVoltage	vBaseInputVoltage
		ExpInputVoltage1To16	vExpInputVoltage1To16
		HostSignature	vHostSignature
		ExpInputStatus17To24	vExpInputStatus17To24
		ExpInputStatus25To32	vExpInputStatus25To32
		ExpInputVoltage17to32	vExpInputVoltage17To32

OnStateChange

Optional	ByVal/ByRef	Variable	Datatype
		DeviceType	Byte
		ControllerID	Long
		Status	String

OnSystemEvent

Optional	ByVal/ByRef	Variable	Datatype
		DateTime	Date
		Location	String
		EventNumber	Long
		EventDescription	String
		EventData	String

OnAlarmActive

Optional	ByVal/ByRef	Variable	Datatype
		Message	HirschMessage

OnAlarmSecure

Optional	ByVal/ByRef	Variable	Datatype
		Message	HirschMessage

OnXBoxEvent

Optional	ByVal/ByRef	Variable	Datatype
	ByRef	VelocityEvent	XBoxEvent

OnInternalEvent

Optional	ByVal/ByRef	Variable	Datatype
	ByRef	VelocityEvent	InternalEvent

OnExternalEvent

Optional	ByVal/ByRef	Variable	Datatype
	ByRef	VelocityEvent	ExternalEvent

OnMiscEvent

Optional	ByVal/ByRef	Variable	Datatype
	ByRef	VelocityEvent	MiscEvent

OnSystemTransaction

Optional	ByVal/ByRef	Variable	Datatype
	ByRef	VelocityEvent	TransactionEvent

OnPortEvent

Optional	ByVal/ByRef	Variable	Datatype
	ByRef	VelocityEvent	PortEvent

OnSoftwareEvent

Optional	ByVal/ByRef	Variable	Datatype
	ByRef	VelocityEvent	SoftwareEvent

OnBatchCreate

Optional	ByVal/ByRef	Variable	Datatype
		EventID	Long
		DateTime	Date
		Destination	String
		CommandSet	Long

OnBatchComplete

Optional	ByVal/ByRef	Variable	Datatype
		EventID	Long
		DateTime	Date
		Destination	String
		CommandSet	Long
		NumSubmitted	Long
		NumComplete	Long
		NumErrors	Long

OnBatchError

Optional	ByVal/ByRef	Variable	Datatype
		EventID	Long
		DateTime	Date
		Destination	String
		CommandSet	Long
		ErrDescription	String

OnBatchSubmit

Optional	ByVal/ByRef	Variable	Datatype
		EventID	Long
		DateTime	Date
		Destination	String
		CommandSet	Long

OnBatchStatus

Optional	ByVal/ByRef	Variable	Datatype
		EventID	Long
		DateTime	Date
		Destination	String
		CommandSet	Long
		NumSubmitted	Long
		NumComplete	Long
		NumErrors	Long
		Status	Long

OnAlarmOverride

Optional	ByVal/ByRef	Variable	Datatype
		Message	HirschMessage

OnCommandError

Optional	ByVal/ByRef	Variable	Datatype
		Message	HirschMessage

OnDiagnostic

Optional	ByVal/ByRef	Variable	Datatype
		LineType	String

OnServerMessage

Optional	ByVal/ByRef	Variable	Datatype
		LineType	String

OnDisconnect

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	ForceShutdown	Boolean

OnConnect

No Parameters

OnADOError

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	ADOErrNumber	Long
	ByVal	ADODescr	String

OnACK

Optional	ByVal/ByRef	Variable	Datatype
		AlarmID	Long
		TimeStamp	Date
		Operator	String
		Workstation	String
		ACKType	AlarmTypes

OnCLR

Optional	ByVal/ByRef	Variable	Datatype
		AlarmID	Long
		TimeStamp	Date
		Operator	String
		Workstation	String
		CLRType	AlarmTypes

OnCCTVPacket

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	PacketLength	Long
	ByRef	Packet()	Byte

OnCCTVMessage

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	MsgID	Long
	ByVal	P1	Long
	ByVal	P2	Long
	ByVal	P3	Long
	ByVal	MsgText	String

OnSyncData

Optional	ByVal/ByRef	Variable	Datatype
	ByVal	sType	String
	ByVal	lData	Long
	ByVal	vData	Variant
	ByVal	Sender	String
	ByVal	Recipient	String

Remarks

This sample code creates a `VelocityServer` object:

```
Dim Velocity as New VMO.VelocityServer
Velocity.Connect "ENG-SQL-1"
'Do something
Velocity.Disconnect
Set Velocity = Nothing
```

VB Calls Sample

This example demonstrates how VMO VB Calls can be used to create a working program.

```
Public Velocity As New VMO.VelocityServer

Private Function Connect() As Boolean
    On Error GoTo ErrorHandler
    Dim strDBName As String
    strDBName = DBName.Text

    Velocity.Connect strDBName ' this is your DB SQL
    Server Name
    Connect = True
    Exit Function

ErrorHandler:
    Connect = False
    MsgBox "Error occurred in Connect " & Erl & vbCrLf &
    Err.Source & vbCrLf & Err.Number & " - " &
    Err.Description, vbCritical, "Connect"

End Function

Private Function AddPerson()
    On Error GoTo ErrorHandler

    strProgress1 = "Adding a person - Last Name = " &
    PersonLastName & vbCrLf
    WriteReport0 (strProgress1)
    Set Person = Velocity.Persons.AddNew 'API AddNew
    Function
        ' Define the Person - Set person Properties
        Person.FirstName = PersonFirstName
        Person.LastName = PersonLastName
        Person.MiddleName = PersonMiddleName
        Person.Title = PersonTitle
        Person.Suffix = PersonSuffix

        ' Do the insert
        Person.Update

        ' Get the host id from velocity
        PersonID = Person.PersonID

        strProgress1 = "Successfully added a person -
        PersonID = " & PersonID & vbCrLf
        WriteReport0 (strProgress1)

        Set Person = Nothing

    Exit Function

ErrorHandler:
```

```
        MsgBox "Error occurred in ParsePersonRecord " & Err
        & vbCrLf & Err.Source & vbCrLf & Err.Number & " - " &
        Err.Description, vbCritical, "ParsePersonRecord"

End Function
Private Function UpdPerson()

    On Error GoTo ErrorHandler

    strProgress1 = "Updating a person - Last Name = " &
    PersonLastName & vbCrLf
    WriteReport0 (strProgress1)
    'Set the person you want to update - personid =
    unique Velocity number
    Set Person = Velocity.Persons(PersonID)

    ' Define the Person
    Person.FirstName = PersonFirstName
    Person.LastName = PersonLastName
    Person.MiddleName = PersonMiddleName
    Person.Title = PersonTitle
    Person.Suffix = PersonSuffix

    'do the update
    Person.Update

    PersonID = Person.PersonID

    strProgress1 = "Successfully Updated a person -
    PersonID = " & PersonID & vbCrLf
    WriteReport0 (strProgress1)

    Set Person = Nothing

    Exit Function

ErrorHandler:

    MsgBox "Error occurred in UpdPerson" & Err & vbCrLf
    & Err.Source & vbCrLf & Err.Number & " - " &
    Err.Description, vbCritical, "UpdPerson"

End Function
Private Function DelPerson()

    On Error GoTo ErrorHandler

    strProgress1 = "Removing a person - Last Name = " &
    PersonLastName & vbCrLf
    WriteReport0 (strProgress1)
    Set Person = Velocity.Persons(PersonID)

    Velocity.Persons.Remove (PersonID)

    strProgress1 = "Successfully Removed a person -
    PersonID = " & PersonID & vbCrLf
    WriteReport0 (strProgress1)
```

```
        PersonID = 0
        Set Person = Nothing
        Exit Function

ErrorHandler:
        MsgBox "Error occurred in DelPerson" & Erl & vbCrLf
        & Err.Source & vbCrLf & Err.Number & " - " &
        Err.Description, vbCritical, "DelPerson"

End Function

Private Function Disconnect() As Boolean

        On Error GoTo ErrorHandler

        Velocity.Disconnect
        Set Velocity = Nothing

        Disconnect = True

        Exit Function

ErrorHandler:

        Disconnect = False
        MsgBox "Error occurred in Disconnect " & Erl &
        vbCrLf & Err.Source & vbCrLf & Err.Number & " - " &
        Err.Description, vbCritical, "Disconnect"

End Function
```

XML Interoperability Services



XML Overview

Velocity has adopted an XML exchange format to support the emerging demand for interoperability between disparate hardware and software vendors using XML. Velocity XML can be used to trigger hardware from one platform to another as well as serve as an SDK for the Pocket PC platform for wireless guard applications.

The XML kit includes:

- Sample XML code, *TestXML.exe* and *XMLEar.exe*. *XMLEar* is a program designed to listen at the designated port for XML code sent from Velocity's XML Writer.
- Sample XML projects using ASP and T-SQL in addition to several other languages.

Velocity XML resides within the Velocity Web Server. Rather than require separate ports for HTTP and XML, XML requests are expected to be issued in HTTP format using a POST to the Velocity Web Server. Additional XML functionality is provided via the service extension platform for pushing alarms & events to outside consumers.

Remote Procedure Calls

Using an attribute, the document is tagged as RPC and provided with a procedure to call. An RPC may or may not return a result. An example of this is shown below:

```
<?xml version="1.0" encoding="us-ascii" ?>
<Velocity.XML RPC="CommandSet.Execute">
  <Authentication>
    <UserName>UserName</UserName>
    <Password>Password</Password>
    <Domain>Domain</Domain>
    <Workstation>Workstation</Workstation>
  </Authentication>
  <Parameters>
    <Identity>40</Identity>
  </Parameters>
</Velocity.XML>
```

Acknowledging and Clearing Alarms

When a third-party application first executes, it can request a list of active alarms in the system using `REPORT="List.Alarms"`.

From that list, the application will obtain enough information to later make RPC posts to acknowledge and/or clear the alarm.

```
<?xml version="1.0" encoding="us-ascii" ?>
<Velocity.XML REPORT="List.Alarms">
  <Authentication>
    <UserName>UserName</UserName>
    <Password>Password</Password>
    <Domain>Domain</Domain>
    <Workstation>Workstation</Workstation>
  </Authentication>
</Velocity.XML>
```

Acknowledge

An example of Acknowledge is shown in the following example:

```
<?xml version="1.0" encoding="us-ascii" ?>
<Velocity.XML RPC="Alarm.Acknowledge">
  <Authentication>
    <UserName>UserName</UserName>
    <Password>Password</Password>
    <Domain>Domain</Domain>
    <Workstation>Workstation</Workstation>
  </Authentication>
  <Parameters>
    <Identity>1049</Identity>
  </Parameters>
</Velocity.XML>
```

Clear

An example of Clear is shown below:

```
<?xml version="1.0" encoding="us-ascii" ?>
<Velocity.XML RPC="Alarm.Clear">
  <Authentication>
    <UserName>UserName</UserName>
    <Password>Password</Password>
    <Domain>Domain</Domain>
    <Workstation>Workstation</Workstation>
  </Authentication>
  <Parameters>
    <Identity>1048</Identity>
  </Parameters>
</Velocity.XML>
```

SQL Statement Execution (Insert, Update, Delete, Query, etc.)

Using an attribute, the document is tagged as QUERY and returns a result in the form of XML. It is the author's responsibility to apply a style sheet if a presentation layer exists.

The result can be cast into an *ADO Stream* and *Recordset*.

An example of such an statement is shown in the following code:

```
<?xml version="1.0" encoding="us-ascii" ?>
<Velocity.XML QUERY="SELECT DoorID, DoorName FROM Doors">
  <Authentication>
    <UserName>UserName</UserName>
    <Password>Password</Password>
    <Domain>Domain</Domain>
    <Workstation>Workstation</Workstation>
  </Authentication>
</Velocity.XML>
```

Channel Responses

When executing documents against the web server there are four types of responses:

1. **Response** – The document was accepted without error and was executed. No results are returned.
2. **EXCEPTION** – The document was received and an error was generated while parsing the document or during the execution of the document contents. An XML exception will be returned to the consumer with an error number and description of the error.
3. **QUERY** – Results to QUERY posts are in XML form that can be used as-is or transformed into a *Microsoft ADODB Stream* and then used as a fire-hose *recordset* object as if queried directly from the SQL Server.
4. **REPORT** – Results from List requests are XML documents that can be used to discover identity information for making RPC's.

Error Handling

Exceptions are handled as a response to a POST to the service. Errors are returned as an XML-formatted exception with the error number and description.

An example of error-handling is shown in the following sample:

```
<?xml version="1.0" encoding="us-ascii" ?>
<Velocity.XML Interop="Exception">
  <Parameters>
    <ErrorNumber/>
    <Description/>
  </Parameters>
</Velocity.XML>
```

Discovery

To ease XML development for remote procedure calls, Velocity XML publishes a list of reports that can be requested to enable discovery of the system without knowing the SQL schema.

The following POST returns an XML document containing all published reports available to outside sources:

```
<?xml version="1.0" encoding="us-ascii" ?>
<Velocity.XML REPORT="List.Reports">
  <Authentication>
    <UserName>UserName</UserName>
    <Password>Password</Password>
    <Domain>Domain</Domain>
    <Workstation>Workstation</Workstation>
  </Authentication>
</Velocity.XML>
```

Each of those reports can then be POST'd to the service for further discovery.

In this next example, the following POST returns a document containing a list of all Command Sets in the system:

```
<?xml version="1.0" encoding="us-ascii" ?>
<Velocity.XML REPORT="List.CommandSets">
  <Authentication>
    <UserName>UserName</UserName>
    <Password>Password</Password>
    <Domain>Domain</Domain>
    <Workstation>Workstation</Workstation>
  </Authentication>
</Velocity.XML>
```

The resulting list can then be used to prepare RPC requests providing the parameter information needed.

Default Supported Lists

These are the currently supported default lists:

- List.CommandSets
- List.Doors
- List.Relays
- List.Expansion Relays
- List.Alarms

These lists can be expanded or modified to meet the end user's needs. Lists are stored in *dbo.Velocity.XMLReports*. The ReportLayout column defines what data is returned when the list is executed.

All reports found in this table are returned to the call `REPORT="List.Reports"`.

This can be used by a third party to write custom reports generated from the resulting XML document. Most current reporting engines accept XML input.

Many other extensions can be created by either ad hoc or scheduled interval REPORT queries such as EDI and middleware translations.

Receiving Alarms & Events

In order to receive alarms and events, a third-party application must open a second TCP/IP channel that connects to the Velocity XML server extension.

Once the application's credential has been validated using `Interop="Logon"`, the application starts receiving new alarms and events from that point. We recommend that the application post an `REPORT="List.Alarms"` to discover active alarms that have not yet been acknowledged and/or cleared.

As alarms are cleared by another XML channel or standard Velocity client, the application receives an ACK or CLR event, allowing it to update its display for alarms that are no longer active.

Alarm & Event Messaging

`Interop`, `RPC`, and `Exceptions` are managed through the Velocity Web Server, which is an integral part of the Security Domain. For performance reasons, alarms and events are not part of the web service architecture. Alarm and event management is handled on a separate TCP/IP port as a service extension, *Velocity XML Service Extension*.

While a third-party application interacts with the Security Domain over HTTP, it receives alarms and events from the XML Server Extension in XML format.

Non-Alarm Event

This sample code elicits a non-alarm event:

```
<?xml version="1.0" encoding="us-ascii" ?>
<Velocity.XML Interop="Event">
  <Parameters>
    <HostTime/>
    <ControllerTime/>
    <EventID/>
    <Description/>
    <Address/>
  </Parameters>
</Velocity.XML>
```

Alarm

An example of an alarm call is shown here:

```
<?xml version="1.0" encoding="us-ascii" ?>
<Velocity.XML Interop="Alarm">
  <Parameters>
    <HostTime/>
    <ControllerTime/>
    <AlarmIdentity/>
    <EventID/>
    <Level/>
    <Description/>
    <Address/>
  </Parameters>
</Velocity.XML>
```

Supported Remote Procedure Calls

The supported RPCs are shown in the list below:

- 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
- 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
- Alarm.Acknowledge
- Alarm.Clear
- Post.Alarm (future implementation)
- Post.Event (future implementation)

Communicating with Velocity Web Server

XML documents communicate with the Velocity Web Server through the execution of RPCs, as described in the following topic.

Executing Remote Procedure Calls

XML documents are submitted to the Velocity Web Server over HTTP. Depending on the development tools available to the third-party developer, this can be easy.

The following code fragment is an example of how a developer can submit a document over HTTP using Microsoft Visual Basic:

```
Dim FSO As New FileSystemObject
Dim oFile As File
Dim TS As TextStream

' This test will send an XML LOGON request
Set oFile = FSO.GetFile(App.Path & "\Interop.Logon.xml")
Set TS = oFile.OpenAsTextStream(ForReading)

InetControl.Execute "http://MyServer", "POST", TS.ReadAll, "Content-Type: text/xml"

Set FSO = Nothing
Set oFile = Nothing
Set TS = Nothing
```

Before doing this, add a project reference for *Microsoft Scripting Runtime (scrrun.dll)* and *Microsoft Internet Controls (shdocvw.ocx)*. This will provide a quick way to handle file I/O and HTTP without writing additional source code.

The response can be handled on the Internet Control's `StateChanged` event as follows:

```
Private Sub InetControl_StateChange(Object Val State As Integer)

    Dim Response As Variant

    Select Case State

        Case icResponseCompleted

            Response = Inet1.GetChunk(1024, icString)

            Do While LenB(Response) > 0
                Text1.Text = Text1.Text & Response
                Response = Inet1.GetChunk(1024, icString)
            Loop

        End Select

    End Sub
```

This example displays the response in a text box.

