Overview

The Corel Visual CADD API provides an list of functions for users to implement in their applications. By combining and utilizing these different functions, powerful, robust applications can be written. Corel Visual CADD is written on a completely open architecture API structure. What this means to you is that you can create an almost limitless number of custom tools and applications that will work with Corel Visual CADD to enhance your drawing abilities.
These applications can range from simple add-on tools, such as the new Midline tool that draws from the center of a line to its endpoint, to a completely customized interface with limited tool sets, such as a “red liner” application. The Corel Visual CADD API can be broken down into several categories based on functionality and purpose. The basic function categories provided with this API Reference include:

- Entity Creation Functions
- Entity Editing Functions
- Environment Settings Functions
- User Data Functions
- Parsing Functions
- Calculation Functions
- Tool Functions
- Dialog Functions
- Query Functions
- 3D Functions

These categories briefly discuss issues related to the Corel Visual CADD programming environment and some of the difficulties that you may encounter. In addition, this reference provides basic information on what is required to develop an application using Corel Visual CADD, such as how to create new tools and how to create custom interfaces. For more specific information, please refer to the code examples provided on the Numera Internet Home Page at http://www.numera.com or contact Numera about joining the Third Party Developer’s Program.

Entity Creation Functions

The core of any CAD system is the ability to create and collect graphical entities with pre-defined properties to form a drawing. The Corel Visual CADD API provides extensive commands for creating and adding entities to the drawing database. These commands can add entities directly through the code or as a result of user interaction within an interface.

File translation, field data collection and database interaction are examples of situations when entities would be added directly through code. Since the entire Corel Visual CADD database is accessible through the API, custom file translation routines can easily be implemented. These translation routines allow a developer to store graphical entity information in a format required by the developer. Graphical output for field data and database information can be created directly from that information. For example, a file that contains point values from a COGO data reader or a spreadsheet that generates point values from an internal macro operation can be used to create graphical output in a drawing. For more information, please see User Data.

Entities can also be added through user interaction and input. The Corel Visual CADD API provides several methods for capturing and processing user interaction. By utilizing a tool command, an application can allow users to add entities to the drawing database while Corel Visual CADD handles all messaging events ,such as key presses, mouse down or mouse move events. In addition, the Visual CADD API offers powerful event handling routines to create custom tools not directly available through the Corel Visual CADD interface. These applications are based on a set of user tool operations that allow the developer to control every aspect of a tool. The external application processes the Window events and sends the appropriate response back to Corel Visual CADD. These commands can be used to generate powerful processing and rubber banding techniques for creating virtually any type of tool.

Note that no matter what method is used to create an entity, new entities are always appended to the end of the drawing database. By utilizing the various parsing routines of the API, the newly created entity can be accessed quickly and efficiently.

Entity Editing Functions

The Corel Visual CADD API provides full editing capabilities for all existing entities. Entities can be modified by applying the currently existing properties (such as line type, line width, color and layer information) to an existing entity or by recreating the entity based on entirely new property values. These two methods offer a great deal of flexibility in modifying the entity.

The easiest way to change the properties of an existing entity is to apply the current property settings to the desired entity. This can be done through the Corel Visual CADD API by simply changing the desired properties with the various settings commands and then using the VCApplySettingsToCurrentEntity(). This, in effect, changes the properties of the entity in the database just as if a user had used the “change” command in Corel Visual CADD.

A second method allows an existing entity to be duplicated to the end of the database using the VCDuplicate() routine, which recreates the entity using all of the current entity properties . This method offers an advantage over the VCApplySettingsToCurrentEntity() routine in that locating the newly-modified entity can be done quickly and easily by locating the last entity in the database. In addition, the duplicate method offers the ability to use the “undo” command, while merely applying the settings does not.

Environment Settings Functions

Corel Visual CADD maintains an extensive collection of user-defined environment settings. These settings range from system

settings, such as background color and cursor size, to specific default entity settings, such as line type and color. All of the settings that are available through the normal Corel Visual CADD dialogs are also available through API routines.

When creating stand-alone applications, it is very important to be aware of the current settings for the Corel Visual CADD environment. For example, when an entity is deleted in Corel Visual CADD, the entity is not really erased, but simply tagged as erased and redrawn in the background color. The entity itself remains in the Corel Visual CADD database until a “pack data” command is issued. This can present a problem in stand-alone applications where the Visual CADD background color is different than that of the window being used as a drawing world. An erased entity may be drawn in blue if the Corel Visual CADD background color is blue, but if the stand-alone drawing world has a white background, this would result in a blue entity that does not appear to be erased.

User Data Functions

User data may be attached to any drawing entity or drawing header and used for the storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of type double, float, long, short or byte. In addition to these types, a user-defined type of “chunk” may also be stored. A chunk can be declared to be any size and is simply a pointer to a memory location. The size of the chunk is passed to Corel Visual CADD so that it can retrieve the appropriate amount of data from the specified memory location.

Whenever using user data, an application must first set a user data name in order to protect private data and to ensure that different applications do not interfere with another application’s data. The user data name must be set before adding any user data to a drawing. By registering as a Numera Visual CADD 3rd Party developer, Numera will provide you with a unique user data name to be used for this purpose.

Parsing Functions

Corel Visual CADD stores all drawing information in a database accessible only to the Visual CADD engine. While other file formats typically require libraries or reverse engineering to access the data, the Corel Visual CADD database is fully accessible through the Visual CADD API. This includes all of the entity properties, special data attached to entities and environment settings.

When building a file converter for a format not directly supported by Corel Visual CADD, entities can be parsed and added to another file without manually reading in and deciphering the Visual CADD file.

Several Corel Visual CADD API functions are specifically designed to parse through the database and extract information. They are analogous to common database functions and provide the ability to move quickly to the beginning of the database (VCFirstEntity)e, to the end of the database (VCLastEntity), to the next entity in the database (VCNextEntity) or to select a specific entity by its index or handle (VCSetCurrentEntity). The Corel Visual CADD API can then be used to extract any entity information desired.

As with any database file, it is often not desirable to parse through the entire database from one entity to the next. The Corel Visual CADD API has several routines designed to limit the search criteria and to apply filters for certain entity properties. There are routines to parse only selected items (VCFirstSelected, VCNextSelected), items that are currently on screen (VCFirstOnScreen, VCNextOnScreen), Symbol Definitions (VCSetSymbolSection) and exploded entities (VCFirstEntityExpand, VCNextEntityExpand).

Calculation Functions

Several calculation routines are available through the Corel Visual CADD API. These routines are provided to help calculate some of the complicated CAD operations that an application may need. These routines range from converting distances and angles into string values (VCDistToString(), VCAngleToString()), to complex operations for computing virtual intersections and arc definition points (VCComputeIntersection(), VCGetCurrentEntityArcData()).

Tool Routines

The tool routines allow for quick access to all of the existing Corel Visual CADD tools. When called, they act just as they would if a user had clicked on the tool while running Visual CADD normally. They offer advantages over the entity creation routines in that Corel Visual CADD handles all of the messaging and creation events while allowing the end user to interact and draw normally. The disadvantages are that the developer has little control over the sequence of events of the tool or of the user interaction possibilities. Some examples of Corel Visual CADD tools include VCLine(), VCCircle2Pt() and VCEllipse().

Dialog Routines

Corel Visual CADD makes use of dialogs to display various settings and tool options. These dialogs are also available for use in external applications. When utilizing the built-in Corel Visual CADD dialogs in a stand-alone application, it is first necessary to initialize the dialogs with the VCInitDialogs() routine. Once initialized, you can call all of the normal dialog routines (such as VCPrintDlg()) and they will be displayed just as if they had been called from Corel Visual CADD.

These dialog functions also give developers access to the positioning of the tool palette and main speed bar. By using the returned screen coordinates, applications can effectively simulate a Corel Visual CADD speedbar anywhere they want to. This allows an application to take advantage of the built-in features that users are already accustomed to in the Corel Visual CADD interface.

Note that when you are done using the dialog routines, you must call the VCTerminateDialogs() routine to free up the memory used by dialogs.

Query Functions

The query functions are used to retrieve various entity property information while parsing the drawing database. These routines can be used to retrieve properties from the current entity or from the current overall property settings. These functions will return current entity properties such as color (VCGetCurrentEntityColor ()) and line type (VCGetCurrentEntityLineType()), in addition to calculated properties such as length (VCGetCurrentEntityLength()) and area (VCGetCurrentEntityArea()).

3D Functions

While the Corel Visual CADD interface is strictly two dimensional, there are 3D capabilities built into the Corel Visual CADD API. These routines allow 3D points, lines and polygons to be added to the database. Perspective views with different target and eye positions can be set to view the drawing. In addition, AutoCAD 3D drawings and blocks can be loaded directly into the Corel Visual CADD interface. Some examples of 3D routines are VCAddLine3D(),VCAddPoint3D() and VCSetProjection3D().

Development Requirements

Corel Visual CADD is open to development with any Windows programming language. The programming language needs to support calls to external libraries contained in Windows Dynamic Link Libraries (DLLs). Visual Basic for Applications, Visual Basic Standard/Professional, C/C++ , FORTRAN, PASCAL, CA-Realizer and Delphi are a few languages that support external Corel Visual CADD API routines.

This manual is designed to be as language-independent as possible. However, the exact declaration syntax for functions changes for each different language. This manual provides declarations for some of the most common programming languages, including Visual BASIC, Delphi and C/C++.

Certain methods of returning data from a few Corel Visual CADD routines are incompatible with some languages. In these cases, a new routine has been created to provide the same functionality as the original routine, but in a way that is not quite as elegant as the original routine. These new routines are nearly identical to the original call, but have the letters “BP” attached to the end of them.

Visual Basic and Delphi are a good examples of this drawback since they require parameters of a user-defined type to be called by reference. The group of commands to add entities directly to the drawing database, such as VCAddLineEnity(), require the user-defined variable type, Point2D to be passed by value. Since Visual BASIC and Delphi will not allow the parameter to be passed by value, the Corel Visual CADD API provides a companion function called VCAddLineEntityBP() which is functionally identical to VCAddLineEnity(), but accepts the Point2D parameter passed by reference.

As a final note, all sample code was written in the current version of each respective language environment. Visual CADD 1.2.X requires a 16 bit development language and Corel Visual CADD 2.0.X requires a 32 bit programming language. There is no “thunking” layer between the versions, a 32 bit language must be used to develop with v2.0.X. No API routines have been deleted form the 32 bit version, any code written on v1.2.X will function on the 32 bit Corel Visual CADD once recompiled. The DLLs were renamed between the versions and the appropriate header files should be used. Please see the Parameter Detail section for more information on the Corel Visual CADD engine.

Languages used for sample applications are:

Visual CADD 1.2.X	Corel Visual CADD 2.0.X
Microsoft Visual Basic 4.0 - 16 bit	Microsoft Visual Basic 4.0 - 32 bit
Microsoft Visual C++ 1.5.1 - 16 bit	Microsoft Visual C++ 4.0 - 32 bit
Borland C++ 4.5.1 - 16/32 bit	Borland C++ 4.5.1 - 16/32 bit
Borland Delphi 1.0 - 16 bit	

Developing Corel Visual CADD Applications

There are two basic methods for developing applications through the Corel Visual CADD API. A developer can either use the existing Corel Visual CADD interface to display and modify drawings, or they can design their own to allow an application to utilize a separate interface while accessing commands from the Corel Visual CADD engine.

Each method presents different advantages that should be considered when writing an application based on the Corel Visual CADD engine. The principle advantages for using the already existing Corel Visual CADD interface are that there will be less development time in creating an interface and that the interface will already be familiar to an existing Corel Visual CADD user base. The advantages for writing a custom interface are the availability of creating an application-specific interface with an application-specific tool set, full control over the user’s interaction with the application and the ability to control every aspect of the creation and implementation of the application.

If the external application is simply a drafting tool used specifically to enhance the functionality of Corel Visual CADD, the standard interface should be used. If you require file conversion and display capabilities to build a red-liner, for example, and do not want the user to modify the original drawing, use a custom interface with a specific tool set. Since the Corel Visual CADD interface gives full control to the user, it would be preferable to provide a custom interface and only allow access to the necessary functionality, such as loading and saving files.

- Developing Tools For Corel Visual CADD
- Creating A Custom Interface

Developing Tools For Corel Visual CADD

The easiest way to develop external applications is to utilize the Corel Visual CADD interface. Once Corel Visual CADD is running, all the necessary DLLs are loaded into memory for use by any external application. An application can start Corel Visual CADD as it is being launched or it can allow the user to start the application from within the Visual CADD interface (i.e., from the toolbar). Please note that various application errors and General Protection Faults (GPFs) will probably occur if Corel Visual CADD API calls are made to a DLL that has not been loaded into memory. See the Corel Visual CADD User Guide for details about customizing tools, buttons and menus when allowing users to launch an application directly from the Visual CADD interface.

All tools need an interface or some means of interaction with the drawing screen. This may be as simple as locating points in the drawing area to construct a compound entity, or as complex as tracking a mouse drag across the drawing screen. It is therefore necessary to establish a communication link between Corel Visual CADD and the external application in order to develop tools to be used in the Visual CADD interface.

The Windows environment has a built-in messaging system which it uses to notify applications of changes in the environment or circumstances that may require an application to respond. Corel Visual CADD has taken advantage of this built-in messaging

system and allows an external application to access what is happening within the drawing screen. This mechanism is provided by the Corel Visual CADD VCSetAlertApp() routine.

By passing this routine the handle of the window (HWND) that you want to receive the messages and a parameter stating which messages to receive, Corel Visual CADD will send event messages back to the window as the events occur. The VCClearAlertApp() routine turns off these messaging events.

Any Windows programming language that you decide to use will support some form of Windows messaging since this is the foundation on which Windows is based. All messages sent to the external application will be of exactly the same type and format as those sent by the operating system itself.

In the case of C/C++ compliant languages, message handling is easily implemented through a specific procedure called a WNDPROC which processes each message as it is sent to an application. New messages can be added or removed as needed and processed however the programmer needs them to be processed.

Unfortunately with Visual BASIC and Delphi, the programmer cannot as easily add cases to handle all of the different messages available from Corel Visual CADD since both languages contain pre-defined event handlers for each specific control or form. With a little bit of planning, however, a suitable control can be found that handles most of the messages an application might require. A simple example of this is a picture box which can handle mouse movement and click events. This will work just fine if those are the only events that need to be processed. However, the best choice is actually to use the form itself, since it provides almost every event procedure that Corel Visual CADD supports through the messaging system and requires no extra controls. This is also convenient because the form will contain all your code for each individual event triggered by a message. By using the Visual CADD API, it can also be minimized, closed or hidden based on a user's actions in Corel Visual CADD to prevent the user from seeing any interface at all.

One of the specific messages handled by VCSetAlertApp() is the key press. When a user presses a key in Corel Visual CADD, a message is sent to the registered application through VCSetAlertApp(). In the message processing code, these key press events can then be processed by using VCGetCmdStr() which returns any characters left on the command line or command string of the last command initiated. If the command string is not one needed by the external application, it can then send the key press message back to Corel Visual CADD with the VCSetCmdStr() routine, which will then simply execute the string as if the user had typed it in. In the case of Visual BASIC, code can be put on the keypress event for the form or other control whose HWND is registered with VCSetAlertApp(), and it will then execute each time a user enters a character into Corel Visual CADD.

Something else to consider when writing an external tool is the presentation of the prompts that a user sees in the status bar to guide them through the task. The prompt for a user tool can be defined using the VCSetUserTool() routine. This establishes which prompt will appear at the first stage of the user tool. The VCSetPrompt() routine allows each subsequent prompt to be explicitly set for the current user tool.

Creating A Custom Interface

When using the functionality of Corel Visual CADD within a custom interface, the Visual CADD DLLs must first be initialized and a drawing window established before any drawing can occur. The procedures for achieving these requirements vary depending on the development environment and language used.

The first requirement is to activate the Corel Visual CADD engine. A simple call to the VCInit() function will initialize the DLLs and prepare them for use. If this function is omitted and Corel Visual CADD is not running when an API call is made, you will receive various error messages and possibly severe GPFs from Windows.

After the Corel Visual CADD engine has been initialized, the main drawing area (called a "world" by the Corel Visual CADD API) must be established within the interface. This world is created by passing the handle of an existing window (the HWND) to the VCNewWorld() routine. This function notifies Corel Visual CADD that the specified HWND is to be the container of a new drawing world and sets up internal information required to handle the new world. The VCSetCurrWorld() command takes the handle of the world that was returned from VCNewWorld() as a parameter and tells Corel Visual CADD that any further database or drawing actions should take places within this world.

Depending on an application's needs, various status message areas can be set up by passing the HWND of either an edit box or a text field to the corresponding routines in Corel Visual CADD that handle the display of status messages. These areas are used to display the current message prompts (VCSetMessageHandle()), the current drawing coordinates (VCSetXYHandle), the current distance (VCSetDistanceHandle()) and the current angle (VCSetAngleHandle()). These are by no means necessary, but are an easy-to-include user interface item that significantly adds to the functionality of the application.

Once the interface is set up and the drawing world created, any incoming Windows messages need to be relayed to Corel Visual CADD in one form or another as they are received by the external application. This message transmittal can be done by either directly relaying the message to Visual CADD if the external application could not process it, or by invoking a specific routine in response to the message. For example, when the application receives a WM_PAINT message, instead of passing the message on to Corel Visual CADD, it needs to invoke the VCPaintWorld() routine in order to tell Corel Visual CADD to repaint the drawing area. If the application receives a message without a corresponding routine, the application can use VCPostMessage() or VCSendMessage() to send the exact message back to Corel Visual CADD for processing.

Any tools used in the external application must be explicitly supported in the application code; i.e., there must be a button or menu item for each command which is accessible by the user. All mouse events must also be relayed back to Corel Visual CADD if you want the default behavior to be identical to that of Visual CADD. There are routines for processing each and every mouse event, from double-clicks to mouse moves.

Finally, when the external application has completed execution, it must call VCTerminate() in order to de-allocate the memory used internally for the drawing worlds and to free up memory for other applications.

Declarations

The Corel Visual CADD API contains four basic parts in the declaration: the Corel Visual CADD API Name, the Library Location, the Parameter List, and the Return Value. The following routine will be used as an example for description:

Declare Function VCGetCurEntAtbRecCount Lib "VCMAIN32.DLL" (iError As Integer, ByVal iWhichAtb As Integer) As Integer

Corel Visual CADD API Name: The Corel Visual CADD API has been simplified by providing descriptive names for each of the routines. For example, VCGetCurEntAtbRecCount() indicates how many attributes are attached to a symbol. Other calls, such as VCSetCurrentErased(), erase the current entity from a drawing.

Library Location: The declarations for the Corel Visual CADD API are contained in a set of four library files called **VCMAIN32**, **VCTRANS32**, **VCTOOL32** and **VCDLG32**. The names of these files correspond directly to the DLL in which the routine itself is stored. Since all of these declarations are available for direct inclusion into your application, the library locations are rarely a concern to the programmer, but are provided in case you wish to include a minimal set of declarations in your application:

Note: The 16 bit versions of these DLL don't have the "32" in the DLL name.

- **VCMAIN32** contains the majority of the database routines, such as entity creation and system settings, and is a more-or-less a general purpose library.
- **VCTRANS32** contains all the file reading and writing (translation) routines. For example, a call to load an AutoCAD 3D file is represented in this library.
- **VCTOOL32** contains tool commands that are available directly through the Corel Visual CADD interface, such as 2-point lines and circles.
- **VCDLG32** contains all of the built in dialogs that show up while working in Corel Visual CADD, such as the Layer Manager and the Symbol Manager.

Parameter List: When working with the Corel Visual CADD API, it is necessary to pass information to Visual CADD about the specific information you want to set or have returned. This is reflected in the parameter list for each routine. Different routines will require different parameters. For example, in a sample declaration such as VCGetCurEntAtbRecCount, you must specify the attribute index in order to retrieve the record count.

Return Value: The return value is the end result for the routine if it is declared as a function. For example, a sample declaration like VCGetLineTypeIndex() returns the current line type property index number.. Other routines may return information that is related in some way to the parameter information being passed by the function. For example, the name of a drawing is passed back as a parameter with the VCGetDrawingName() routine, and its return value is the number of characters in that name. Remember that procedures (sometimes called subroutines) do not have a return value.

The one common ground for most of these routines (both functions and procedures) is the **iError** value. This value represents the success or failure of the function. Some calls to set properties will only return an iError value since no information is needed on return. An iError value of 0 is true or succeed, while all other values other than 0 is failed or false.

Parameter Detail

Most of the functions listed utilize a specific set of parameters which are needed by the routine in order to return the information requested. Please see the specific call for more information on the required parameters. The following parameters are discussed in more detail and apply to all of the Corel Visual CADD API routines in one way or another: iError, distances, angles, toggles, strings, user data, and special types.

iError - This is set depending on the success or failure of the function.

0 - Succeeded.
1 - Failed: Usually due to an invalid drawing world. Please see the specific routine for more detailed information.

distance - All distances are stored in the Corel Visual CADD database in inches. When retrieving or setting distance values, you need to convert them into the proper units. VCGetUnitConversionFactor() returns a multiplier that can be used to convert the values based on the current unit setting in Corel Visual CADD.

angles- All angles are stored in radians in the Corel Visual CADD drawing database. When retrieving or setting angle values, you need to convert to the appropriate display format, typically degrees.

toggle - Most of the Get/Set calls simply return a toggle state for the specified setting. The values returned are 1, indicating "on," "checked" or "true," and 0, indicating "off," "unchecked" or "false."

string - Calls to retrieve a string value also return the length of the string. Visual Basic requires fixed length strings for return values. These can then be trimmed to the returned string length. In some languages, a "Null" value can be passed into the routine in place of the string variable, allowing the call to only return the string length. The string variable can then be allocated before call the function again.

User Data - Attaches or retrieves data of the specified type for the current entity. User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data can be of the variable types double, float, long, short, string or byte. In addition to these types, a user defined type of "chunk" may also be stored. A chunk can be any size and is simply a pointer to a memory location. The size of the chunk is also passed to Corel Visual CADD so that it can retrieve the appropriate amount of data from the specified memory location.

Special Types - There are various special cases for calls which return either a double or a user-defined variable type. Visual BASIC and Delphi do not allow user defined types to be passed by value, therefore they can not call these routines. The solution is to utilize the "BP" routine which operates the same as the original routine, but accepts the user-defined data type passed by pointer (or reference).

VCAbortOperation

Version 1.2.1

Description Ends the current operation and discards all undo information.

Declaration

C/C++: extern "C" void WINAPI VCAbortOperation(short* iError);

Visual Basic: Declare Sub VCAbortOperation Lib "VCMAIN32.DLL" (iError As Integer)

Delphi: procedure VCAbortOperation(var iError: Integer); far;

Parameters No additional parameters are used with this subroutine.

Notes Corel Visual CADD provides a set of user tool functions to build and create tools not directly supported in the interface. For example, a multi-line tool that automatically hatches or fills the segments. Since this tool is not provided directly in the Corel Visual CADD interface, it must be created through code to interact with the existing commands such as snaps and undo operations. In order for the tools to respond appropriately to undo operations it should set undo and redo levels during the operation. A complex entity tool, one that adds multiple entities such as the multi-line example, can allow each individual entity or instead the entire operation to be undone with a single user undo operation. This depends on the design criteria specified for the application. The level of undo is set with the VCBeginOperation and VCEndOperation API routines. An application should set the beginning of the undo level prior to adding any entities to the drawing database and finish the tool with an end operation. In certain situations, the tool may be aborted by the user typically by pressing the <ESC> key. An application should respond appropriately by aborting the undo level to return it to the state prior to the user tool operation. VCAbortOperation will handle this for the application. When used in conjunction with VCBeginOperation and VCEndOperation, VCAbortOperation will discard all undo information complied since the last VCBeginOperation. The VCEndOperation should be used to mark the end of an undo level if the tool completes as designed, while the VCAbortOperation should be used when the tool ends unexpectedly or if the user manually aborts the tool. VCAbortOperation ensures that there is no residual undo information left.

See Also VCBeginOperation, VCEndOperation

{button ,AL(`Creating a User Tool;Modifying Existing Entities',0,`',`')} Task Guide Examples

VCAcadBlockRead

Version 1.2

Description Loads a file as an AutoCAD block.

Declaration

C/C++: extern "C" void WINAPI VCAcadBlockRead(char* pName);

Visual Basic: Declare Sub VCAcadBlockRead Lib "VCTRAN32.DLL" (ByVal pName As String)

Delphi: procedure VCAcadBlockRead(pName: PChar); far;

Parameters *pName* - path and filename for the file.

Notes The routine loads an AutoCAD file as a block, allowing Corel Visual CADD to treat the file as a native symbol. Symbols act as a collection of entities that can be inserted repeatedly in a drawing. The symbols can be inserted at different locations with different rotations and scales while maintaining a unique identity separate from the objects composing the definition. When working with AutoCAD file types it is necessary to provide settings for certain conversion criteria. The criteria can be either set through code or as a result of user input to the application. The AutoCAD conversion criteria include base units, color translation, X-Ref conversion and font mapping. Corel Visual CADD provides a dialog for a user to edit these settings for conversion operations. These settings may or may not correspond to those required by the application. In situations where the application needs to control these settings the calls VCGetAcadImportUnit, VCGetPreserveAcadColorNums and VCGetKeepAcadFontName can be used to set the desired values.

See Also VCAcadRead, VCAcadReadWith3D, VCAcadWriteDWG, VCAcadWriteDXF, VCGetAcadImportUnit, VCGetPreserveAcadColorNums, VCGetKeepAcadFontName

{button ,AL(`Loading a Symbol;Modifying a Symbol Definition;Parsing a Symbol Definition;Placing a Symbol;Symbol Operations',0,`',`')} Task Guide Examples

VCAcadRead

Version 1.2

Description Loads an AutoCAD file into the current drawing world.

Declaration

C/C++: extern "C" void WINAPI VCAcadRead(char* pN);

Visual Basic: Declare Sub VCAcadRead Lib "VCTRAN32.DLL" (ByVal pN As String)

Delphi: procedure VCAcadRead(pN: PChar); far;

Parameters *pName* - path and filename for the file.

Notes VCAcadRead allows an AutoCAD file to be loaded and converted to a Corel Visual CADD drawing in the current drawing session. VCAcadRead strips all 3D entity information from the drawing while VCAcadReadWith3D allows a complete 3D file to be interpreted. VCAcadRead is a specific load routine to work with AutoCAD files. An error will occur if attempting to load files other than *.DWG files. In situations where other vector drawing formats such *.VCD, *.GCD or *.DXF will also be used the routine VCLoadDrawing should be implemented which will load all these vector file types. When working with AutoCAD file types it is necessary to provide settings for certain conversion criteria. The criteria can be either set through code or as a result of user input to the application. The AutoCAD conversion criteria include base units, color translation, X-Ref conversion and font mapping. Corel Visual CADD provides a dialog for a user to edit these settings for conversion operations. These settings may or may not correspond to those required by the application. In situations where the application needs to control these settings the calls VCGetAcadImportUnit, VCGetPreserveAcadColorNums and VCGetKeepAcadFontName can be used to set the desired values.

See Also VCAcadBlockRead, VCAcadReadWith3D, VCAcadWriteDWG, VCAcadWriteDXF, VCGetAcadImportUnit, VCLoadDrawing, VCAcadBlockRead3D, VCGetAcadImportUnit, VCGetPreserveAcadColorNums, VCGetKeepAcadFontName

VCAcadReadWith3D

Version 1.2

Description Loads a 3D AutoCAD file into the current drawing world.

Declaration

C/C++: extern "C" void WINAPI VCAcadReadWith3D(char* pName);

Visual Basic: Declare Sub VCAcadReadWith3D Lib "VCTRAN32.DLL" (ByVal pName As String)

Delphi: procedure VCAcadReadWith3D(pName: PChar); far;

Parameters *pName* - path and filename for the file.

Notes VCAcadRead3D allows an AutoCAD file to be loaded and converted to a Corel Visual CADD drawing in the current drawing session. VCAcadReadWith3D allows a complete 3D file to be interpreted into Corel Visual CADD, while VCAcadRead strips all 3D entity information from the drawing file. When working with block 3D block definitions an application should use VCAcadBlockRead3D allowing Corel Visual CADD to treat the file as a native symbol. Symbols act as a collection of entities that can be inserted repeatedly in a drawing. The symbols can be inserted at different locations with different rotations and scales while maintaining a unique identity separate from the objects composing the definition. When working with AutoCAD file types it is necessary to provide settings for certain conversion criteria. The criteria can be either set through code or as a result of user input to the application. The AutoCAD conversion criteria include base units, color translation, X-Ref conversion and font mapping. Corel Visual CADD provides a dialog for a user to edit these settings for conversion operations. These settings may or may not correspond to those required by the application. In situations where the application needs to control these settings the calls VCGetAcadImportUnit, VCGetPreserveAcadColorNums and VCGetKeepAcadFontName can be used to set the desired values.

See Also VCAcadBlockRead, VCAcadRead, VCAcadWriteDWG, VCGetAcadImportUnit,VCAcadWriteDXF, VCLoadDrawing, VCAcadBlockRead3D, VCGetAcadImportUnit, VCGetPreserveAcadColorNums, VCGetKeepAcadFontName

VCAcadWriteDWG

Version 1.2

Description Saves an AutoCAD DWG file from the current drawing to the specified filename.

Declaration

C/C++: extern "C" void WINAPI VCAcadWriteDWG(char* pN);

Visual Basic: Declare Sub VCAcadWriteDWG Lib "VCTRAN32.DLL" (ByVal pN As String)

Delphi: procedure VCAcadWriteDWG(pN: PChar); far;

Parameters *pN* - path and filename for the file.

Notes VCAcadWriteDWG converts the current drawing to DWG format and writes to the specified file and location. VCAcadWriteDWG is a specific load routine to work with AutoCAD files. An error will occur if attempting to save files other than *.DWG files. In situations where other vector drawing formats such *.VCD, *.GCD or *.DXF will be used the routine VCSaveDrawing should be implemented which will save all these vector file types. When working with AutoCAD file types it is necessary to provide settings for certain conversion criteria. The criteria can be either set through code or as a result of user input to the application. The AutoCAD conversion criteria include base units, color translation, X-Ref conversion and font mapping. Corel Visual CADD provides a dialog for a user to edit these settings for conversion operations. These settings may or may not correspond to those required by the application. In situations where the application needs to control these settings the calls VCGetAcadImportUnit, VCGetPreserveAcadColorNums and VCGetKeepAcadFontName can be used to set the desired values.

See Also VCAcadRead, VCAcadWriteDXF, VCAcadReadWith3D, VCSaveDrawing

VCAcadWriteDXF

Version 1.2

Description Saves an AutoCAD DXF file from the current drawing to the specified filename.

Declaration

C/C++: extern "C" void WINAPI VCAcadWriteDXF(char* pN);

Visual Basic: Declare Sub VCAcadWriteDXF Lib "VCTRAN32.DLL" (ByVal pN As String)

Delphi: procedure VCAcadWriteDXF(pN: PChar); far;

Parameters *pN* - path and filename for the file.

Notes VCAcadWriteDXF converts the current drawing to DXF format and writes to the specified file and location. VCAcadWriteDXF is a specific load routine to work with AutoCAD files. An error will occur if attempting to save files other than *.DWG files. In situations where other vector drawing formats such *.VCD, *.GCD or *.DXF will be used the routine VCSaveDrawing should be implemented which will save all these vector file types. When working with AutoCAD file types it is necessary to provide settings for certain conversion criteria. The criteria can be either set through code or as a result of user input to the application. The AutoCAD conversion criteria include base units, color translation, X-Ref conversion and font mapping. Corel Visual CADD provides a dialog for a user to edit these settings for conversion operations. These settings may or may not correspond to those required by the application. In situations where the application needs to control these settings the calls VCGetAcadImportUnit, VCGetPreserveAcadColorNums and VCGetKeepAcadFontName can be used to set the desired values.

See Also VCAcadRead, VCAcadWriteDWG, VCAcadReadWith3D, VCSaveDrawing

VCAddAngularDimensionEntity

Version 1.2

Description Adds an angular dimension entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddAngularDimensionEntity(short* iError, short iSymbolIndex, Point2D dpP0, Point2D dpP1, Point2D dpP2, Point2D dpP3);
extern "C" void WINAPI VCAddAngularDimensionEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0, Point2D* dpP1, Point2D* dpP2, Point2D* dpP3);

Visual Basic: Declare Sub VCAddAngularDimensionEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D, dpP1 As Point2D, dpP2 As Point2D, dpP3 As Point2D)

Delphi: procedure VCAddAngularDimensionEntityBP(var iError: Integer; iSymbolIndex: Integer; var dpP0: Point2D; var dpP1: Point2D; var dpP2: Point2D; var dpP3 Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the first dimension point.
dpP1 - the Point2D structure containing the coordinates to place the second dimension point on the first ray.
dpP2 - the Point2D structure containing the coordinates to place the second dimension point on the second ray.
dpP3 - the Point2D structure containing the coordinates to place the dimension line.

Notes Any dimension added to the Corel Visual CADD drawing database or to a symbol definition utilizes the current dimension settings from the dimension and dimension text tabs in the settings dialog. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing. To add dimension entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddDiameterDimensionEntity, VCAddLinearDimensionEntity , VCCreateSymbolDef, VCGetSymbolIndex

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Parsing the Database;Symbol Operations',0,`',`')}
Task Guide Examples

VCAddArcEntity

Version 1.2

Description Adds an arc entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddArcEntity(short* iError, short iSymbolIndex, Point2D dpP0, Point2D dpP1, Point2D dpP2);
extern "C" void WINAPI VCAddArcEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0, Point2D* dpP1, Point2D* dpP2);

Visual Basic: Declare Sub VCAddArcEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D, dpP1 As Point2D, dpP2 As Point2D)

Delphi: procedure VCAddArcEntityBP(var iError: Integer; iSymbolIndex: Integer; dpP0: Point2D; var dpP1: Point2D; var dpP2: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the first endpoint of the arc.
dpP1 - the Point2D structure containing the coordinates to place the mid point on the arc.
dpP2 - the Point2D structure containing the coordinates to place the second endpoint.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add arc entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddEllipticalArcEntity, VCCreateSymbolDef, VCGetColorIndex, VCGetLayerIndex, VCGetLineTypeIndex, VCGetLineWidthIndex, VCGetSymbolIndex

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Applying Settings to an Entity;Creating a Symbol;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAddAtbDef

Version 1.2

Description Adds an attribute definition to the drawing database and sets the value for the first field.

Declaration

C/C++: extern "C" void WINAPI VCAddAtbDef(short* iError, char* szName, char* Label0, char* Value0);

Visual Basic: Declare Sub VCAddAtbDef Lib "VCMAIN32.DLL" (iError As Integer, ByVal szName As String, ByVal Label0 As String, ByVal Value0 As String)

Delphi: procedure VCAddAtbDef(var iError: Integer; szName: PChar; Label0: PChar; Value0: PChar); far;

Parameters *szName* - name of the attribute.
Label0 - label text for the first field.
Value0 - default value assigned to the first field.

Notes VCAddAtbDef must be used to create an attribute definition prior to attachment to a symbol definition. Once the definition has been created, additional fields may be added using VCSetAtbDefLabelValue.

See Also VCGetAtbDefRecordCount, VCGetAtbDefValue, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbCount, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecLabel, VCGetCurEntAtbRecValue

{button ,AL(`Attaching User Data;Attribute Manipulation;Creating a Symbol;Modifying a Symbol Definition;Parsing a Symbol Definition;Retrieving Attributes',0,`',`')} Task Guide Examples

VCAddBezierEntity

Version 1.2

Description Adds a single Bezier entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddBezierEntity(short* iError, short iSymbolIndex, Point2D dpP0, Point2D dpP1, Point2D dpP2, Point2D dpP3);
extern "C" void WINAPI VCAddBezierEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0, Point2D* dpP1, Point2D* dpP2, Point2D* dpP3);

Visual Basic: Declare Sub VCAddBezierEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D, dpP1 As Point2D, dpP2 As Point2D, dpP3 As Point2D)

Delphi: procedure VCAddBezierEntityBP(var iError: Integer; iSymbolIndex: Integer; dpP0: Point2D; var dpP1: Point2D; var dpP2: Point2D; var dpP3: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the first endpoint.
dpP1 - the Point2D structure containing the coordinates to place the second end point.
dpP2 - the Point2D structure containing the coordinates to place the first control point.
dpP3 - the Point2D structure containing the coordinates to place the second control point.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddSplineEntity, VCAddArcEntity, VCAddEllipticalArcEntity, VCAddContinuousBezierEntity,VCGetColorIndex , VCGetLayerIndex,VCGetLineTypeIndex, VCGetLineWidthIndex

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Applying Settings to an Entity;Modifying Existing Entities;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAddCircleEntity

Version 1.2

Description Adds a circle entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddCircleEntity(short* iError, short iSymbolIndex, Point2D dpP0, Point2D dpP1);
extern "C" void WINAPI VCAddCircleEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0, Point2D* dpP1);

Visual Basic: Declare Sub VCAddCircleEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D, dpP1 As Point2D)

Delphi: procedure VCAddCircleEntityBP(var iError: Integer; iSymbolIndex: Integer; dpP0: Point2D; var dpP1: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the center of the circle.
dpP0 - the Point2D structure containing the coordinates to place a radius point of the circle.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddEllipseEntity, VCAddArcEntity, VCCreateSymbolDef, VCGetSymbolIndex

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Applying Settings to an Entity;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAddContinuousBezierEntity

Version 1.2

Description Adds a continuous Bezier entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddContinuousBezierEntity(short* iError, short iSymbolIndex);

Visual Basic: Declare Sub VCAddContinuousBezierEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer)

Delphi: procedure VCAddContinuousBezierEntity(var iError: Integer; iSymbolIndex: Integer); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.

Notes VCAddContinuousLineEntity, VCAddSplineEntity and VCAddContinuousBezierEntity allow for an infinite number of points to be placed with the VCSetCurrentEntityPoint command instead of through a parameter. Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddContinuousLineEntity, VCSetCurrentEntityPoint, VCAddBezierEntity, VCGetSymbolIndex, VCCreateSymbolDef, VCAddSplineEntity

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Applying Settings to an Entity;Parsing a Filtered Entity List;Parsing an On Screen List;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAddContinuousLineEntity

Version 1.2

Description Adds a continuous line entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddContinuousLineEntity(short* iError, short iSymbolIndex);

Visual Basic: Declare Sub VCAddContinuousLineEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer)

Delphi: procedure VCAddContinuousLineEntity(var iError: Integer; iSymbolIndex: Integer); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.

Notes AddContinuousLineEntity, VCAddSplineEntity and VCAddContinuousBezierEntity allow for an infinite number of points to be placed with the VCSetCurrentEntityPoint command instead of through a parameter. Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddContinuousBezierEntity, VCSetCurrentEntityPoint, VCGetSymbolIndex, VCCreateSymbolDef. VCAddSplineEntity, VCAddLineEntity

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Applying Settings to an Entity;Parsing a Filtered Entity List;Parsing an On Screen List;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAddContinuousLine3DEntity

Version 2.0

Description Adds a continuous 3D line to the drawing database.

Declaration

C/C++ extern "C" void WINAPI VCAddContinuousLine3DEntity(short* iError, short iSymbolIndex);

Visual Basic Declare Sub VCAddContinuousLine3DEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer)

Delphi procedure VCAddContinuousLine3DEntity(var iError: Integer; iSymbolIndex: Integer);far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.

Notes VCAddContinuousLineEntity3D allow for an infinite number of points to be placed with the VCSetCurrentEntityPoint3D command instead of through a parameter. Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddLine3D, VCAddPoint3D, VCAddPolygon3D , VCSetCurrentEntityPoint3D

VCAddCurrentEntityUserDataByte

Version 1.2

Description Adds a byte to the end of the user data to the current entity.

Declaration

C/C++: extern "C" void WINAPI VCAddCurrentEntityUserDataByte(short* iError, BYTE b);

Visual Basic: Declare Sub VCAddCurrentEntityUserDataByte Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)

Delphi: procedure VCAddCurrentEntityUserDataByte(var iError: Integer; b: Integer); far;

Parameters *b* - the byte of data to add.

Notes User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of the C variable types double, float, long, or short. In addition to these types, a user defined type of "chunk" may also be stored. A chunk may be any size and is simply a pointer to a memory location. The size of the chunk is also passed so Corel Visual CADD can retrieve the appropriate amount of data from the specified memory location. Whenever using user data, an application must set a user data name in order to protect private data and to ensure that different applications do not interfere with the others data. VCSetUserDataName is provided for this purpose, while VCGetUserDataName checks the currently set user data name. The name must only be set one time before adding any user data. The VCAddCurrentEntityUserData* calls always append the new variable as the last user data variable. The VCSetCurrentEntityUserData* calls add the user data variable at the index specified in the call, provided that there are indeed that many indices already attached, and overwrite any existing user data at that index. As previously mentioned, user data may be attached to the drawing header. This is achieved by using VCSetHeaderUserData and then attaching the appropriate user data. Once VCNextEntity or any other current entity selections are used, the user data calls will again be used on the current entity.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize

{button ,AL(`Attaching User Data;Attribute Manipulation;Database Operations;Parsing an Expanded List;Retrieving Attributes',0,`',`')} Task Guide Examples

VCAddCurrentEntityUserDataChunk

Version 1.2

Description Adds a chunk record to the end of the user data to the current entity.

Declaration

C/C++: extern "C" void WINAPI VCAddCurrentEntityUserDataChunk(short* iError, void* p, short iSize);

Visual Basic: Declare Sub VCAddCurrentEntityUserDataChunk Lib "VCMAIN32.DLL" (iError As Integer, ByVal p As String, ByVal iSize As Integer)

Delphi: procedure VCAddCurrentEntityUserDataChunk(var iError: Integer; var p: Pointer; iSize: Integer); far;

Parameters *p* - a pointer to a memory location where the data chunk is stored.
iSize - the size of the data chunk.

Notes User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of the C variable types double, float, long, or short. In addition to these types, a user defined type of "chunk" may also be stored. A chunk may be any size and is simply a pointer to a memory location. The size of the chunk is also passed so Corel Visual CADD can retrieve the appropriate amount of data from the specified memory location. Whenever using user data, an application must set a user data name in order to protect private data and to ensure that different applications do not interfere with the others data. VCSetUserDataName is provided for this purpose, while VCGetUserDataName checks the currently set user data name. The name must only be set one time before adding any user data. The VCAddCurrentEntityUserData* calls always append the new variable as the last user data variable. The VCSetCurrentEntityUserData* calls add the user data variable at the index specified in the call, provided that there are indeed that many indices already attached, and overwrite any existing user data at that index. As previously mentioned, user data may be attached to the drawing header. This is achieved by using VCSetHeaderUserData and then attaching the appropriate user data. Once VCNextEntity or any other current entity selections are used, the user data calls will again be used on the current entity.

See Also VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID,VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize

{button ,AL(`Attaching User Data;Database Operations;Parsing a Filtered Entity List;Parsing the Database;User Data Retrieval;User Data Tasks',0,`','')} Task Guide Examples

VCAddCurrentEntityUserDataDouble

Version 1.2

Description Adds user data of the type floating point double precision to the current entity at the end of the user data.

Declaration

C/C++: extern "C" void WINAPI VCAddCurrentEntityUserDataDouble(short* iError, double dRet);

Visual Basic: Declare Sub VCAddCurrentEntityUserDataDouble Lib "VCMAIN32.DLL" (iError As Integer, ByVal dRet As Double)

Delphi: procedure VCAddCurrentEntityUserDataDouble(var iError: Integer; dRet: Double); far;

Parameters *dRet* - the double value to add.

Notes User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of the C variable types double, float, long, or short. In addition to these types, a user defined type of "chunk" may also be stored. A chunk may be any size and is simply a pointer to a memory location. The size of the chunk is also passed so Corel Visual CADD can retrieve the appropriate amount of data from the specified memory location. Whenever using user data, an application must set a user data name in order to protect private data and to ensure that different applications do not interfere with the others data. VCSetUserDataName is provided for this purpose, while VCGetUserDataName checks the currently set user data name. The name must only be set one time before adding any user data. The VCAddCurrentEntityUserData* calls always append the new variable as the last user data variable. The VCSetCurrentEntityUserData* calls add the user data variable at the index specified in the call, provided that there are indeed that many indices already attached, and overwrite any existing user data at that index. As previously mentioned, user data may be attached to the drawing header. This is achieved by using VCSetHeaderUserData and then attaching the appropriate user data. Once VCNextEntity or any other current entity selections are used, the user data calls will again be used on the current entity.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID,VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize

{button ,AL(`Attaching User Data;Attribute Manipulation;Database Operations;Parsing an Expanded List;Retrieving Attributes',0,`',`')} Task Guide Examples

VCAddCurrentEntityUserDataFloat

Version 1.2

Description Adds user data of the type float to the current entity at the end of the user data.

Declaration

C/C++: extern "C" void WINAPI VCAddCurrentEntityUserDataFloat(short* iError, float f);

Visual Basic: Declare Sub VCAddCurrentEntityUserDataFloat Lib "VCMAIN32.DLL" (iError As Integer, ByVal f As Double)

Delphi: procedure VCAddCurrentEntityUserDataFloat(var iError: Integer; f: Double); far;

Parameters *f* - the float value to add.

Notes User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of the C variable types double, float, long, or short. In addition to these types, a user defined type of "chunk" may also be stored. A chunk may be any size and is simply a pointer to a memory location. The size of the chunk is also passed so Corel Visual CADD can retrieve the appropriate amount of data from the specified memory location. Whenever using user data, an application must set a user data name in order to protect private data and to ensure that different applications do not interfere with the others data. VCSetUserDataName is provided for this purpose, while VCGetUserDataName checks the currently set user data name. The name must only be set one time before adding any user data. The VCAddCurrentEntityUserData* calls always append the new variable as the last user data variable. The VCSetCurrentEntityUserData* calls add the user data variable at the index specified in the call, provided that there are indeed that many indices already attached, and overwrite any existing user data at that index. As previously mentioned, user data may be attached to the drawing header. This is achieved by using VCSetHeaderUserData and then attaching the appropriate user data. Once VCNextEntity or any other current entity selections are used, the user data calls will again be used on the current entity.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID,VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize

{button ,AL(`Attaching User Data;Attribute Manipulation;Database Operations;Parsing an Expanded List;Retrieving Attributes',0,`','`')} Task Guide Examples

VCAddCurrentEntityUserDataLong

Version 1.2

Description Adds user data of the type long for the current entity to the end of the user data.

Declaration

C/C++: extern "C" void WINAPI VCAddCurrentEntityUserDataLong(short* iError, long l);

Visual Basic: Declare Sub VCAddCurrentEntityUserDataLong Lib "VCMAIN32.DLL" (iError As Integer, ByVal l As Long)

Delphi: procedure VCAddCurrentEntityUserDataLong(var iError: Integer; l: Longint); far;

Parameters *l* - the long integer value to add.

Notes User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of the C variable types double, float, long, or short. In addition to these types, a user defined type of "chunk" may also be stored. A chunk may be any size and is simply a pointer to a memory location. The size of the chunk is also passed so Corel Visual CADD can retrieve the appropriate amount of data from the specified memory location. Whenever using user data, an application must set a user data name in order to protect private data and to ensure that different applications do not interfere with the others data. VCSetUserDataName is provided for this purpose, while VCGetUserDataName checks the currently set user data name. The name must only be set one time before adding any user data. The VCAddCurrentEntityUserData* calls always append the new variable as the last user data variable. The VCSetCurrentEntityUserData* calls add the user data variable at the index specified in the call, provided that there are indeed that many indices already attached, and overwrite any existing user data at that index. As previously mentioned, user data may be attached to the drawing header. This is achieved by using VCSetHeaderUserData and then attaching the appropriate user data. Once VCNextEntity or any other current entity selections are used, the user data calls will again be used on the current entity.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID,VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize

{button ,AL(`Attaching User Data;Attribute Manipulation;Database Operations;Parsing an Expanded List;Retrieving Attributes',0,`',`')} Task Guide Examples

VCAddCurrentEntityUserDataShort

Version 1.2

Description Adds user data of the type short for the current entity to the end of the user data.

Declaration

C/C++: extern "C" void WINAPI VCAddCurrentEntityUserDataShort(short* iError, short s);

Visual Basic: Declare Sub VCAddCurrentEntityUserDataShort Lib "VCMAIN32.DLL" (iError As Integer, ByVal s As Integer)

Delphi: procedure VCAddCurrentEntityUserDataShort(var iError: Integer; s: Integer); far;

Parameters *s* - the short integer value to add.

Notes User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of the C variable types double, float, long, or short. In addition to these types, a user defined type of "chunk" may also be stored. A chunk may be any size and is simply a pointer to a memory location. The size of the chunk is also passed so Corel Visual CADD can retrieve the appropriate amount of data from the specified memory location. Whenever using user data, an application must set a user data name in order to protect private data and to ensure that different applications do not interfere with the others data. VCSetUserDataName is provided for this purpose, while VCGetUserDataName checks the currently set user data name. The name must only be set one time before adding any user data. The VCAddCurrentEntityUserData* calls always append the new variable as the last user data variable. The VCSetCurrentEntityUserData* calls add the user data variable at the index specified in the call, provided that there are indeed that many indices already attached, and overwrite any existing user data at that index. As previously mentioned, user data may be attached to the drawing header. This is achieved by using VCSetHeaderUserData and then attaching the appropriate user data. Once VCNextEntity or any other current entity selections are used, the user data calls will again be used on the current entity.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCGetUserDataName, VCGetCurrentEntityUID,VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize

{button ,AL(`Attaching User Data;Attribute Manipulation;Database Operations;Parsing an Expanded List;Retrieving Attributes',0,`','`')} Task Guide Examples

VCAddDiameterDimensionEntity

Version 1.2

Description Adds a diameter dimension entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddDiameterDimensionEntity(short* iError, short iSymbolIndex, Point2D dpP0, Point2D dpP1, Point2D dpP2, Point2D dpP3);
extern "C" void WINAPI VCAddDiameterDimensionEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0, Point2D* dpP1, Point2D* dpP2, Point2D* dpP3);

Visual Basic: Declare Sub VCAddDiameterDimensionEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D, dpP1 As Point2D, dpP2 As Point2D, dpP3 As Point2D)

Delphi: procedure VCAddDiameterDimensionEntityBP(var iError: Integer; Integer; var dpP0: Point2D; var dpP1: Point2D; var dpP2: Point2D; var dpP3: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the first point of the diameter.
dpP1 - the Point2D structure containing the coordinates to place the second point of the diameter.
dpP2 - the Point2D structure containing the coordinates to place the dimension line.
dpP3 - currently unused and will be ignored.

Notes Any dimension added to the Corel Visual CADD drawing database or to a symbol definition utilizes the current dimension settings from the dimension and dimension text tabs of the settings dialog These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddAngularDimensionEntity, VCAddLinearDimensionEntity , VCGetSymbolIndex, VCCreateSymbolDef

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Applying Settings to an Entity;Parsing a Filtered Entity List;Parsing an On Screen List;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAddEllipseEntity

Version 1.2

Description Adds an ellipse entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddEllipseEntity(short* iError, short iSymbolIndex, Point2D dpP0, Point2D dpP1, Point2D dpP2, Point2D dpP3);
extern "C" void WINAPI VCAddEllipseEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0, Point2D* dpP1, Point2D* dpP2, Point2D* dpP3);

Visual Basic: Declare Sub VCAddEllipseEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D, dpP1 As Point2D, dpP2 As Point2D, dpP3 As Point2D)

Delphi: procedure VCAddEllipseEntityBP(var iError: Integer; iSymbolIndex: Integer; dpP0: Point2D; var dpP1: Point2D; var dpP2: Point2D; var dpP3: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the first end point on the major axis.
dpP1 - the Point2D structure containing the coordinates to place the second end point on the major axis.
dpP2 - the Point2D structure containing the coordinates to place the second end point on the minor axis.
dpP3 - the Point2D structure containing the coordinates to place the second end point on the minor axis.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddEllipticalArcEntity, VCGetSymbolIndex, VCCreateSymbolDef

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Applying Settings to an Entity;Parsing a Filtered Entity List;Parsing an On Screen List;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAddEllipticalArcEntity

Version 1.2

Description Adds an elliptical arc entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddEllipticalArcEntity(short* iError, short iSymbolIndex, Point2D dpP0, Point2D dpP1, Point2D dpP2, Point2D dpP3, Point2D dpP4, Point2D dpP5, Point2D dpP6);
extern "C" void WINAPI VCAddEllipticalArcEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0, Point2D* dpP1, Point2D* dpP2, Point2D* dpP3, Point2D* dpP4, Point2D* dpP5, Point2D* dpP6);

Visual Basic: Declare Sub VCAddEllipticalArcEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D, dpP1 As Point2D, dpP2 As Point2D, dpP3 As Point2D, dpP4 As Point2D, dpP5 As Point2D, dpP6 As Point2D)

Delphi: procedure VCAddEllipticalArcEntityBP(var iError: Integer; Integer; var dpP0: Point2D; var dpP1: Point2D; var dpP2: Point2D; var dpP3 Point2D; var dpP4: Point2D; var dpP5: Point2D; var dpP6: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the first end point on the major axis on the ellipse.
dpP1 - the Point2D structure containing the coordinates to place the second end point on the major axis on the ellipse.
dpP2 - the Point2D structure containing the coordinates to place the second end point on the minor axis on the ellipse.
dpP3 - the Point2D structure containing the coordinates to place the second end point on the minor axis on the ellipse.
dpP4 - the Point2D structure containing the coordinates to place the starting point for the arc definition
dpP5 - the Point2D structure containing the coordinates to place the mid point for the arc definition.
dpP6 - the Point2D structure containing the coordinates to place the second end point for the arc definition

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddEllipseEntity, VCAddArcEntity, VCGetSymbolIndex, VCCreateSymbolDef

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Applying Settings to an Entity;Parsing a Filtered Entity List;Parsing an On Screen List;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAddFillEntity

Version 1.2

Description Adds a fill entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddFillEntity(short* iError, short iSymbolIndex);

Visual Basic: Declare Sub VCAddFillEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer)

Delphi: procedure VCAddFillEntity(var iError: Integer; iSymbolIndex: Integer); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.

Notes VCAddFillEntity and VCAddHatchEntity allow hatch and fill boundaries to be specified by any other entity types available in Corel Visual CADD. A hatch or fill entity is created by adding a reference to the entity type, building the boundary from other entity types and the sorting the boundary to finish the hatch or fill entity. VCSortCurrentHatchFillEntity forces Corel Visual CADD to evaluate the input boundary entities for hatching or filling. The input entities must form a closed boundary.

See Also VCAddHatchEntity, VCGetSymbolIndex, VCCreateSymbolDef, VCSetCurrentEntityPoint, VCSetCurrentSelected. VCHatchSelected, VCFillSelected, VCSortCurrentHatchFillEntity

{button ,AL(`Adding a Continuous Entity;Adding a Hatch/Fill Entity;Database Operations;Duplicating an Entity;Parsing a Filtered Entity List;Parsing the Database',0,`',`')} Task Guide Examples

VCAddHatchEntity

Version 1.2

Description Adds a hatch entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddHatchEntity(short* iError, short iSymbolIndex);

Visual Basic: Declare Sub VCAddHatchEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer)

Delphi: procedure VCAddHatchEntity(var iError: Integer; iSymbolIndex: Integer); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.

Notes VCAddFillEntity and VCAddHatchEntity allow hatch and fill boundaries to be specified by any other entity types available in Corel Visual CADD. A hatch or fill entity is created by adding a reference to the entity type, building the boundary from other entity types and the sorting the boundary to finish the hatch or fill entity. VCSortCurrentHatchFillEntity forces Corel Visual CADD to evaluate the input boundary entities for hatching or filling. The input entities must form a closed boundary.

See Also VCAddFillEntity, VCGetSymbolIndex, VCCreateSymbolDef, VCSetCurrentEntityPoint, VCSetCurrentSelected. VCHatchSelected, VCFillSelected , VCSortCurrentHatchFillEntity

{button ,AL(`Adding a Continuous Entity;Adding a Hatch/Fill Entity;Database Operations;Duplicating an Entity;Parsing a Filtered Entity List;Parsing the Database',0,`',`')} Task Guide Examples

VCAddLeaderEntity

Version 1.2

Description Adds a leader entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddLeaderEntity(short* iError, short iSymbolIndex, Point2D* P, short iPointCount);

Visual Basic: Declare Sub VCAddLeaderEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, P As Point2D, ByVal iPointCount As Integer)

Delphi: procedure VCAddLeaderEntity(var iError: Integer; iSymbolIndex: Integer; var P: Point2D; iPointCount: Integer); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
P - a pointer to an array of Point2D structures containing the coordinates of each vertex on the leader entity.
iPointCount - the number of items in the array P and the number of points contained in the leader.

Notes Any leader added to the Corel Visual CADD drawing database or to a symbol definition will take on the current leader settings as found in the dimension and dimension text tabs of the settings dialog for version prior to 2.0 or the leader tab in versions 2.0 and later. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddAngularDimensionEntity, VCAddDiameterDimensionEntity , VCAddLinearDimensionEntity , VCGetSymbolIndex, VCCreateSymbolDef

{button ,AL(`Creating a Symbol;Loading a Symbol;Modifying a Symbol Definition',0,`',`')} Task Guide Examples

VCAddLine3D

Version 1.2

Description Add a 3D line to the drawing database that is not constrained to the z=0 plane.

Declaration

C/C++: extern "C" void WINAPI VCAddLine3D(short* iError, short iSymbolIndex, Point3D* dpP0, Point3D* dpP1);

Visual Basic: Declare Sub VCAddLine3D Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point3D, dpP1 As Point3D)

Delphi: procedure VCAddLine3D(var iError: Integer; iSymbolIndex: Integer; var dpP0: Point3D; var dpP1: Point3D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 -the Point3D structure containing the coordinates to place the starting point for the line definition.
dpP1 - the Point3D structure containing the coordinates to place the ending point for the line definition.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddLineEntity , VCAddPoint3D , VCAddPolygon3D, VCGetSymbolIndex, VCCreateSymbolDef

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Parsing a Filtered Entity List;Parsing the Database;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAddLinearDimensionEntity

Version 1.2

Description Adds a linear dimension entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddLinearDimensionEntity(short* iError, short iSymbolIndex, Point2D dpP0, Point2D dpP1, Point2D dpP2, Point2D dpP3);
extern "C" void WINAPI VCAddLinearDimensionEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0, Point2D* dpP1, Point2D* dpP2, Point2D* dpP3);

Visual Basic: Declare Sub VCAddLinearDimensionEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D, dpP1 As Point2D, dpP2 As Point2D, dpP3 As Point2D)

Delphi: procedure VCAddLinearDimensionEntityBP(var iError: Integer; Integer; var dpP0: Point2D; var dpP1: Point2D; var dpP2: Point2D; var dpP3:Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the first dimension point.
dpP1 - the Point2D structure containing the coordinates to place the second dimension point.
dpP2 - the Point2D structure containing the coordinates to place the dimension line.
dpP3 - ignored.

Notes Any dimension added to the Corel Visual CADD drawing database or to a symbol definition utilizes the current dimension settings from the dimension and dimension text tabs of the settings dialog These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddAngularDimensionEntity, VCAddDiameterDimensionEntity , VCGetSymbolIndex, VCCreateSymbolDef, VCDimGetDimMode

VCAddLineEntity

Version 1.2

Description Adds a line entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddLineEntity(short* iError, short iSymbolIndex, Point2D dpP0, Point2D dpP1);
extern "C" void WINAPI VCAddLineEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0, Point2D* dpP1);

Visual Basic: Declare Sub VCAddLineEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D, dpP1 As Point2D)

Delphi: procedure VCAddLineEntityBP(var iError: Integer; iSymbolIndex: Integer; dpP0: Point2D; var dpP1: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates of the first endpoint.
dpP1 - the Point2D structure containing the coordinates of the second endpoint.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddLine3D, VCAddLineType, VCGetSymbolIndex, VCCreateSymbolDef

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Applying Settings to an Entity;Parsing the Database',0,`',`')} Task Guide Examples

VCAddLineType

Version 1.2

Description Creates a line type at the current line type index using the included array as the definition for the line type as the specified line type name.

Declaration

C/C++: extern "C" short WINAPI VCAddLineType(short* iError, char* pName, short bCode, short iDashCount, double* pDashes);

Visual Basic: Declare Function VCAddLineType Lib "VCMAIN32.DLL" (iError As Integer, ByVal pName As String, ByVal bCode As Integer, ByVal iDashCount As Integer, pDashes As Double) As Integer

Delphi: function VCAddLineType(var iError: Integer; pName: PChar; bCode: Integer; iDashCount: Integer; var pDashes: Double):Integer; far;

Parameters

pName - the name to be assigned to the line type.
bCode - determines whether the line a world scale or device scale.
1 - WORLD_SCALE.
2 - DEVICE_SCALE.
iDashCount - the number of dashes used and the size of the pDashes array.
pDashes - points to and array of double values representing each dash length.

Notes Corel Visual CADD line types use either a world scale or a device scale. Device line types will always appear with the appropriate lengths regardless of the drawing view on screen or the print size. World scale line types will always be displayed and printed to scale, that is a 1" dash printed at ¼ scale will be ¼" long on paper. The pDashes array must contain dash lengths for the line type in order they are to be drawn in the line. A positive value indicates a displayed (or on) dash length while a negative value indicates a non-displayed (or off) dash length. These non-displayed dash lengths can be thought of as an offset length from the end of the last dash length to the beginning of the next dash length.

See Also VCAddLine3D, VCAddLineEntity

VCAddNamedLayer

Version 1.2

Description Names the current layer and returns the current layer index.

Declaration

C/C++: extern "C" short WINAPI VCAddNamedLayer(short* iError, char* pName);

Visual Basic: Declare Function VCAddNamedLayer Lib "VCMAIN32.DLL" (iError As Integer, ByVal pName As String) As Integer

Delphi: function VCAddNamedLayer(var iError: Integer; pName: PChar):Integer; far;

Parameters *pName* - the name to assign to the current layer.
return - the layer index from 0 to 1024.

Notes The API provides two methods for naming layers in the active drawing. The first utilizes VCAddNamedLayer and simply names the first layer in the list that has not already been named. The function begins a parse on a 0 based layer index until the first non-named layer. It then names the layer the given value and returns the index for the layer. This routine is generally used when building a setup routine where the entire layer naming scheme is known up front. The second method allows the application to apply a name to a specific layer. VCSetNamedLayer takes a layer index as a parameter for naming. This operates more in hand with the Corel Visual CADD interface since a user or application can pick the layer to name prior to the operation.

See Also VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetLayerIndex VCGetLayerIndexFromName VCGetLayerNameFromIndex

VCAddPlotter

Version 2.0

Description Creates a new plotter definition for the direct plot list.

Declaration

C/C++: extern "C" void WINAPI VCAddPlotter(short* iError, char* szPlotterName);

Visual Basic: Declare Sub VCAddPlotter Lib "VCDLG32.DLL" (iError As Integer, ByVal szPlotterName As String)

Delphi: procedure VCAddPlotter(var iError: Integer; szPlotterName: PChar); far;

Parameters *szPlotterName* - the name of the plotter driver to add.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

If a plotter is not supported by drivers provided, an application or end user may create a new driver form the plotters language control. This requires the user or application to name the new driver being created. The actual plotter language strings are then defined through the API or Corel Visual CADD interface.

See Also VCAddPlotterLanguageName, VCAddPlotterPageSize, VCAddPlotterPenMapName, VCGetPlotterCount

VCAddPlotterLanguageName

Version 2.0

Description Adds a plotter language name to the direct plot routine.

Declaration

C/C++: extern "C" void WINAPI VCAddPlotterLanguageName(short* iError, char* szLanguageName);

Visual Basic: Declare Sub VCAddPlotterLanguageName Lib "VCDLG32.DLL" (iError As Integer, ByVal szLanguageName As String)

Delphi: procedure VCAddPlotterLanguageName(var iError: Integer; szLanguageName: PChar); far;

Parameters *szLanguageName* - the plotter language name to add.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs to be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type

See Also VCAddPlotter, VCAddPlotterPageSize, VCAddPlotterPenMapName, VCGetPlotterCount

VCAddPlotterPageSize

Version 2.0

Description Adds a plotter page size from the direct plot options.

Declaration

C/C++: extern "C" void WINAPI VCAddPlotterPageSize(short* iError, Point2D* pPageSize);

Visual Basic: Declare Sub VCAddPlotterPageSize Lib "VCDLG32.DLL" (iError As Integer, pPageSize As Point2D)

Delphi: procedure VCAddPlotterPageSize(var iError: Integer; var pPageSize: Point2D); far;

Parameters *szPageSize* - the page size to add to the list.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine allows for custom page sizes to be defined with the VCAddPlotterPageSizeRoutine and by the user through the Corel Visual CADD interface. These can be removed from the interface by the user or through the API with VCRemovePlotterPageSize and added with VCAddPlotterPageSize. Custom page sizes enhance the users control over vector output devices and allows the user or an application to set page parameters suited to a desired output.

See Also VCAddPlotterLanguageName, VCAddPlotter, VCAddPlotterPenMapName, VCGetPlotterCount

VCAddPlotterPenMapName

Version 2.0

Description Adds a plotter pen map to the direct plot interface.

Declaration

C/C++: extern "C" void WINAPI VCAddPlotterPenMapName(short* iError, char* szPenMapName);

Visual Basic: Declare Sub VCAddPlotterPenMapName Lib "VCDLG32.DLL" (iError As Integer, ByVal szPenMapName As String)

Delphi: procedure VCAddPlotterPenMapName(var iError: Integer; szPenMapName: PChar); far;

Parameters *szPenMapName* - the pen map name to add to the plotter list.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device. The pen map controls the color, speed and width setting for each pen used by the plotter.

See Also VCAddPlotterLanguageName, VCAddPlotter, VCAddPlotterPageSize, VCGetPlotterCount

VCAddPoint3D

Version 1.2

Description Add a 3D point to the drawing database that is not constrained to the z=0 plane.

Declaration

C/C++: extern "C" void WINAPI VCAddPoint3D(short* iError, short iSymbolIndex, Point3D* dpP);

Visual Basic: Declare Sub VCAddPoint3D Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP As Point3D)

Delphi: procedure VCAddPoint3D(var iError: Integer; iSymbolIndex: Integer; var dpP: Point3D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point3D structure containing the coordinates to place the point definition.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAcadReadWith3D,VCAddLine3D, VCAddPolygon3D, VCGetSymbolIndex, VCCreateSymbolDef

VCAddPointEntity

Version 1.2

Description Adds a point entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddPointEntity(short* iError, short iSymbolIndex, Point2D dpP0);
extern "C" void WINAPI VCAddPointEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0);

Visual Basic: Declare Sub VCAddPointEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D)

Delphi: procedure VCAddPointEntityBP(var iError: Integer; iSymbolIndex: Integer; dpP0: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the entity.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddLineEntity, VCCreateSymbolDef, VCGetSymbolIndex, VCAddPoint3D

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity;Parsing an Expanded List;Parsing an On Screen List;Parsing the Database',0,`',`')} Task Guide Examples

VCAddPolygon3D

Version 1.2

Description Add a 3D polygon to the drawing database that is not constrained to the z=0 plane.

Declaration

C/C++: extern "C" void WINAPI VCAddPolygon3D(short* iError, short iSymbolIndex, Point3D* dpP, short iNumPnts);

Visual Basic: Declare Sub VCAddPolygon3D Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP As Point3D, ByVal iNumPnts As Integer)

Delphi: procedure VCAddPolygon3D(var iError: Integer; iSymbolIndex: Integer; var dpP: Point3D; iNumPnts: Integer); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP - a array of user defined Point3D structures.
lNumPnts - the number of points contained in the array.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. Once a polygon3D is added to the drawing it contains no points and must have points added using VCSetCurrentEntityPoint3D. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef make an empty symbol definition for creating a new symbol

See Also VCAddPoint3D , VCAddLine3D, VCCreateSymbolDef, VCGetSymbolIndex, VCGetCurrentEntityPoint3D

VCAddPopupCommand

Version 1.2

Description Adds a command to right button pop-up menu used with the current tool.

Declaration

C/C++: extern "C" void WINAPI VCAddPopupCommand(char* szNativeCmd, short iPlacement);

Visual Basic: Declare Sub VCAddPopupCommand Lib "VCTOOL32.DLL" (ByVal szNativeCmd As String, ByVal iPlacement As Integer)

Delphi: procedure VCAddPopupCommand(szNativeCmd: PChar; iPlacement: Integer); far;

Parameters *szNativeCmd* - the name of a command as defined in cmdext.def.
iPlacement - determines where in the menu to place the item.
0 - INSERT
1 - APPEND
2 - SEPARATOR

Notes While pop-up menus can be defined independently by the user, VCAddPopupCommand allows a native command to be added to the pop-up of the currently active tool for only the current session of that tool. When the tool is no longer active, any commands added to the tool will be lost and need to be re-added if required for the next instance of that tool. If the pop-up needs to be cleared of all default commands, VCDeletePopupMenu will remove all the existing defaults for the current instance of the tool. VCDeletePopupMenu will not affect commands added with VCAddPopupCommand. These commands only work on the current tool i.e. there must be a tool active in order to add to or clear the contents of the pop-up menu.

See Also VCDeletePopupMenu

{button ,AL(`Creating a User Tool;Using the Corel Visual CADD Interface;Utilizing a Custom Interface',0,`',`')} Task Guide Examples

VCAddRadialDimensionEntity

Version 1.2

Description Adds a radial dimension entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddRadialDimensionEntity(short* iError, short iSymbolIndex, Point2D dpP0, Point2D dpP1, Point2D dpP2, Point2D dpP3);
extern "C" void WINAPI VCAddRadialDimensionEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0, Point2D* dpP1, Point2D* dpP2, Point2D* dpP3);

Visual Basic: Declare Sub VCAddRadialDimensionEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D, dpP1 As Point2D, dpP2 As Point2D, dpP3 As Point2D)

Delphi: procedure VCAddRadialDimensionEntityBP(var iError: Integer; Integer; var dpP0: Point2D; var dpP1: Point2D; var dpP2: Point2D; var dpP3: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the center of the radius.
dpP1 - the Point2D structure containing the coordinates to place the endpoint of the radius.
dpP2 - the Point2D structure containing the coordinates to place the dimension line.
dpP3 - currently unused and will be ignored.

Notes Any dimension added to the Corel Visual CADD drawing database or to a symbol definition utilizes the current dimension settings from the dimension and dimension text tabs of the settings dialog These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddAngularDimensionEntity, VCAddDiameterDimensionEntity, VCAddLinearDimensionEntity , VCGetSymbolIndex, VCCreateSymbolDef

VCAddRefFrameEntity

Version 2.0

Description Adds a reference frame entity.

Declaration

C/C++ extern "C" void WINAPI VCAddRefFrameEntity(short* iError, short iSymbolIndex, Point2D* dpP0);

Visual Basic Declare Sub VCAddRefFrameEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D)

Delphi procedure VCAddRefFrameEntity(var iError: Integer; iSymbolIndex: Integer; var

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the reference frame.

Notes Reference frame entities enable a drawing file to be referenced or linked into another drawing. The frames can be used to layout drawings for printing or to create overlay patterns. The reference frame can be bound, data is not dynamic and is stored in the parent drawing, or dynamic in which the referenced file is updated as changes are made to the original.

When linked, the files are represented by a relative path between the current file location and the absolute path to the file. For example, if the current active drawing for an open VCD files is "C:\VCADD\SAMPLES\THISFILE.VCD" and a file is referenced into this drawing located at an absolute location of "C:\VCADD\LINKEDFILE.VCD" VCRelativePath will return the difference of the paths. In this case it will return " ..\" or indication that the linked file is located back one subdirectory.

The reference frame, the actual border around the linked file, behaves as a primitive entity with color, rotation, scale and other properties. All these can be used to manipulate the frame for displaying the desired data.

To add a reference frame, the application should first set a pointer to the file being referenced with VCSetRefFrameName. VCAddRefFrameEntity will then reference this file in at the current position.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

{button ,AL(`Adding a Reference Frame Entity',0,`',`')} Task Guide Examples%!Alink(Adding a Reference Frame Entity, , ,)

VCAddSplineEntity

Version 1.2

Description Add a spline entity to the drawing database without any points to allow data points to added later.

Declaration

C/C++: extern "C" void WINAPI VCAddSplineEntity(short* iError, short iSymbolIndex);

Visual Basic: Declare Sub VCAddSplineEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer)

Delphi: procedure VCAddSplineEntity(var iError: Integer; iSymbolIndex: Integer); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.

Notes VCAddContinuousLineEntity, VCAddSplineEntity and VCAddContinuousBezierEntity allow for an infinite number of points to be placed with the VCSetCurrentEntityPoint command instead of through a parameter. Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCAddArcEntity, VCAddBezierEntity, VCGetSymbolIndex, VCCreateSymbolDef, VCAddContinuousBezierEntity

VCAddSymbolEntity

Version 1.2

Description Adds a symbol entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddSymbolEntity(short* iError, short iSymbolIndex, Point2D dpP0);
extern "C" void WINAPI VCAddSymbolEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0);

Visual Basic: Declare Sub VCAddSymbolEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D)

Delphi: procedure VCAddSymbolEntityBP(var iError: Integer; iSymbolIndex: Integer; dpP0: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the entity.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCGetSymbolIndex, VCCreateSymbolDef VCGetSymName, VCGetSymbolName, VCGetSymbolIndex

{button ,AL(`Creating a Symbol;Loading a Symbol;Parsing a Symbol Definition;Placing a Symbol;Symbol Operations',0,`',`')} Task Guide Examples

VCAddSymbol3DEntity

Version 2.0

Description Adds a 3D symbol entity to the drawing or another symbol definition.

Declaration

C/C++ extern "C" void WINAPI VCAddSymbol3DEntity(short* iError, short iSymbolIndex, Point3D* dpP0);

Visual Basic Declare Sub VCAddSymbol3DEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point3D)

Delphi procedure VCAddSymbol3DEntity(var iError: Integer; iSymbolIndex: Integer; var

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point3D structure containing the coordinates to place the entity.

Notes Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These properties should be set before adding the entity or they may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCGetSym3DName , VCGetSym3DNormal , VCGetSym3DRot , VCGetSym3DScale

VCAddTextEntity

Version 1.2

Description Adds a text line entity to the drawing database or to a symbol definition.

Declaration

C/C++: extern "C" void WINAPI VCAddTextEntity(short* iError, short iSymbolIndex, Point2D dpP0);
extern "C" void WINAPI VCAddTextEntityBP(short* iError, short iSymbolIndex, Point2D* dpP0);

Visual Basic: Declare Sub VCAddTextEntityBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iSymbolIndex As Integer, dpP0 As Point2D)

Delphi: procedure VCAddTextEntityBP(var iError: Integer; iSymbolIndex: Integer; dpP0: Point2D); far;

Parameters *iSymbolIndex* - index location for adding the entity.
-1 - NONDEFENTITY (Drawing)
-2 - HATCHFILLENTITY
> 0 - Use VCGetSymbolIndex to retrieve the symbol index for creating a symbol definition.
dpP0 - the Point2D structure containing the coordinates to place the text entity.

Notes Any text added to the Corel Visual CADD drawing database or to a symbol definition will take on the current text properties for font, color, layer, size, spacing, justification, formatting and aspect. The string to be added is set with VCSetTextString prior to placing the text line with VCAddTextEntity. These all need to be set before creating these entities or may be changed after creation with the text edit commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of one higher than the last entity in the drawing before the addition. To add entities to a symbol definition, the index of an existing symbol is retrieved with VCGetSymbolIndex while VCCreateSymbolDef creates an empty definition for a new symbol.

See Also VCGetTextString, VCGetSymbolIndex, VCCreateSymbolDef

{button ,AL(`Adding a Text Entity;Applying Settings to an Entity;Parsing a Filtered Entity List;Parsing an On Screen List;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAngleToString

Version 1.2

Description Converts a supplied angle to a string formatted according to current angle display settings.

Declaration

C/C++ extern "C" short WINAPI VCAngleToString(short* iError, char* pS, double* pA);

Visual Basic Declare Function VCAngleToString Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String, pA As Double) As Integer

Delphi function VCAngleToString(var iError: Integer; pS: PChar; var pA: Double):Integer; far;

Parameters *pS* - the string returned by the function.
pA - the angle in radians to be formatted.
r*eturn* - the number of characters in the formatted string.

Notes When displaying angles, the output must be in the same units as the user has set in the numeric tab settings. This maintains a consistent look across applications and prevents user confusion that may occur if several different display formats are used. The supplied angle must be in radians, as is the case with all Corel Visual CADD API calls.

See Also VCStringToAngle, VCStringToAngle, VCDistToString, VCGetUnitConversionFactor

VCAppExit

Version	1.2
Description	Alerts Corel Visual CADD that the application is exiting and initiates internal clean-up.
Declaration	
C/C++	extern "C" void WINAPI VCAppExit(short* iError);
Visual Basic	Declare Sub VCAppExit Lib "VCMAIN32.DLL" (iError As Integer)
Delphi	procedure VCAppExit(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	VCAppExit is a general clean up routine utilized to free memory after the completion of a tool. When running external application tool sets, memory from the API and the tool itself may not always be cleared. For instance when creating a preview window with a drawing world one of the five HDC available in Windows 3.1 will be used. VCAppExit will alert Corel Visual CADD to attempt any maintenance required to free up resources used by the external application.
See Also	VCBeginOperation, VCEndOperation, VCAbortOperation, VCSetAlertApp, VCClearAlertApp

{button ,AL(`Creating a User Tool;Utilizing a Custom Interface',0,`',`')} Task Guide Examples

VCApplyPlotterLanguageDefaults

Version 2.0

Description Resets the direct plot language settings to the default values.

Declaration

C/C++ extern "C" void WINAPI VCApplyPlotterLanguageDefaults(short* iError);

Visual Basic Declare Sub VCApplyPlotterLanguageDefaults Lib "VCDLG32.DLL" (iError As Integer)

Delphi procedure VCApplyPlotterLanguageDefaults(var iError: Integer); far;

Parameters No additional parameters are used with this subroutine.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

See Also VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMoveString, VCGetPlotterPenUpString

VCApplyPlotterPenMapDefaults

Version 2.0

Description Resets the direct plot pen mapping settings to the default values.

Declaration

C/C++: extern "C" void WINAPI VCApplyPlotterPenMapDefaults(short* iError);

Visual Basic: Declare Sub VCApplyPlotterPenMapDefaultsLib "VCDLG32.DLL" (iError As Integer)

Delphi: procedure VCApplyPlotterPenMapDefaults(var iError: Integer); far;

Parameters No additional parameters are used with this subroutine.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

See Also VCGetPlotterCurrentPenMapName

VCApplySettingsToCurrentEntity

Version	1.2.1
Description	Forces all current applicable settings to be applied to the current entity.
Declaration	
C/C++:	extern "C" void WINAPI VCApplySettingsToCurrentEntity(short* iError);
Visual Basic:	Declare Sub VCApplySettingsToCurrentEntity Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCApplySettingsToCurrentEntity(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	This subroutine provides an easy way to change the settings of the current entity without using VCDuplicate. All settings that are used by the entity are applied while ignoring all others. The current entity must first be set using VCFirstEntity, VCNextEntiy, VCFirstSelected, VCNextSelected, VCFirstOnScreen, or VCNextOnScreen.
See Also	VCDuplicate, VCFirstEntity, VCNextEntity, VCFirstSelected, VCNextSelected, VCFirstOnScreen, VCNextOnScreen, VCChangeSelected, VCDuplicateWithTransform

{button ,AL(`Adding a Continuous Entity;Applying Settings to an Entity;Retrieving Entity Properties',0,`',`')} Task Guide Examples

VCAuditUIDS

Version	2.0
Description	Audits the Unique Entity Ids to ensure there are no duplicates.
Declaration	
C/C++	extern "C" long WINAPI VCAuditUIDS(short* iError);
Visual Basic	Declare Function VCAuditUIDS Lib "VCMAIN32.DLL" (iError As Integer) As Long
Delphi	function VCAuditUIDS(var iError: Integer):Longint; far;
Parameters	No additional parameters are used with this subroutine.
Notes	Each entity in Corel Visual CADD 2.0 maintains a unique entity identifier in order to track the entity. This is in addition to the dynamic ENTITYHANDLE which changes as entities are deleted and modified in the database. As entities are added to the drawing both an entity handle and a UID are assigned to the entity. The entity handle will change as items are deleted and modified on the database while the UID will remain constant. Whenever linking entities to external databases or static arrays, the application should utilize the UID due to its unchanging value with each entity. The entity handle is used when parsing the database or setting specific entities within the drawing session. The UID can should be audited prior to any external storage in order to ensure uniqueness in the ID.
See Also	VCGetCurrentEntityUID

VCBeginOperation

Version 1.2

Description Marks the start of an operation where an undo level begins.

Declaration

C/C++: extern "C" void WINAPI VCBeginOperation(short* iError);

Visual Basic: Declare Sub VCBeginOperation Lib "VCMAIN32.DLL" (iError As Integer)

Delphi: procedure VCBeginOperation(var iError: Integer); far;

Parameters No additional parameters are used with this subroutine.

Notes Corel Visual CADD provides a set of user tool functions to build and create tools not directly supported in the interface. For example, a multi-line tool that automatically hatches or fills the segments. Since this tool is not provided directly in the Corel Visual CADD interface, it must be created through code to interact with the existing commands such as snaps and undo operations. In order for the tools to respond appropriately to undo operations it should set undo and redo levels during the operation. A complex entity tool, one that adds multiple entities such as the multi-line example, can allow each individual entity or instead the entire operation to be undone with a single user undo operation. This depends on the design criteria specified for the application. The level of undo is set with the VCBeginOperation and VCEndOperation API routines. An application should set the beginning of the undo level prior to adding any entities to the drawing database and finish the tool with an end operation. In certain situations, the tool may be aborted by the user typically by pressing the <ESC> key. An application should respond appropriately by aborting the undo level to return it to the state prior to the user tool operation. VCAbortOperation will handle this for the application. When used in conjunction with VCBeginOperation and VCEndOperation, VCAbortOperation will discard all undo information complied since the last VCBeginOperation. The VCEndOperation should be used to mark the end of an undo level if the tool completes as designed, while the VCAbortOperation should be used when the tool ends unexpectedly or if the user manually aborts the tool. VCAbortOperation ensures that there is no residual undo information left.

See Also VCEndOperation, VCAbortOperation

{button ,AL(`Creating a User Tool',0,`',`')} Task Guide Examples

VCButton

Version 1.2

Description Send a mouse button click message to Corel Visual CADD.

Declaration

C/C++: extern "C" void WINAPI VCButton(short iButton, short iState);

Visual Basic: Declare Sub VCButton Lib "VCMAIN32.DLL" (ByVal iButton As Integer, ByVal iState As Integer)

Delphi: procedure VCButton(iButton: Integer; iState: Integer); far;

Parameters *iState* - represents the up or down state, where 0 denotes and 1 denotes down.
iButton - the specific button number on the puck starting with 0 and ending with 15 to represent a 16 button digitizer puck.

Notes This is analogous to the user pressing a button on the pointing device within Corel Visual CADD. Depending on what script assignments have been made to each button, different events may occur. Keep in mind that button 1 (number 0) and the right button (number 1 or 2 depending on the mouse) have special meanings within Corel Visual CADD and will be interpreted as such. These of course also depend on the cursor location in the drawing and the current, if any, active commands.

See Also VCMouseMove, VCMouseMove2, VCLButtonDown, VCLButtonDown2

VCChangeSelected

Version	1.2
Description	Changes all selected entities to the line attributes specified in EAttr
Declaration	
C/C++:	extern "C" void WINAPI VCChangeSelected(EAttr* ea);
Visual Basic:	Declare Sub VCChangeSelected Lib "VCTOOL32.DLL" (ea As EAttr)
Delphi:	procedure VCChangeSelected(var ea: EAttr); far;
Parameters	*EAttr* - user defined type containing the entity properties.
Notes	VCChangeSelected operates on all currently selected entities and immediately applies the line properties defined in the EAttr parameter. For the structure of EAttr, see Appendix C.
See Also	VCApplySettingsToCurrentEntity, VCDuplicate, VCDuplicateWithTransform, VCChangeSelected2

{button ,AL(`Duplicating an Entity with Transformation;Modifying Existing Entities',0,`',`')} Task Guide Examples

VCChangeSelected2

Version 2.0

Description Operate the same as VCChangeSelected except allows the current color setting to overwrite any values.

Declaration

C/C++: extern "C" void WINAPI VCChangeSelected2(EAttr* ea, short iUseSymbolColor);

Visual Basic: Declare Sub VCChangeSelected2 Lib "VCTOOL32.DLL" (ea As EAttr, ByVal iUseSymbolColor As Integer)

Delphi: procedure VCChangeSelected2(var ea: EAttr; iUseSymbolColor: Integer); far;

Parameters *EAttr* - user defined type containing the linetype properties used for the change.
IUseSymbolColor - determines if the current color setting is used or the setting in Eattr.
0 - do not change the color value.
1 - use the value supplied in EAttr.

Notes Symbols can be placed using the current color or maintain the colors used during creation. VCChangeSelectd2 gives the option to adjust this color setting after the entity has been placed. If on, the individual entity colors used when creating the symbol will be maintained otherwise the current color setting is applied.

See Also VCApplySettingsToCurrentEntity, VCDuplicate, VCDuplicateWithTransformVCChangeSelected

{button ,AL(`Duplicating an Entity with Transformation;Modifying Existing Entities',0,`',`')} Task Guide Examples

VCChangeView

Version	2.0
Description	Changes to another view of the active drawing.
Declaration	
C/C++	extern "C" void WINAPI VCChangeView(short* iError, long hWnd_);
Visual Basic	Declare Sub VCChangeView Lib "VCMAIN32.DLL" (iError As Integer, ByVal hWnd_ As Long)
Delphi	procedure VCChangeView(var iError: Integer; hWnd_: Longint); far;
Parameters	*hWnd* - the Windows HWND containing the view.
Notes	Corel Visual CADD allows for multiple views of a drawing. Each of these views is placed into a separate MDI Window within the Corel Visual CADD frame. The view can be changed by moving to the Window containing the desired view.
See Also	VCNewView, VCFirstView , VCNextView

VCChangeView3D

Version 1.2

Description Moves the view eye position while maintaining the target position.

Declaration

C/C++: extern "C" void WINAPI VCChangeView3D(short* iError, short iCode, double dFact);

Visual Basic: Declare Sub VCChangeView3D Lib "VCMAIN32.DLL" (iError As Integer, ByVal iCode As Integer, ByVal dFact As Double)

Delphi: procedure VCChangeView3D(var iError: Integer; iCode: Integer; dFact: Double);

Parameters *iCode* - determines the direction to move the viewers location.
0 - CHANGE_VIEW3D_LEFT
1 - CHANGE_VIEW3D_RIGHT
2 - CHANGE_VIEW3D_UP
3 - CHANGE_VIEW3D_DOWN
dFact - the distance to move in the specified direction.

Notes When creating 3D views of a drawing, three parameters are required: view type, eye location, and viewed position. VCSetProjection3D determines the view type and thus how the lines will be viewed in relation to each other, that is flat, parallel or perspective. VCSetView3D establishes the absolute 3D coordinate of the viewers eye and thus the level of perspective exaggeration used or the relative size of the view. VCChangeView3D can allow the users view point to be moved incrementally in certain directions and thus creates a limited "walk-through" functionality. 3D views can be viewed in wireframe or with Corel Visual CADD's built in quick shading. VCSet3DDisplay provides the ability to view the drawing as a quick shade and VCSet3DQShadeOptions determines the level of quick shade when the drawing is shaded.

See Also VCAddLine3D, VCAddPoint3D, VCAddPolygon3D, VCGetCurrentEntityNormal3D, VCSet3DQShadeOptions, VCSet3DDisplay, VCSetView3D, VCSetProjection3D

VCChar

Version 1.2

Description Sends a text character to the Corel Visual CADD program to initiate two letter commands or for coordinate entry. Acts as if user typed the characters directly through the Corel Visual CADD interface.

Declaration

C/C++: extern "C" void WINAPI VCChar(short c);

Visual Basic: Declare Sub VCChar Lib "VCMAIN32.DLL" (ByVal c As Integer)

Delphi: procedure VCChar(c: Integer); far;

Parameters *c* - the ASCII equivalent of a character that is to be sent to the Corel Visual CADD command parser.

Notes Any character sent to the command parser will be processed in whatever context it was received. For example, two consecutive letters will be interpreted as a two letter key command, a pair of coordinates will be seen as coordinate entry in the current tool and entry mode, and a single number will be interpreted as a direct distance entry for the current tool. Be aware that whatever has the focus at the time of the call will receive the character input, i.e. if a speedbar is currently active, it will receive the input and not the command line.

See Also VCGetCMDStr

{button ,AL(`Creating a User Tool;Utilizing a Custom Interface',0,`',`')} Task Guide Examples

VCClearAlertApp

Version 1.2

Description Clears the hWnd from the messaging registry list.

Declaration

C/C++: extern "C" void WINAPI VCClearAlertApp(short* iError, HWND hWnd);

Visual Basic: Declare Sub VCClearAlertApp Lib "VCMAIN32.DLL" (iError As Integer, ByVal hWnd As Integer)

Delphi: procedure VCClearAlertApp(var iError: Integer; hWnd: Integer); far;

Parameters *hWnd* - the HWND of the object to receive messages from Corel Visual CADD.

Notes To initialize the Windows messaging between Corel Visual CADD and an external application, the hWnd of some control or object must be sent to Corel Visual CADD using VCSetAlertApp. When registering the hWnd, the VCSetAlertApp code must specify which messages the application will receive. These can be added together to get multiple messages. For example, a VCSetAlertApp iCode of 12 would specify that the command line characters and abort messages would be sent to Corel Visual CADD. To handle these messages, the application must have specific code to handle a Windows message. In Visual BASIC this is handled by supplying code in the mousedown event for the control for each mouse down message sent by Corel Visual CADD. Corel Visual CADD is fairly intelligent about when to send this message and only send the message when a drawing point has been selected. This means that the user can issue snaps or use tracking without invoking the application code for the mousedown event. To retrieve the point the user selected in the drawing area, use VCGetUserToolLBDown which sets a Point2D of the last point picked. When trapping the user input, register the control with an iCode of either 0 (all messages) or 8 and add code to the control for keypress. When the keypress code is activated by the message from Corel Visual CADD, use VCGetCmdStr to retrieve the last keypress from Corel Visual CADD. Once the keypress has been determined, the application can act according to process the information or send it back for Corel Visual CADD to use with VCSetCmdStr. Once the application has completed with the messaging, use VCClearAlertApp to remove the application from the messaging registry.

See Also VCClearAlertApp, VCGetCmdStr, VCGetUserToolLBDown, VCSetAlertApp, VCSetUserTool

VCClearAlertAppDll

Version 2.0

Description Clears a DLL from the messaging registry.

Declaration

C/C++ extern "C" void WINAPI VCClearAlertAppDll(short* iError, char* DllName, char* NativeCmd);

Visual Basic Declare Sub VCClearAlertAppDll Lib "VCMAIN32.DLL" (iError As Integer, ByVal DllName As String, ByVal NativeCmd As String)

Delphi procedure VCClearAlertAppDll(var iError: Integer; DllName: PChar; NativeCmd:

Parameters *DLLName* - the name of the DLL to register.
NativeCmd - the native command name used to reference the tool operation.

Notes A new option available to version 2.0 of Corel Visual CADD is to make tools and interfaces in dynamic link libraries (DLL's). This interface to Corel Visual CADD provides all the functionality of the message based EXE tools that were used with version 1.x. Some advantages to DLL's over EXE are: a DLL shares the same memory space as Corel Visual CADD, once loaded into memory, a DLL will stay in memory until Corel Visual CADD closes, code can be run on load and different code can be run each time a function is called, no interface or HWND are required, no checking is required to see if Corel Visual CADD is running since it is the one calling the DLL, and several tools can be in one DLL without command line options necessary for EXE to achieve the same functionality.

Any tool is made up of several functions that handle each of the events passed by Corel Visual CADD. The old way was to use VCSetAlertApp to register a list of messages your user tool needed in order to function properly. This was limiting in many development languages like Visual BASIC because only certain controls could receive the needed messages and even those controls were limited by the number of messages they could handle. Even if all the needed messages were available they could accidentally be triggered if the interface was displayed on screen. Now, VCSetAlertAppDLL registers a group of exported functions in a DLL to be used instead relying on message handlers.

See Also VCClearAlertApp, VCSetAlertApp, VCSetAlertAppDll

VCClearDrawing

Version	1.2
Description	Clears the referenced drawing world after prompting the user for verification.
Declaration	
C/C++:	extern "C" void WINAPI VCClearDrawing(WORLDHANDLE hW);
Visual Basic:	Declare Sub VCClearDrawing Lib "VCTOOL32.DLL" (ByVal hW As Long)
Delphi:	procedure VCClearDrawing(hW: Longint); far;
Parameters	*hW* - the WORLDHANDLE to reference open drawing worlds.
Notes	Clears the referenced drawing creating a "blank slate" for the user. The command erases all the entities in the drawing but maintains the current settings. The user is prompted for verification when VCClearDrawing is used, while with VCClearDrawingNoPrompt they are not. The drawing handle can be retrieved with a VCGetCurrWorld function.
See Also	VCClearDrawingNoPrompt, VCGetCurrWorld, VCNewWorld, VCIsDrawingDirty

VCClearDrawingNoPrompt

Version 1.2

Description Initiates command to clear the current drawing of all entities. Will not prompt user to verify the command.

Declaration

C/C++: extern "C" void WINAPI VCClearDrawingNoPrompt(WORLDHANDLE hW);

Visual Basic: Declare Sub VCClearDrawingNoPrompt Lib "VCTOOL32.DLL" (ByVal hW As Long)

Delphi: procedure VCClearDrawingNoPrompt(hW: Longint); far;

Parameters *hW* - the WORLDHANDLE to reference open drawing worlds.

Notes Clears the referenced drawing creating a "blank slate" for the user. The command erases all the entities in the drawing but maintains the current settings. Unlike the VCClearDrawing routine, the user is not prompted for verification. The drawing handle can be retrieved with a VCGetCurrWorld function.

See Also VCClearDrawing, VCGetCurrWorld, VCNewWorld, VCIsDrawingDirty, VCDestroyWorld

VCClearLayerProperties

Version 2.0.1

Description Clears all the layer property settings for the input layer.

Declaration

C/C++ extern "C" void WINAPI VCClearLayerProperties(short* iError, short iLayer);

Visual Basic Declare Sub VCClearLayerProperties Lib "VCMAIN32.DLL" (iError As Integer, ByVal iLayer As Integer)

Delphi procedure VCClearLayerProperties(var iError: Integer; iLayer: Integer); far;

Parameters *iLayer* - the layer index to clear from 0 to 1023.

Notes Layer properties were introduced into v2.0.1 allowing properties to be assigned by layer rather than by entity. For example, a layer can be set so all entities drawn on the layer will be a specific color, line type and line width. This will override the current properties settings when active. VCGetUseByLayerProperties is used to determine if the layer has active property settings while VCSetUseByLayerProperties allows an application to choose which properties to use. VCSetLayerProperties will set the values for the layer and VCClearLayerProperties turns the capability off and clears all associated values. It is important to keep track of the state of layer properties when modifying entities in the drawing. For example, if you set the color index using VCSetColorIndex but the layer properties are enabled the proper color may not get applied. Therefore when attempting to control the properties of entities as they are placed it is imperative that the application monitor the setting for by layer control as the information is being supplied by the API.

See Also VCGetLayerProperties, VCLayerHasProperties

VCClose

Version 1.2

Description Closes the drawing specified by the input handle.

Declaration

C/C++: extern "C" void WINAPI VCClose(WORLDHANDLE hW);

Visual Basic: Declare Sub VCClose Lib "VCMAIN32.DLL" (ByVal hW As Long)

Delphi: procedure VCClose(hW: Longint); far;

Parameters *hW* - the WORLDHANDLE to reference open drawing worlds.

Notes All opened drawings are referenced by an internal world handle. This handle can be retrieved by VCGetCurrWorld as each drawing screens receive focus. VCClose utilizes this handle to prompt the user if they want to save the file and then close the file with the current focus.

See Also VCGetCurrWorld, VCNewWorld, VCDestroyWorld

VCComputeArcMid

Version 1.2

Description Calculates the midpoint of an arc-length that lies on the arc.

Declaration

C/C++: extern "C" void WINAPI VCComputeArcMid(Point2D* dpC, Point2D* dpP0, Point2D* dpP2, Point2D* dpPreviousMid, Point2D* dpRet);

Visual Basic: Declare Sub VCComputeArcMid Lib "VCMAIN32.DLL" (dpC As Point2D, dpP0 As Point2D, dpP2 As Point2D, dpPreviousMid As Point2D, dpRet As Point2D)

Delphi: procedure VCComputeArcMid(var dpC: Point2D; var dpP0: Point2D; var dpP2: Point2D; var dpPreviousMid: Point2D; var dpRet: Point2D); far;

Parameters
dpC - the center point of the arc.
dpP0 - the first endpoint of the arc.
dpP1 - the second endpoint of the arc.
dpPreviousMid - a pick point for locating the midpoint.
dpRet - the returned midpoint

Notes When constructing an arc in code, the endpoint and midpoint of the arc are not always available. VCComputeArcMid takes the endpoints and center point of the arc to calculate the midpoint location. The resulting points can then be used directly by the VCAddArcEntity routine to add the curve to the drawing database.

See Also VCAddArcEntity, VCAddEllipticalArcEntity, VCComputeIntersection

VCComputeIntersection

Version 1.2

Description Calculates the intersection of two entities closest to a specified point.

Declaration

C/C++: extern "C" void WINAPI VCComputeIntersection(short* iError, ENTITYHANDLE l0, ENTITYHANDLE l1, Point2D* dpPick, Point2D* dpIntersect);

Visual Basic: Declare Sub VCComputeIntersection Lib "VCMAIN32.DLL" (iError As Integer, ByVal l0 As Long, ByVal l1 As Long, dpPick As Point2D, dpIntersect As Point2D)

Delphi: procedure VCComputeIntersection(var iError: Integer; l0: Longint; l1: Longint; var dpPick: Point2D; var dpIntersect: Point2D); far;

Parameters
l0 - the entityhandle of the first entity.
l1 - the entityhandle of the second entity.
dpPick - the point close to the desired intersection.
dpIntersect - returned as the calculated intersection.

Notes VCComputeIntersection will calculate the intersection of any non-linear entities in the database. The dpPick point is needed in order to narrow the search and to specify which intersection should be returned in the case of entities such as circles and curves which may intersect in more than one location.

See Also VCComputeArcMid, VCComputeSplineTangentPoints, VCSnapInt

VComputeSplineTangentPoints

Version 1.2

Description Calculates the spline tangent points given an array of points on the curve.

Declaration

C/C++: extern "C" void WINAPI VComputeSplineTangentPoints(short* iError, Point2D* pInput, short iCount, Point2D* pOutput, short* iOutCount);

Visual Basic: Declare Sub VComputeSplineTangentPoints Lib "VCMAIN32.DLL" (iError As Integer, pInput As Point2D, ByVal iCount As Integer, pOutput As Point2D, iOutCount As Integer)

Delphi: procedure VComputeSplineTangentPoints(var iError: Integer; var pInput Point2D; iCount: Integer; var pOutput: Point2D; var iOutCount: Integer); far;

Parameters

pInput-the input array of points on the curve.
ICount - the count for thenumber of input points.
POutput - the returned array of points for the spline curves.
IOutCount - the returned count number.

Notes To define a spline curve it is necessary to provide the tangent points corresponding to the vertex points on the curve. VCComputeSplineTangentPoint calculates these tangent construction points based on the input point array. The routine returns an array of points with two defined tangent points for every input point on the curve.

See Also VCComputeArcMid, VCComputeIntersection

VCCreateGraphicsHandle

Version 2.0

Description Creates a GRAHICSHANDLE for parsing inside complex entities.

Declaration

C/C++ extern "C" GRAPHICHANDLE WINAPI VCCreateGraphicHandle(short* iError);

Visual Basic Declare Function VCCreateGraphicHandle Lib "VCMAIN32.DLL" (iError As Integer) As Long

Delphi function VCCreateGraphicHandle(var iError: Integer):Longint; far;

Parameters *return* - the GRAPHICHANDLE created used for parsing operations.

Notes Some entities defined by several graphical objects, hatch patterns, fills, line types and fonts. For instance, a hatch pattern is defined by lines to make a useful pattern. These entities are not available for access through the standard database parsing routines provided. This is due to the fact that typically an application will not need this specific information. Most applications will need to simply parse the database and retrieve the entity information provided. In situations where a custom vector output file is being defined or to guide a CNC milling machine, the application may need to define all the vectors making up even the complex entities. The graphic handle method allow for this detailed parsing functionality.

In order to access the information an application should first create a graphics handle using VCCreateGraphicsHandle. This function creates a parsing list from the current entity if it is a graphic entity, hatch, fill, text or line type. The iError return will be > 0 if the current entity is not a graphic entity. The application can then parse the new set with VCFirstGraphic and VCNextGraphic. Any required information can be retrieved using any standard query function such as VCGetCurrentEntityPoint. The entity is considered read-only and only retrieval API routines may be utilized. The individual graphic entities can not be set with any command. After completing the parse the application should call VCDeleteGraphicHandle to destroy the created handle.

See Also VCDeleteGraphicsHandle, VCIsGraphic, VCFirstGraphic, VCNextGraphic

VCCreateMDIWindow

Version 1.2

Description Creates a new MDI drawing window within the Corel Visual CADD program frame.

Declaration

C/C++: extern "C" vbool WINAPI VCCreateMDIWindow(short* iError, short iNewMDIWindow);

Visual Basic: Declare Function VCCreateMDIWindow Lib "VCMAIN32.DLL" (iError As Integer, ByVal iNewMDIWindow As Integer) As Integer

Delphi: function VCCreateMDIWindow(var iError: Integer; iNewMDIWindow: Integer):Boolean; far;

Parameters *iNewMDIWindow* - determines whether to force the creation of the window.
0 - create only if there is no existing window or if it already has drawing information.
1 - create new window regardless of current window or drawing state.

Notes When opening or creating a new drawing, it is necessary to create a new MDI window in which Corel Visual CADD creates the new drawing world. If a new MDI window is not created, all edits or drawings opened will be placed in or on top of any existing drawing information.

See Also VCLoadDrawing, VCNewWorld, VCDestroyWorld

{button ,AL(`Utilizing a Custom Interface',0,`',`')} Task Guide Examples

VCCreateOleClass

Version 2.0

Description Creates a class from an OLE DLL.

Declaration

C/C++ extern "C" long WINAPI VCCreateOleClass(short* iError, char* OleDllName, char* OleClassName);

Visual Basic Declare Function VCCreateOleClass Lib "VCMAIN32.DLL" (iError As Integer, ByVal OleDllName As String, ByVal OleClassName As String) As Long

Delphi function VCCreateOleClass(var iError: Integer; OleDllName: PChar;OleClassName: PChar):Longint; far;

Parameters *OleDllName* - the name of the OLE DLL.
OleClassName - the class name to create.
returns - an index for the created class.

Notes An application can be created as an EXE, a Windows DLL or an OLE DLL. Each has advantages in functionality and interaction with the CAD engine. In addition, each is accessed through the Corel Visual CADD interface in different methods. An OLE DLL is a specialized link library containing methods and classes for controlling various operations. These DLL are specifically related to Visual Basic programmers. The OLE class allows a developer to create a class member function that can be directly run from the Corel Visual CADD interface allowing an application to take advantage of the performance increase associated with a DLL. In order to access this functionality the DLL and the class must be registered. VCCreateOLEClass registers the DLL and class. VCInvokeMethod will invoke the DLL method and VCDeleteOleClass will delete the registered DLL and class.

See Also VCDeleteOleClass, VCOleClassMethodInvoke

VCCreateSymbolDef

Version 1.2

Description Creates a new empty definition for building a symbol by adding entities.

Declaration

C/C++: extern "C" short WINAPI VCCreateSymbolDef(short* iError, char* pName);

Visual Basic: Declare Function VCCreateSymbolDef Lib "VCMAIN32.DLL" (iError As Integer, ByVal pName As String) As Integer

Delphi: function VCCreateSymbolDef(var iError: Integer; pName: PChar):Integer; far;

Parameters *pName* - the name of the symbol.
r*eturns* - the symbol index number.

Notes To create a new symbol from an external application, it is first necessary to create an empty symbol definition using VCCreateSymbolDef. VCCreateSymbolDef returns a symbol index which is be used by all the add entity routines that are used to build the symbol.

See Also VCGetSymbolIndex, VCGetSymName, VCGetSymbolName

{button ,AL(`Creating a Symbol;Loading a Symbol;Parsing a Symbol Definition;Placing a Symbol;Symbol Operations',0,`',`')} Task Guide Examples

VCCreateSymbolFromSelection

Version 2.0

Description Creates a symbol from the selected entities.

Declaration

C/C++ extern "C" void WINAPI VCCreateSymbolFromSelection(short* iError, char* szName, Point2D dpP);

Visual Basic Declare Sub VCCreateSymbolFromSelection Lib "VCMAIN32.DLL" (iError As Integer, ByVal szName As String, dpP As Point2D)

Delphi procedure VCCreateSymbolFromSelection (var iError: Integer; pName: PChar; var dpP: Point2D); far;

Parameters *szName* - the internal name to use for the symbol.
dpP - the handle point for the symbol.

Notes The API provides several methods for creating a symbol definition. The first method is to use VCCreateSymbolDef and then add entities to the new definition. This works well in situations where the symbol is being created externally from a set of parameters. In certain situation it is necessary to build the symbol from entities already existing in the drawing database. In these cases an application can actually parse the definition and recreate the symbol by adding the appropriate entities. This generally is not desirable as the application must build cases for each possible entity type. VCCreateSymbolFromSelection allows an application to directly build the symbol form a selection set of existing entities. The application can select the entities through code with VCSetCurrentSelected or as a result of user action. In either case the symbol is then built internally with the given name and handle placement point.

See Also VCCreateSymbolDef

VCCrossingSelect

Version 1.2

Description Selects any objects passing through or contained entirely in the specified window.

Declaration

C/C++: extern "C" void WINAPI VCCrossingSelect(Point2D* dpP0, Point2D* dpP1);

Visual Basic: Declare Sub VCCrossingSelect Lib "VCMAIN32.DLL" (dpP0 As Point2d, dpP1 As Point2d)

Delphi: procedure VCCrossingSelect(var dpP0: Point2D; var dpP1: Point2D); far;

Parameters *dpP0* - the coordinates of one corner of the window.
dpP1 - the coordinates of the second corner of the window.

Notes Operates the same as the select crossing tool except allows for input points from the external application. The application can process the points from a mouse down event or code in the coordinates for the selection routine.

See Also VCSelectCrossing, VCWindowSelect

VCDeInitPrintMode

Version	2.0
Description	De-Initializes the print routines for use outside the Corel Visual CADD interface.
Declaration	
C/C++	extern "C" void WINAPI VCDeInitPrintMode(short* iError);
Visual Basic	Declare Sub VCDeInitPrintMode Lib "VCDLG32.DLL" (iError As Integer)
Delphi	procedure VCDeInitPrintMode(var iError: Integer); far;
Parameters	No additional parameters are used for this subroutine.
Notes	When creating a custom interface that utilizes the Corel Visual CADD print routines, an application must initialize the mode on start and terminate it on close. The API provides access to the both the print and plot dialogs in which Corel Visual CADD handles all the output as if it were part of the interface by simply displaying the built in dialogs. The second method allows the application to create all the command and bypass the Corel Visual CADD interface. When using the first dialog method simply use VCInitDialogs and VCTerminateDialogs. When using the second method the initialization is handled by VCInitPrintMode and the de-initialization is handled by VCDeInitPrintMode.
See Also	VCInitPrintMode

VCDeleteCurrentEntityUserData

Version 1.2

Description Deletes the user data record at the specified index.

Declaration

C/C++: extern "C" void WINAPI VCDeleteCurrentEntityUserData(short* iError, short iIndex);

Visual Basic: Declare Sub VCDeleteCurrentEntityUserData Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer)

Delphi: procedure VCDeleteCurrentEntityUserData(var iError: Integer; iIndex: Integer); far;

Parameters *iIndex* - the index number within the current entity where the data is stored.

Notes User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of the C variable types double, float, long, or short. In addition to these types, a user defined type of "chunk" may also be stored. A chunk may be any size and is simply a pointer to a memory location. The size of the chunk is also passed so Corel Visual CADD can retrieve the appropriate amount of data from the specified memory location. Whenever using user data, an application must set a user data name in order to protect private data and to ensure that different applications do not interfere with other applications data. VCSetUserDataName is provided for this purpose, while VCGetUserDataName checks the current user data name. The name needs to be set only one time before adding any user data. The VCAddCurrentEntityUserData* calls always append the new variable as the last user data variable. The VCSetCurrentEntityUserData* calls add the user data variable to the index specified in the call, provided that there are indeed that many indices already attached, and will overwrite any existing user data at that index. User data may also be attached to the drawing header by using VCSetHeaderUserData and then attaching the appropriate user data. Once VCNextEntity or any other current entity selections are used, the user data calls will again be used on the current entity. VCDeleteCurrentEntityUserData deletes the user data from the current entity.

See Also VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble , VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong , VCAddCurrentEntityUserDataShort, VCAddCurrentEntityUserDataChunk, VCSetCurrentEntity, VCSetHeaderUserData, VCFirstEntity, VCNextEntity

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCDeleteGraphicsHandle

Version 2.0

Description Creates a GRAHICSHANDLE after parsing inside complex entities.

Declaration

C/C++ extern "C" void WINAPI VCDeleteGraphicHandle(short* iError, GRAPHICHANDLE hG);

Visual Basic Declare Sub VCDeleteGraphicHandle Lib "VCMAIN32.DLL" (iError As Integer, ByVal hG As Long)

Delphi procedure VCDeleteGraphicHandle(var iError: Integer; hG: Longint); far;

Parameters hG - the GRAPHICSHANDLE to delete.

Notes Some entities defined by several graphical objects, hatch patterns, fills, line types and fonts. For instance, a hatch pattern is defined by lines to make a useful pattern. These entities are not available for access through the standard database parsing routines provided. This is due to the fact that typically an application will not need this specific information. Most applications will need to simply parse the database and retrieve the entity information provided. In situations where a custom vector output file is being defined or to guide a CNC milling machine, the application may need to define all the vectors making up even the complex entities. The graphic handle method allow for this detailed parsing functionality.

In order to access the information an application should first create a graphics handle using VCCreateGraphicsHandle. This function creates a parsing list from the current entity if it is a graphic entity, hatch, fill, text or line type. The iError return will be > 0 if the current entity is not a graphic entity. The application can then parse the new set with VCFirstGraphic and VCNextGraphic. Any required information can be retrieved using any standard query function such as VCGetCurrentEntityPoint. The entity is considered read-only and only retrieval API routines may be utilized. The individual graphic entities can not be set with any command. After completing the parse the application should call VCDeleteGraphicHandle to destroy the created handle.

See Also VCCreateGraphicsHandle, VCIsGraphic, VCFirstGraphic, VCNextGraphic

VCDeleteOleClass

Version	2.0
Description	Creates a class from an OLE DLL.
Declaration	
C/C++	extern "C" void WINAPI VCDeleteOleClass(short* iError, long id);
Visual Basic	Declare Sub VCDeleteOleClass Lib "VCMAIN32.DLL" (iError As Integer, ByVal id As Long)
Delphi	procedure VCDeleteOleClass(var iError: Integer; id: Longint); far;
Parameters	id - the internal ID given to the OLE class with VCCreateOleClass.
Notes	An application can be created as an EXE, a Windows DLL or an OLE DLL. Each has advantages in functionality and interaction with the CAD engine. In addition, each is accessed through the Corel Visual CADD interface in different methods. An OLE DLL is a specialized link library containing methods and classes for controlling various operations. These DLL are specifically related to Visual Basic programmers. The OLE class allows a developer to create a class member function that can be directly run from the Corel Visual CADD interface allowing an application to take advantage of the performance increase associated with a DLL. In order to access this functionality the DLL and the class must be registered. VCCreateOLEClass registers the DLL and class. VCInvokeMethod will invoke the DLL method and VCDeleteOleClass will delete the registered DLL and class.
See Also	VCCreateOleClass, VCOleClassMethodInvoke

VCDeletePopupMenu

Version 1.2

Description Toggles the display of the default commands on the current tools pop-up menu.

Declaration

C/C++: extern "C" void WINAPI VCDeletePopupMenu(vbool tf);

Visual Basic: Declare Sub VCDeletePopupMenu Lib "VCTOOL32.DLL" (ByVal tf As Integer)

Delphi: procedure VCDeletePopupMenu(tf: Boolean); far;

Parameters *tf* - set according to whether the default pop-up commands should be displayed.
0 - do not display.
1 - display the default commands.

Notes While pop-up menus can be defined independently by the user, VCAddPopupCommand allows a native command to be added to the pop-up menu of the currently active tool for only the current session of that tool. When the tool is no longer active, any commands added to the tool will be lost and need to be re-added if required for the next instance of that tool. If the pop-up needs to be cleared of all default commands, VCDeletePopupMenu will remove all the existing defaults for the current instance of the tool. VCDeletePopupMenu will not affect commands added with VCAddPopupCommand. Remember that these commands only work on the current tool i.e. there must be a tool active in order to add to or delete the contents of the pop-up menu.

See Also VCAddPopupCommand, Custimizing Corel Visual CADD, Custimizing Mouse Menus

{button ,AL(`Creating a User Tool;Using the Corel Visual CADD Interface',0,`',`')} Task Guide Examples

VCDestroyWorld

Version	1.2
Description	Destroys a drawing world and frees allocated memory.
Declaration	
C/C++:	extern "C" void WINAPI VCDestroyWorld(WORLDHANDLE hW);
Visual Basic:	Declare Sub VCDestroyWorld Lib "VCMAIN32.DLL" (ByVal hW As Long)
Delphi:	procedure VCDestroyWorld(hW: Longint); far;
Parameters	*hW* - the worldhandle of the world to be destroyed.
Notes	When a world is created, whether for another MDI window or for a window in another application, a handle is created for referencing the drawing. When the window is removed the world must be destroyed in order to free its memory by calling VCDestroyWorld. When a world is created via VCNewWorld, a WORLDHANDLE is returned and should be used when you need to destroy the drawing world.
See Also	VCNewWorld, VCGetCurrWorld

{button ,AL(`Creating a User Tool;Using the Corel Visual CADD Interface;Utilizing a Custom Interface',0,`',`')} Task Guide Examples

VCDimDirectionMode

Version 2.0

Description The dimension direction is the orientation used when measuring a distance and drawing a dimension line.

Declaration

C/C++ extern "C" void WINAPI VCDimDirectionMode(short iMode);

Visual Basic Declare Sub VCDimDirectionMode Lib "VCMAIN32.DLL" (ByVal iMode As Integer)

Delphi procedure VCDimDirectionMode(iMode: Integer); far;

Parameters *i* - the value of the dimension line direction.
1 - DIMALIGNED
2 - DIMHORIZONTAL
3 - DIMVERTICAL
4 - DIMATANANGLE

Notes Measured distances are projected onto the dimension direction. *Horizontal* - Only the horizontal component of the entity is measured. *Vertical* - Only the vertical component of the entity is measured. *Aligned* - The dimension line is placed parallel to the entity. Aligned dimensions always represent the true length of the entity. *Angle* - Sets the dimension to a specified angle. The distance measured is the length of the entity projected onto the defined angle.

See Also VCGetDimLineAngleVCGetDimLineDirect

VCDimGetDimMode
VCDimSetDimMode

Version 1.2

Description Determine whether dimensions are to be placed as individual dimension, cumulative dimension, or as a partitioned dimension, and how grouped dimensions are related.

Declaration

C/C++:
```
extern "C" short WINAPI VCDimGetDimMode(short* iError);
extern "C" void WINAPI VCDimSetDimMode(short* iError, short b);
```

Visual Basic:
```
Declare Function VCDimGetDimMode Lib "VCMAIN32.DLL" (iErr As Integer) As Integer
Declare Sub VCDimSetDimMode Lib "VCMAIN32.DLL" (iErr As Integer, ByVal b As Integer)
```

Delphi:
```
function VCDimGetDimMode(var iError: Integer):Integer; far;
procedure VCDimSetDimMode(var iError: Integer; b: Integer); far;
```

Parameters *b* - the mode value.
1 - DIMMODESINGLE
2 - DIMMODECUMULATIVE
3 - DIMMODEPARTITIONED

Notes Single dimensions are placed one at a time, as individual entities. Once a single dimension is placed, the dimension command is completed. Cumulative places a sequence of dimensions, each originating from the same point or baseline. Partitioned places a string or chain of connected dimensions, placed end-to-end. Dimension lines are collinear for the entire chain.

See Also VCGetDimLineDirect, VCDimGetDimExtStretch, VCDimGetDimProximity, VCAddLinearDimensionEntity, VCAddAngularDimensionEntity

VCDimGetDimExtStretch
VCDimSetDimExtStretch

Version 1.2

Description Stretches the below section of the extension line to fill the gap between the Offset distance and the dimension line.

Declaration

C/C++:
```
extern "C" short WINAPI VCDimGetDimExtStretch(short* iError);
extern "C" void WINAPI VCDimSetDimExtStretch(short* iError, short b);
```

Visual Basic:
```
Declare Function VCDimGetDimExtStretch Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCDimSetDimExtStretch Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCDimGetDimExtStretch(var iError: Integer):Integer; far;
procedure VCDimSetDimExtStretch(var iError: Integer; b: Integer); far;
```

Parameters *b* - extension stretch status
0 - No Stretch
1 - Stretch

See Also VCAddLinearDimensionEntity, VCDimGetDimProximity, VCGetDimExtAbove, VCGetDimExtBelow, VCGetDimExtOffset, VCGetUnitConversionFactor

VCDimGetDimProximity
VCDimSetDimProximity

Version 1.2

Description Once turned on, dimension lines are placed a fixed distance away from the dimensioned object equal to the Below distance plus the Offset distance.

Declaration

C/C++:
```
extern "C" short WINAPI VCDimGetDimProximity(short* iError);
extern "C" void WINAPI VCDimSetDimProximity(short* iError, short b);
```

Visual Basic:
```
Declare Function VCDimGetDimProximity Lib "vcmain32.dll" ( iError As Integer) As Integer
Declare Sub VCDimSetDimProximity Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCDimGetDimProximity(var iError: Integer):Integer; far;
procedure VCDimSetDimProximity(var iError: Integer; b: Integer); far;
```

Parameters *b* - dimension proximity status
0 - on.
1 - off.

See Also VCDimGetDimExtStretch, VCGetDimtExtAbove, VCGetDimExtBelow, VCGetDimExtOffset

VCDispatchCommand

Version	2.0
Description	Sends the current command value to Corel Visual CADD.
Declaration	
C/C++	extern "C" void WINAPI VCDispatchCommand();
Visual Basic	Declare Sub VCDispatchCommand Lib "VCTOOL32.DLL" ()
Delphi	procedure VCDispatchCommand; far;
Parameters	No additional parameters are used in this routine.
Notes	All command entries and direct entry point values are entered through the Corel Visual CADD command line. Normally, as commands are entered in the interface Corel Visual CADD automatically recognizes and dispatches these to the appropriate command sequence. When working through the API an application can force the command through the event handler with VCDispatchCommand and VCDispatchPoint. The application sets the command sequence with VCSetCmdStr and then forces the entry with these commands. Typically, the API will recognize the command entry and not need to be forced.
See Also	VCDispatchNextPoint, VCGetCmdStr

VCDispatchNextPoint

Version	2.0
Description	Sends the current point to Corel Visual CADD.
Declaration	
C/C++	extern "C" void WINAPI VCDispatchNextPoint();
Visual Basic	Declare Sub VCDispatchNextPoint Lib "VCTOOL32.DLL" ()
Delphi	procedure VCDispatchNextPoint; far;
Parameters	No additional parameters are used in this routine.
Notes	All command entries and direct entry point values are entered through the Corel Visual CADD command line. Normally, as commands are entered in the interface Corel Visual CADD automatically recognizes and dispatches these to the appropriate command sequence. When working through the API an application can force the command through the event handler with VCDispatchCommand and VCDispatchPoint. The application sets the command sequence with VCSetCmdStr and then forces the entry with these commands. Typically, the API will recognize the command entry and not need to be forced.
See Also	VCDispatchNextCommand, VCGetCmdStr

VCDistToString

Version 1.2

Description Converts a given distance to a formatted string.

Declaration

C/C++: extern "C" short WINAPI VCDistToString(short* iError, char* pS, double* pD);

Visual Basic: Declare Function VCDistToString Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String, pD As Double) As Integer

Delphi: function VCDistToString(var iError: Integer; pS: PChar; var pD: Double):Integer; far;

Parameters *pS* - the returned string.
pD - the distance.
Returns - the number of characters in the returned string.

Notes When displaying distances, the output must be in the same units as the user has set in the numeric settings tab. This maintains a consistent look across applications and prevents user confusion that may occur if several different display formats are used. The supplied distance must be in inches, as is the case with all Corel Visual CADD API calls. Remember to use VCGetUnitConversionFactor to return a multiplier that can be used to convert the values based on the current unit setting in Corel Visual CADD. VCDistToString returns the number of chararacters in a distance string.

See Also VCStringToAngle, VCStringToAngle, VCStringToDist, VCGetUnitConversionFactor

VCDllRun

Version 2.0

Description Runs the function in a specified DLL.

Declaration

C/C++ extern "C" void WINAPI VCDllRun(short* iError, char* DllName, char* FunctionName, char* CommandLine);

Visual Basic Declare Sub VCDllRun Lib "VCMAIN32.DLL" (iError As Integer, ByVal DllName As String, ByVal FunctionName As String, ByVal CommandLine As String)

Delphi procedure VCDllRun(var iError: Integer; DllName: PChar; FunctionName: PChar;

Parameters DllName - the name of the DLL
FunctionName - the name of the function within the DLL
CommandLine - the command line argument for the function. This can be NULL.

Notes Corel Visual CADD supports applications written as a DLL directly in the interface. This allows applications to be built as general tool sets into a DLL with exported functions. In the Corel Visual CADD interface these routines can then be accessed by the end user through assigning a script. The exported functions can have only a character string command line.

See Also VCGetExeName, VCRunNested, VCRun

VCDrawCurrentEntity

Version	1.2
Description	Forces the current entity to be drawn.
Declaration	
C/C++:	extern "C" void WINAPI VCDrawCurrentEntity(short* iError);
Visual Basic:	Declare Sub VCDrawCurrentEntity Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCDrawCurrentEntity(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	After adding a new entity to the database, it is necessary to draw the entity on screen so the user may see the result. After the entity has been added it must be made current by using VCLastEntity, as it will be the last entity in the drawing, then VCDrawCurrentEntity can be used to draw the current entity.
See Also	VCDrawCurrentEntityXOR, VCGetCurrentEntityHandle

{button ,AL(`Adding a Continuous Entity;Adding a Hatch/Fill Entity;Adding a Single Entity;Adding a Text Entity;Applying Settings to an Entity',0,`',`')} Task Guide Examples

VCDrawCurrentEntityXOR

Version	1.2
Description	Forces the current entity to be drawn in XOR mode thus enabling a rubberband effect.
Declaration	
C/C++:	extern "C" void WINAPI VCDrawCurrentEntityXOR(short* iError);
Visual Basic:	Declare Sub VCDrawCurrentEntityXOR Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCDrawCurrentEntityXOR(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	When making tools that provide a rubberbanding preview of what the construction will look like, a means by which to create this dynamic feedback is needed. An entity can be added to the drawing database and edited according to mouse movements. After each of these updates the entity must be redrawn using VCDrawCurrentEntityXOR so the old image will be removed and be replaced with the updated image.
See Also	VCDrawCurrentEntity,VCGetCurrentEntityHandle, VCTools

{button ,AL(`Creating a User Tool',0,`',`')} Task Guide Examples

VCDrawToDC

Version 2.0

Description Sets a DC for displaying the current drawing zoom.

Declaration

C/C++ extern "C" void WINAPI VCDrawToDC(short* iError, short hDC_, long bottom, long left, long right, long top);

Visual Basic Declare Sub VCDrawToDC Lib "VCMAIN32.DLL" (iError As Integer, ByVal hDC_ As Integer, ByVal bottom As Long, ByVal left As Long, ByVal right As Long, ByVal top As Long)

Delphi procedure VCDrawToDC(var iError: Integer; hDC_: Integer; bottom: Longint; left: Longint; right: Longint; top: Longint); far;

Parameters
hDC - the handle for the Windows device context.
bottom - the screen value for the bottom edge of the DC.
top - the screen value for the top edge of the DC.
left - the screen value for the left edge of the DC.
right - the screen value for the right edge of the DC.

Notes Provides a quick and direct method for displaying the current view to a Window device context. The API will draw the current view in the active world or viewport and display the vectors in a device context. The routine requires the application to define the device context along with a bounding rect for the device boundary. These are passed in screen coordinates as separate parameters.

See Also VCNewWorld

VCDuplicate

Version	1.2
Description	Makes a copy of the specified entity, replicating its current settings.
Declaration	
C/C++:	extern "C" void WINAPI VCDuplicate(short* iError, ENTITYHANDLE lH);
Visual Basic:	Declare Sub VCDuplicate Lib "VCMAIN32.DLL" (iError As Integer, ByVal lH As Long)
Delphi	Declare Sub VCDuplicate Lib "VCMAIN32.DLL" (iError As Integer, ByVal lH As Long)
Parameters	*lH* - the handle of the entity to be duplicated.
Notes	Normally when changing or editing entities in Corel Visual CADD it would be necessary to first query for each of the coordinates of the entity and then reintroduce the entity into the database with those coordinates. VCDuplicate does this by copying the specified entity with all its data points while still adopting all the current applicable settings. VCDuplicateWithTransform allows for scaling, movement and rotation of the copied entity without the need for other routines.
See Also	VCAddArcEntity, VCAddTextEntity, VCAddBezierEntity, VCAddCircleEntity, VCAddLineEntity, VCAddEllipseEntity, VCDuplicateWithTransformVCDuplicateToWorld

{button ,AL(`Duplicating an Entity;Duplicating an Entity with Transformation',0,`',`')} Task Guide Examples

VCDuplicateToWorld

Version 2.0

Description Duplicates an entity to a new drawing world.

Declaration

C/C++ extern "C" void WINAPI VCDuplicateToWorld(short* iError, ENTITYHANDLE hE, WORLDHANDLE TargetWorld);

Visual Basic Declare Sub VCDuplicateToWorld Lib "VCMAIN32.DLL" (iError As Integer, ByVal hE As Long, ByVal TargetWorld As Long)

Delphi procedure VCDuplicateToWorld(var iError: Integer; hE: Longint; TargetWorld:

Parameters hE - handle for the entity to duplicate
TargetWorld - world handle for the drawing to place the duplicated entity.

Notes Normally when changing or editing entities in Corel Visual CADD it would be necessary to first query for each of the coordinates of the entity and the re-introduce the entity into the database with those coordinates. VCDuplicate does this by copying the specified entity using all its data points while still adopting all the current applicable settings. VCDuplicateWithTransform allows for scaling, movement and rotation of the copied entity without the need for other routines. VCDuplicateToWorld effectively copies one entity to a new drawing within the current setting.

See Also VCDuplicate, VCDuplicateWithTransformation

VCDuplicateWithTransform

Version 1.2.1

Description Makes a copy of the specified entity, replicating its current settings while allowing for scaling, movement and rotation of the copied entity directly.

Declaration

C/C++: extern "C" void WINAPI VCDuplicateWithTransform(short* iError, ENTITYHANDLE lH, Point2D* dpTrans, Point2D* dpScale, double dAngle);

Visual Basic: Declare Sub VCDuplicateWithTransform Lib "VCMAIN32.DLL" (iError As Integer, ByVal lH As Long, dpTrans As Point2D, dpScale As Point2D, ByVal dAngle As Double)

Delphi procedure VCDuplicateWithTransform(var iError: Integer; lH: Longint; var dpTrans: Point2D; var dpScale: Point2D; dAngle: Double); far;

Parameters *lH* - the handle of the entity to be duplicated.
dpTrans - the coordinate pair distance to move the duplicated entity. Use 0,0 to keep same position.
dpScale - the X and Y scale factors to apply to the duplicated entity. Use X=1, Y=1 to keep the same scale.
dAngle - the angle to rotate the duplicate entity from the horizontal.

Notes Normally when changing or editing entities in Corel Visual CADD it would be necessary to first query for the coordinates of the entity and the re-introduce the entity into the database with those coordinates. VCDuplicate does this by copying the specified entity using all its data points while still adopting all the current applicable settings. VCDuplicateWithTransform allows for scaling, movement and rotation of the copied entity without the need for other routines. In order to control the rotation angle, the entity should first be transposed to the drawing origin. A rotation angle is then set by duplicating the new entity and finally transposing the entity back to the desired location.

See Also VCAddArcEntity, VCAddTextEntity, VCAddBezierEntity, VCAddCircleEntity, VCAddLineEntity, VCAddEllipseEntity, VCDuplicateVCDuplicateToWorld

{button ,AL(`Duplicating an Entity;Duplicating an Entity with Transformation',0,`',`')} Task Guide Examples

VCEditAbort
VCEditChange
VCEditComplete

Version 1.2

Description VCEditAbort is used to abort the current text edit and revert to the pre-edit text. VCEditChange sends a message to the drawing area to redraw the bounding box to approximate the new text line. VCEditComplete sends a message to the drawing area to replace the old text with the new text.

Declaration

C/C++:
```
extern "C" void WINAPI VCEditAbort();
extern "C" void WINAPI VCEditChange();
extern "C" void WINAPI VCEditComplete();
```

Visual Basic:
```
Declare Sub VCEditAbort Lib "VCTOOL32.DLL" ()
Declare Sub VCEditChange Lib "VCTOOL32.DLL" ()
Declare Sub VCEditComplete Lib "VCTOOL32.DLL" ()
```

Delphi:
```
procedure VCEditAbort; far;
procedure VCEditChange; far;
procedure VCEditComplete; far;
```

Parameters No additional parameters are used with this subroutine.

Notes Text is the only entity requiring editing or creation outside of the world context. Because of this, it requires special considerations when aborting, ending or changing occurs. VCEditAbort, VCEditComplete, and VCEditChange provide these functions and will update the screen accordingly.

See Also VCAddTextEntity, VCTools

VCEndOperation

Version 1.2

Description Marks an operation where an undo level ends.

Declaration

C/C++: extern "C" void WINAPI VCEndOperation(short* iError);

Visual Basic: Declare Sub VCEndOperation Lib "VCMAIN32.DLL" (iError As Integer)

Delphi: procedure VCEndOperation(var iError: Integer); far;

Parameters No additional parameters are used with this subroutine.

Notes Corel Visual CADD provides a set of user tool functions to build and create tools not directly supported in the interface. For example, a multi-line tool that automatically hatches or fills the segments. Since this tool is not provided directly in the Corel Visual CADD interface, it must be created through code to interact with the existing commands such as snaps and undo operations. In order for the tools to respond appropriately to undo operations it should set undo and redo levels during the operation. A complex entity tool, one that adds multiple entities such as the multi-line example, can allow each individual entity or instead the entire operation to be undone with a single user undo operation. This depends on the design criteria specified for the application. The level of undo is set with the VCBeginOperation and VCEndOperation API routines. An application should set the beginning of the undo level prior to adding any entities to the drawing database and finish the tool with an end operation. In certain situations, the tool may be aborted by the user typically by pressing the <ESC> key. An application should respond appropriately by aborting the undo level to return it to the state prior to the user tool operation. VCAbortOperation will handle this for the application. When used in conjunction with VCBeginOperation and VCEndOperation, VCAbortOperation will discard all undo information complied since the last VCBeginOperation. The VCEndOperation should be used to mark the end of an undo level if the tool completes as designed, while the VCAbortOperation should be used when the tool ends unexpectedly or if the user manually aborts the tool. VCAbortOperation ensures that there is no residual undo information left.

See Also VCBeginOperation

{button ,AL(`Creating a User Tool',0,`',`')} Task Guide Examples

VCEntityBreak

Version 1.2

Description Breaks the specified entity between the included points.

Declaration

C/C++: extern "C" void WINAPI VCEntityBreak(short* iError, ENTITYHANDLE lH, Point2D* dpP0, Point2D* dpP1);

Visual Basic: Declare Sub VCEntityBreak Lib "VCMAIN32.DLL" (iError As Integer, ByVal lH As Long, dpP0 As Point2D, dpP1 As Point2D)

Delphi: procedure VCEntityBreak(var iError: Integer; lH: Longint; var dpP0: Point2D; var dpP1: Point2D); far;

Parameters *lH* - the entity handle of the object to break.
dpP0 - the coordinates of the first break point.
dpP1 - the coordinates of the second break point.

Notes Entity break actually erases the specified entity and recreates two entities with the same properties. If the specified points don't actually lie on the entity, the break points will be calculated as the closest two points to those locations.

See Also VCGetCurrentEntityHandle

{button ,AL(`Database Operations',0,`',`')} Task Guide Examples

VCEntityExtents

Version 1.2

Description Returns the bounding rectangle of the specified entity.

Declaration

C/C++: extern "C" void WINAPI VCEntityExtents(short* iError, ENTITYHANDLE lH, Point2D* dpMin, Point2D* dpMax);

Visual Basic: Declare Sub VCEntityExtents Lib "VCMAIN32.DLL" (iError As Integer, ByVal lH As Long, dpMin As Point2D, dpMax As Point2D)

Delphi: procedure VCEntityExtents(var iError: Integer; lH: Longint; var dpMin: Point2D; var dpMax: Point2D); far;

Parameters *lH* - the Corel Visual CADD entity Handle of desired entity.
dpMin - the coordinates of the lower left corner of the entity bounding rectangle.
dpMax - the coordinates of the upper right corner of the entity bounding rectangle.

Notes The extents of an entity can be useful in determining where overlapping objects should be trimmed to prevent extraneous lines from obstructing the entity. The bounding box is also used by Corel Visual CADD to determine when an object has been clicked on to be selected. This can also be useful to determine if an entity would be selected if the user clicks within a certain drawing area.

See Also VCGetCurrentEntityPoint

{button ,AL(`Database Operations;Parsing the Database',0,`',`')} Task Guide Examples

VCEraseCursor

Version	1.2
Description	Erases the drawing cursor to eliminate cursor remnants from prior focus.
Declaration	
C/C++:	extern "C" void WINAPI VCEraseCursor(void);
Visual Basic:	Declare Sub VCEraseCursor Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCEraseCursor; far;
Parameters	No additional parameters are used with this subroutine.
Notes	Whenever a dialog or other application captures focus from the drawing area, the drawing cursor would remain in the drawing area unless the original cursor is erased. Using VCEraseCursor before returning to the drawing area will eliminate the extra "ghost" cursor that would be present from the previous cursor.
See Also	VCEraseRubber

VCEraseRubber

Version	1.2
Description	Removes the current rubberband or XOR image from the drawing area.
Declaration	
C/C++:	extern "C" void WINAPI VCEraseRubber();
Visual Basic:	Declare Sub VCEraseRubber Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCEraseRubber; far;
Parameters	No additional parameters are used with this subroutine.
Notes	While dragging an construction image in the drawing area the user may move the cursor outside the drawing area. In an window external to Corel Visual CADD the rubberband image will remain where the cursor last was in the drawing area. To erase this image call VCEraseRubber.
See Also	VCEraseCursor

VCFilterReset

Version 1.2

Description Resets the entity filter so it will accept all entities.

Declaration

C/C++: extern "C" void WINAPI VCFilterReset(short* iError);

Visual Basic: Declare Sub VCFilterReset Lib "VCMAIN32.DLL" (iError As Integer)

Delphi: procedure VCFilterReset(var iError: Integer); far;

Parameters No additional parameters are used with this subroutine.

Notes When using the entity filter to select particular entities or attributes it should be returned to it's default state with VCFilterReset so that the user can select entities normally.

See Also VCGetFilterKind, VCGetFilterKind2, VCGetFilterLayer, VCGetFilterLineType, VCGetFilterName, VCGetFilterWidth, VCSetFilterMatch, VCSetFilterActive

{button ,AL(`Database Operations;Parsing a Filtered Entity List;Parsing a Symbol Definition;Parsing an Expanded List;Parsing an On Screen List;Parsing the Database',0,`',`')} Task Guide Examples

VCFirstEntity

Version 1.2

Description Positions a pointer for entity operations to the first entity in the database.

Declaration

C/C++: extern "C" vbool WINAPI VCFirstEntity(short* iError, short* bKind);

Visual Basic: Declare Function VCFirstEntity Lib "VCMAIN32.DLL" (iError As Integer, bKind As Integer) As Integer

Delphi: function VCFirstEntity(var iError: Integer; var bKind: Integer):Boolean; far;

Parameters *bKind* - set by the function to what type of entity is now current.
Returns - 0 if not successful and 1 otherwise.

Notes Whenever querying entities for their particular properties, it is necessary to have a method to step through the drawing database and select which entity a given query will focus on. The API offers several utility parsing methods for flexibility in locating entities in the database. Each offers advantages in certain situations. VCFirst/NextEntity moves to the first entity in the database and then to each entity in the drawing database. VCFirst/NextEntityExpand parses the database as if the drawing file had been exploded. Every entity, including those in symbol definitions and hatch patterns are included in the VCFirst/NextEntityExpand search. VCFirst/NextOnScreen clips the drawing and allows for quick entity access to only those entities found on the screen at the present time. VCFirst/NextSelected parses only through the selection set. This method combined with a selection filter allow access to specific entities meeting a set of criteria quickly in the drawing database. If no entities exist for the method, the return value will be 0.

See Also VCNextEntity, VCLastEntity, VCFirstEntityExpand, VCNextEntityExpand, VCFirstOnScreen, VCNextOnScreen, VCFirstSelected, VCNextSelected, VCSetCurrentEntity, VCGetCurrentEntityHandle

{button ,AL(`Database Operations;Parsing a Filtered Entity List;Parsing a Symbol Definition;Parsing an Expanded List;Parsing an On Screen List;Parsing the Database',0,`',`')} Task Guide Examples

VCFirstEntityExpand

Version 1.2

Description Locates the first entity, even those within hatch and symbol definitions, in the drawing and makes it current.

Declaration

C/C++: extern "C" vbool WINAPI VCFirstEntityExpand(short* iError, short* bKind);

Visual Basic: Declare Function VCFirstEntityExpand Lib "VCMAIN32.DLL" (iError As Integer, bKind As Integer) As Integer

Delphi: function VCFirstEntityExpand(var iError: Integer; var bKind: Integer):Boolean; far;

Parameters *bKind* - set by the function to what type of entity is now current.
Returns - 0 if not successful and 1 otherwise.

Notes Whenever querying entities for their particular properties, it is necessary to have a method to step through the drawing database and select which entity a given query will focus on. The API offers several utility parsing methods for flexibility in locating entities in the database. Each offers advantages in certain situations. VCFirst/NextEntity moves to the first entity in the database and then to each entity in the drawing database. VCFirst/NextEntityExpand parses the database as if the drawing file had been exploded. Every entity, including those in symbol definitions and hatch patterns are included in the VCFirst/NextEntityExpand search. VCFirst/NextOnScreen clips the drawing and allows for quick entity access to only those entities found on the screen at the present time. VCFirst/NextSelected parses only through the selection set. This method combined with a selection filter allow access to specific entities meeting a set of criteria quickly in the drawing database. If no entities exist for the method, the return value will be 0.

See Also VCNextEntity, VCLastEntity, VCFirstEntity, VCNextEntityExpand, VCFirstOnScreen, VCNextOnScreen, VCFirstSelected, VCNextSelected, VCSetCurrentEntity

{button ,AL(`Database Operations;Parsing a Filtered Entity List;Parsing a Symbol Definition;Parsing an Expanded List;Parsing an On Screen List;Parsing the Database',0,`',`')} Task Guide Examples

VCFirstGraphic

Version 2.0

Description Positions a pointer for entity operations to the first graphic in the entity.

Declaration

C/C++ extern "C" vbool WINAPI VCFirstGraphic(short* iError, GRAPHICHANDLE hG);

Visual Basic Declare Function VCFirstGraphic Lib "VCMAIN32.DLL" (iError As Integer, ByVal hG As Long) As Integer

Delphi function VCFirstGraphic(var iError: Integer; hG: Longint):Boolean; far;

Parameters *hG* - the returned GRAPHICHANDLE for the current entity
Returns - 0 if not successful and 1 otherwise.

Notes Some entities are defined by several graphical objects, hatch patterns, fills, line types and fonts. For instance, a hatch pattern is defined by lines to make a useful pattern. These entities are not available for access through the standard database parsing routines provided. This is due to the fact that typically an application will not need this specific information. Most applications will need to simply parse the database and retrieve the entity information provided. In situations where a custom vector output file is being defined or to guide a CNC milling machine, the application may need to define all the vectors making up even the complex entities. The graphic handle method allow for this detailed parsing functionality.

In order to access the information an application should first create a graphics handle using VCCreateGraphicsHandle. This function creates a parsing list from the current entity if it is a graphic entity, hatch, fill, text or line type. The iError return will be > 0 if the current entity is not a graphic entity. The application can then parse the new set with VCFirstGraphic and VCNextGraphic. Any required information can be retrieved using any standard query function such as VCGetCurrentEntityPoint. The entity is considered read-only and only retrieval API routines may be utilized. The individual graphic entities can not be set with any command. After completing the parse the application should call VCDeleteGraphicHandle to destroy the created handle.

See Also VCCreateGraphicsHandle, VCDeleteGraphicsHandle, VCIsGraphic, VCNextGraphic

VCFirstOnScreen

Version 1.2

Description Locates the first entity in the current zoom and makes it current.

Declaration

C/C++: extern "C" vbool WINAPI VCFirstOnScreen(short* iError, short* bKind);

Visual Basic: Declare Function VCFirstOnScreen Lib "VCMAIN32.DLL" (iError As Integer, bKind As Integer) As Integer

Delphi: function VCFirstOnScreen(var iError: Integer; var bKind: Integer):Boolean; far;

Parameters *bKind* - set by the function to what type of entity is now current.
Returns - 0 if not successful and 1 otherwise.

Notes Whenever querying entities for their particular properties, it is necessary to have a method to step through the drawing database and select which entity a given query will focus on. The API offers several utility parsing methods for flexibility in locating entities in the database. Each offers advantages in certain situations. VCFirst/NextEntity moves to the first entity in the database and then to each entity in the drawing database. VCFirst/NextEntityExpand parses the database as if the drawing file had been exploded. Every entity, including those in symbol definitions and hatch patterns are included in the VCFirst/NextEntityExpand search. VCFirst/NextOnScreen clips the drawing and allows for quick entity access to only those entities found on the screen at the present time. VCFirst/NextSelected parses only through the selection set. This method combined with a selection filter allow access to specific entities meeting a set of criteria quickly in the drawing database. If no entities exist for the method, the return value will be 0.

See Also VCNextEntity, VCLastEntity, VCFirstEntityExpand, VCNextEntityExpand, VCFirstEntity, VCNextOnScreen, VCFirstSelected, VCNextSelected, VCSetCurrentEntity

{button ,AL(`Database Operations;Parsing a Filtered Entity List;Parsing a Symbol Definition;Parsing an Expanded List;Parsing an On Screen List;Parsing the Database',0,`',`')} Task Guide Examples

VCFirstSelected

Version 1.2

Description Locates the first selected entity and makes it current.

Declaration

C/C++: extern "C" vbool WINAPI VCFirstSelected(short* iError, short* bKind);

Visual Basic: Declare Function VCFirstSelected Lib "VCMAIN32.DLL" (iError As Integer, bKind As Integer) As Integer

Delphi: function VCFirstSelected(var iError: Integer; var bKind: Integer):Boolean; far;

Parameters *bKind* - set by the function to what type of entity is now current.
Returns - 0 if not successful and 1 otherwise.

Notes Whenever querying entities for their particular properties, it is necessary to have a method to step through the drawing database and select which entity a given query will focus on. The API offers several utility parsing methods for flexibility in locating entities in the database. Each offers advantages in certain situations. VCFirst/NextEntity moves to the first entity in the database and then to each entity in the drawing database. VCFirst/NextEntityExpand parses the database as if the drawing file had been exploded. Every entity, including those in symbol definitions and hatch patterns are included in the VCFirst/NextEntityExpand search. VCFirst/NextOnScreen clips the drawing and allows for quick entity access to only those entities found on the screen at the present time. VCFirst/NextSelected parses only through the selection set. This method combined with a selection filter allow access to specific entities meeting a set of criteria quickly in the drawing database. If no entities exist for the method, the return value will be 0.

See Also VCNextEntity, VCLastEntity, VCFirstEntityExpand, VCNextEntityExpand, VCFirstOnScreen, VCNextOnScreen, VCFirstEntity, VCGetCurrentEntityHandle, VCNextSelected, VCSetCurrentEntity

{button ,AL(`Database Operations;Parsing a Filtered Entity List;Parsing a Symbol Definition;Parsing an Expanded List;Parsing an On Screen List;Parsing the Database',0,`',`')} Task Guide Examples

VCFirstSelectedRF

Version 2.0

Description Positions a pointer to the first entity in the given reference frame.

Declaration

C/C++ extern "C" vbool WINAPI VCFirstSelectedRF(short* iError, long* hE);

Visual Basic Declare Function VCFirstSelectedRF Lib "VCMAIN32.DLL" (iError As Integer, hE As Long) As Integer

Delphi function VCFirstSelectedRF(var iError: Integer; var hE: Longint):Boolean;

Parameters *hE* - the entity handle for the reference frame to parse.
Returns - 0 if not successful and 1 otherwise.

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

VCFirstSelectedRF and VCNextSelectedRF allow an application to parse the entities inside the reference frame. Any values returned for coordinates, using routines such as VCGetCurrentEntityPoint, are returned in values corresponding to the active drawing not the frame entity. For example if a real world drawing is referenced into a paper space drawing, the values returned will represent the coordinates for the entity in the paper space drawing not the absolute coordinates from the real world drawing. When the absolute coordinates are desired the referenced file must be opened and parsed with other standard database routines.

See Also VCNextSelectedRF, VCGetCurrentEntityPoint

VCFirstView

Version	2.0
Description	Positions a pointer to the first view of the active drawing.
Declaration	
C/C++	extern "C" vbool WINAPI VCFirstView(short* iError);
Visual Basic	Declare Function VCFirstView Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Delphi	function VCFirstView(var iError: Integer):Boolean; far;
Parameters	*Returns* - 0 if not successful and 1 otherwise.
Notes	Corel Visual CADD supports multiple viewports for drawings and displays the views in separate Window frames. These views are created through the API with VCNewView. When working with drawings utilizing multiple viewports, an application can parse through the views to update specific views as needed. The viewports are treated as separate MDI windows are managed by the Corel Visual CADD frame.
See Also	VCNewView, VCNextView, VCZoomAllViews, VCZoomRegenAllViews ,

VCForceWidthOnAllEntities

Version 1.2

Description Changes the line width for each entity in the drawing database to the specified value.

Declaration

C/C++: extern "C" void WINAPI VCForceWidthOnAllEntities(short* iError, short iNewWidth);

Visual Basic: Declare Sub VCForceWidthOnAllEntities Lib "VCMAIN32.DLL" (iError As Integer, ByVal iNewWidth As Integer)

Delphi: procedure VCForceWidthOnAllEntities(var iError: Integer; iNewWidth: Integer); far;

Parameters *iNewWidth* - the new width index for the entities.

Notes Several utilitiy routines to accomplish specifics tasks are available directly in the API. Instead of parsing the database for each entity and then resetting the line width, VCForceWidthOnAllEntities will automatically change the line width to specified value. When outputting to certain printers it is desireable to increase the line width in order to improve the output quality. VCForceWidthOnAllEntities and VcIncremetnWidthOnAllEntities both facilitate this operation under a single routine.

See Also VCIncrementWidthOnAllEntities

VCGeneratePointsFromCurrentEntity

Version 1.2

Description Generates the specified number of points on a entity to an array's Point2D.

Declaration

C/C++: extern "C" void WINAPI VCGeneratePointsFromCurrentEntity(short* iError, Point2D* P, short* iCount, short iMax, double dStep);

Visual Basic: Declare Sub VCGeneratePointsFromCurrentEntity Lib "VCMAIN32.DLL" (iError As Integer, P As Point2D, iCount As Integer, ByVal iMax As Integer, ByVal dStep As Double)

Delphi: procedure VCGeneratePointsFromCurrentEntity(var iError: Integer; var P: Point2D; var iCount: Integer; iMax: Integer; dStep: Double); far;

Parameters *P* - a returned pointer to an array of Point2D's containing all the points of the current entity.
iCount - returned as the number of items in the P array as well as the number of points in the entity.
dStep - a parameter between 0 and 1 that specifies how many steps along the entity path to use for the calculation.
iMax - the size of the array passed to prevent an overflow of the array.

Notes Unlike the function VCGetCurrentEntityPoint, which retrieves the actual construction points of an entity, VCGeneratePointsFromCurrentEntity generates points on the entity. This function will generate as many points as are specified by the parameter iMax. These points are useful for generating bounds of entities as well as approximate intersections of complex entities such as Bezier and ellipses. The number of points calculated is determined by the fractional value of dStep, i.e. .10 would mean 10 steps while .50 would be two steps. iMax is a safeguard to prevent the overflow of the array.

See Also VCGetCurrentEntityPoint

VCGetAcadImportUnit
VCSetAcadImportUnit

Version 1.2

Description The default unit used when converting AutoCAD files.

Declaration

C/C++:
```
extern "C" BYTE WINAPI VCGetAcadImportUnit(short* iError);
extern "C" void WINAPI VCSetAcadImportUnit(short* iError, BYTE b);
```

Visual Basic:
```
Declare Function VCGetAcadImportUnit Lib "VCMAIN32.DLL" (iErr As Integer) As Integer
Declare Sub VCSetAcadImportUnit Lib "VCMAIN32.DLL" (iErr As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetAcadImportUnit(var iError: Integer):Integer; far;
procedure VCSetAcadImportUnit(var iError: Integer; b: Integer); far;
```

Parameters *b* - the units for conversion.
0 - ACAD_UNIT_INCH
1 - ACAD_UNIT_FEET
2 - ACAD_UNIT_MILL
3 - ACAD_UNIT_CENT
4 - ACAD_UNIT_METER

Notes The Corel Visual CADD database stores values in inches while other formats such as AutoCAD use a unit-less database. When converting drawings from the DWG format it is necessary to specify the desired units in Corel Visual CADD.

See Also VCAcadRead, VCAcadReadWith3D, VCGetKeepAcadFontName, VCGetKeepGCDFontName, VCGetGCDDefaultHatchName

VCGetAllLayersEd
VCSetAllLayersEd

Version 1.2

Description Controls how Corel Visual CADD treats visible layers other than the current layer. Specifies if objects on all visible layers can be edited or only those on the current layer can be edited.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetAllLayersEd(short* iError);
extern "C" void WINAPI VCSetAllLayersEd(short* iError, vbool tf);
```

Visual Basic:
```
Declare Function VCGetAllLayersEd Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetAllLayersEd Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetAllLayersEd(var iError: Integer):Integer; far;
procedure VCSetAllLayersEd(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
1 - On (Checked)
0- Off (Unchecked)

See Also VCGetAllLayersSnap, VCGetLayerDisplay

VCGetAllLayersSnap
VCSetAllLayersSnap

Version 1.2

Description Controls how Corel Visual CADD treats visible layers other than the current layer. Specifies if snaps are made to all visible objects or to only those on the current layer.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetAllLayersSnap(short* iError);
extern "C" void WINAPI VCSetAllLayersSnap(short* iError, vbool tf);
```

Visual Basic:
```
Declare Function VCGetAllLayersSnap Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetAllLayersSnap Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetAllLayersSnap(var iError: Integer):Integer; far;
procedure VCSetAllLayersSnap(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetAllLayersEd, VCGetLayerDisplay

VCGetAngle
VCSetAngle

Version 1.2

Description Rotation angle setting for the rotate command. As with all angle settings, the angle value is specified in radians.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetAngle(short* iError);
extern "C" void WINAPI VCSetAngle(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetAngleBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetAngle Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double))
```

Delphi:
```
procedure VCGetAngleBP(var iError: Integer; var dRet: Double); far;
procedure VCSetAngle(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - double value representing the angle setting in radians

See Also VCRotate

VCGetArrowScreenStep
VCSetArrowScreenStep

Version 1.2

Description The cursor can be moved from both mouse and keyboard arrow keys. When using the arrow keys, the movement distance can be relative to the world or screen units. Specifies the screen distance that each arrow key will advance the cursor.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetArrowScreenStep(short* iError);
extern "C" void WINAPI VCSetArrowScreenStep(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetArrowScreenStep Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetArrowScreenStep Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer))
```

Delphi:
```
function VCGetArrowScreenStep(var iError: Integer):Integer; far;
procedure VCSetArrowScreenStep(var iError: Integer; i: Integer); far;
```

Parameters *I* -the screen distance to move in pixels

See Also VCGetArrowWorld, VCGetArrowWorldStep

VCGetArrowWorld
VCSetArrowWorld

Version 1.2

Description Determines whether screen units or world units are used to move the cursor when the arrow keys are used.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetArrowWorld(short* iError);
extern "C" void WINAPI VCSetArrowWorld(short* iError, vbool tf);
```

Visual Basic:
```
Declare Function VCGetArrowWorld Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetArrowWorld Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetArrowWorld(var iError: Integer):Boolean; far;
procedure VCSetArrowWorld(var iError: Integer; tf: Boolean); far;
```

Parameters *tf - toggle setting*
0 - Uses World Distance
1 - Uses Screen Distance

See Also VCGetArrowScreenStep, VCGetArrowWorldStep

VCGetArrowWorldStep
VCSetArrowWorldStep

Version 1.2

Description The cursor can be moved from both mouse and keyboard arrow keys. When using the arrow keys, the movement distance can be relative to the world or screen units. Specifies the "real world" incremental distance that each arrow key will advance the cursor.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetArrowWorldStep(short* iError);
extern "C" void WINAPI VCGetArrowWorldStepBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetArrowWorldStep(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetArrowWorldStepBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetArrowWorldStep Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetArrowWorldStepBP(var iError: Integer; var dRet: Double); far;
procedure VCSetArrowWorldStep(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the distance settings for movement

See Also VCGetArrowWorld, VCGetArrowScreenStep

VCGetAskZoomCenter
VCSetAskZoomCenter

Version 1.2

Description Determines if the user is prompted to pick a center point on screen before initiating the Zoom In or Zoom Out commands. The point becomes the center of the new view on the screen.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetAskZoomCenter(short* iError);
extern "C" void WINAPI VCSetAskZoomCenter(short* iError, vbool tf);
```

Visual Basic:
```
Declare Function VCGetAskZoomCenter Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetAskZoomCenter Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetAskZoomCenter(var iError: Integer):Integer; far;
procedure VCSetAskZoomCenter(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCZoomIn, VCZoomOut, VCGetZoomFactor

VCGetAtbDefLabel

Version 1.2

Description Attributes are non-graphical data that can be attached to a symbol. The attributes are made up of fields represented by a label and a value. The label is a name for the attribute field and is designated when creating the attribute. The value is the value of the attribute field and can be edited after creating the attribute. VCGetAtbDefLabel returns the label at the specified field index.

Declaration

C/C++: extern "C" short WINAPI VCGetAtbDefLabel(short* iError, char* szName, char* szLabel, short iRec);

Visual Basic: Declare Function VCGetAtbDefLabel Lib "VCMAIN32.DLL" (iError As Integer, ByVal szName As String, ByVal szLabel As String, ByVal iRec As Integer) As Integer

Delphi: function VCGetAtbDefLabel(var iError: Integer; var szName: PChar; var szLabel: PChar; iRec: Integer):Integer; far;

Parameters *szName* **-** the internal name of the attribute
szLabel - the returned label name
iRec - the field index for the label

See Also VCGetAtbDefRecordCount, VCGetAtbDefValue, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbCount, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecLabel, VCGetCurEntAtbRecValue

{button ,AL(`Attribute Manipulation;Retrieving Attributes',0,`',`')} Task Guide Examples

VCGetAtbDefRecordCount

Version 1.2

Description Attributes are non-graphical data that can be attached to a symbol. The attribute are made up of fields represented by a label and a value. The label is a name for the attribute field and is designated when creating the attribute. The value is the value of the attribute field and can be edited after creating the attribute. VCGetAtbDefRecordCount returns the number of fields in the attribute definition.

Declaration

C/C++: extern "C" short WINAPI VCGetAtbDefRecordCount(short* iError, char* szName);

Visual Basic: Declare Function VCGetAtbDefRecordCount Lib "VCMAIN32.DLL" (iError As Integer, ByVal szName As String) As Integer

Delphi: function VCGetAtbDefRecordCount(var iError: Integer; var szName:String):Integer; far;

Parameters *szName* - the internal name of the attribute

See Also VCGetAtbDefLabel, VCGetAtbDefValue, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbCount, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecLabel, VCGetCurEntAtbRecValue

{button ,AL(`Attribute Manipulation;Retrieving Attributes',0,`',`')} Task Guide Examples

VCGetAtbDefValue

Version 1.2

Description Attributes are non-graphical data that can be attached to a symbol. The attribute are made up of fields represented by a label and a value. The label is a name for the attribute field and is designated when creating the attribute. The value of the attribute field and can be edited after creating the attribute. VCGetAtbDefValue returns the value at the specified field index.

Declaration

C/C++: extern "C" short WINAPI VCGetAtbDefValue(short* iError, char* szName, char* Value, short iRec);

Visual Basic: Declare Function VCGetAtbDefValue Lib "VCMAIN32.DLL" (iError As Integer, ByVal szName As String, ByVal Value As String, ByVal iRec As Integer) As Integer

Delphi: function VCGetAtbDefValue(var iError: Integer; var szName: PChar; var Value: PChar; iRec: Integer):Integer; far;

Parameters *szName* - the internal name of the attribute
szLabel - the label name
Value - the string value for the field
iRec - the field index for the label

See Also VCGetAtbDefLabel, VCGetAtbDefRecordCount, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbCount, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecLabel, VCGetCurEntAtbRecValue

{button ,AL(`Attribute Manipulation;Retrieving Attributes',0,`',`')} Task Guide Examples

VCGetAtbFont
VCSetAtbFont

Version 1.2

Description Sets the font that will be used for attributes. Special font characteristics, such as bold, italic and underline, are not available for attributes.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetAtbFont(short* iError, char* pS);
extern "C" void WINAPI VCSetAtbFont(short* iError, char* pS);
```

Visual Basic:
```
Declare Function VCGetAtbFont Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String) As Integer
Declare Sub VCSetAtbFont Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String)
```

Delphi:
```
function VCGetAtbFont(var iError: Integer; var pS: PChar):Integer; far;
procedure VCSetAtbFont(var iError: Integer; var pS: PChar); far;
```

Parameters *pS* - the attribute font name

See Also VCGetAtbDefLabel, VCGetAtbDefRecordCount, VCGetAtbDefValue, VCGetAtbInternalName, VCGetCurEntAtbCount, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecLabel, VCGetCurEntAtbRecValue

{button ,AL(`Attribute Manipulation;Retrieving Attributes',0,`',`')} Task Guide Examples

VCGetAtbHeight
VCSetAtbHeight

Version 2.0

Description Specifies the attribute text height.

Declaration

C/C++

```
extern "C" void WINAPI VCGetAtbHeight(short* iError, double* dRet);
extern "C" void WINAPI VCSetAtbHeight(short* iError, double dRet);
```

Visual Basic

```
Declare Sub VCGetAtbHeight Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetAtbHeight Lib "VCMAIN32.DLL" (iError As Integer, ByVal dRet As Double)
```

Delphi

```
procedure VCGetAtbHeight(var iError: Integer; var dRet: Double); far;
procedure VCSetAtbHeight(var iError: Integer; dRet: Double); far;
```

Parameters

Notes Unlike many other Windows programs, Corel Visual CADD measures text height in real world units, specifically inches, instead of points.

See Also VCGetUnitConversionFactor, VCGetAtbDefLabel, VCGetAtbDefRecordCount, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbCount, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecLabel, VCGetCurEntAtbRecValue

{button ,AL(`Attribute Manipulation;Retrieving Attributes',0,`',`')} Task Guide Examples

VCGetAtbInternalName

Version 1.2.1

Description In order to access the attribute labels and values, the internal name of the attribute must be known. VCGetAtbInternalName returns the internal name from the file name.

Declaration

C/C++: extern "C" short WINAPI VCGetAtbInternalName(short* iError, char* pFileName, char* pReturn);

Visual Basic: Declare Function VCGetAtbInternalName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pFileName As String, ByVal pReturn As String) As Integer

Delphi: function VCGetAtbInternalName(var iError: Integer; var pFileName: PChar; var pReturn: PChar):Integer; far;

Parameters *pFileName* - the path and name for the attribute file
pReturn - the internal name for the attribute

Notes Attributes are non-graphical data that can be attached to a symbol. The attribute are made up of fields represented by a label and a value. The label is a name for the attribute field and is designated when creating the attribute. The value is the value of the attribute field and can be edited after creating the attribute. Attributes, like symbols, can be saved to disk for use in other drawings.

See Also VCGetAtbDefLabel, VCGetAtbDefRecordCount, VCGetAtbDefValue, VCGetAtbFont, VCGetCurEntAtbCount, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecLabel, VCGetCurEntAtbRecValue

{button ,AL(`Attribute Manipulation;Retrieving Attributes',0,`',`')} Task Guide Examples

VCGetAutoFillet
VCSetAutoFillet

Version 1.2

Description Specifies if corners are filleted automatically when the continuous line and continuous double line commands are used. In double line commands, the current radius at the interior intersection of each inside corner is used.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetAutoFillet(short* iError);
extern "C" void WINAPI VCSetAutoFillet(short* iError, vbool tf);
```

Visual Basic:
```
Declare Function VCGetAutoFillet Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetAutoFillet Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetAutoFillet(var iError: Integer):Integer; far;
procedure VCSetAutoFillet(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCFillet, VCLineContinuous, VCGetFilletPreview,VCGetFilletRad

VCGetAutoSave
VCSetAutoSave

Version 1.2

Description When auto save is active, Corel Visual CADD will save the drawing to a backup file at specified intervals. It will not overwrite the current file with new information, instead it will save the file with a .VBK file extension. To load a file with a .VBK extension on it, change the extension to .VCD, then load the file into Corel Visual CADD.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetAutoSave(short* iError);
extern "C" void WINAPI VCSetAutoSave(short* iError, vbool tf);
```

Visual Basic:
```
Declare Function VCGetAutoSave Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetAutoSave Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetAutoSave(var iError: Integer):Integer; far;
procedure VCSetAutoSave(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetAutoSaveSec

VCGetAutoSaveSec
VCSetAutoSaveSec

Version 1.2

Description The number of seconds between automatic backup. Corel Visual CADD will not back up at the specified interval if a dialog box is open or a tool is active. The backup will be postponed until the dialog is closed or the tool is inactive.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetAutoSaveSecs(short* iError);
extern "C" void WINAPI VCSetAutoSaveSecs(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetAutoSaveSecs Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetAutoSaveSecs Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetAutoSaveSecs(var iError: Integer):Integer; far;
procedure VCSetAutoSaveSecs(var iError: Integer; i: Integer); far;
```

Parameters *i* - seconds between saves

See Also VCGetAutoSave

VCGetBackgroundColor
VCSetBackgroundColor

Version 1.2

Description Specifies the drawing environment background color. Choosing a background color changes only the way the drawing appears on screen. Because Corel Visual CADD does not print or plot the background, the output is unaffected.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetBackgroundColor(short* iError);
extern "C" void WINAPI VCSetBackgroundColor(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetBackgroundColor Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetBackgroundColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetBackgroundColor(var iError: Integer):Integer; far;
procedure VCSetBackgroundColor(var iError: Integer; i: Integer); far;
```

Parameters *i*- color setting for background between 0 and 15

See Also VCGetColorIndex, VCGetCursorColor

VCGetBackwardsRedraw
VCSetBackwardsRedraw

Version 1.2

Description When redrawing the display after a zoom or redraw command, Corel Visual CADD draws the entities in the order they were placed in the database. By utilizing backwards redraw, the objects will be redrawn last to first. This also changes the order for database parsing commands such as VCFirstEntity and VCNextEntity. They will parse the database in reverse order.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetBackwardsRedraw(short* iError);
extern "C" void WINAPI VCSetBackwardsRedraw(short* iError, vbool tfBRD);
```

Visual Basic:
```
Declare Function VCGetBackwardsRedraw Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetBackwardsRedraw Lib "VCMAIN32.DLL" (iError As Integer, ByVal tfBRD As Integer)
```

Delphi:
```
function VCGetBackwardsRedraw(var iError: Integer):Integer; far;
procedure VCSetBackwardsRedraw(var iError: Integer; tfBRD: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCFirstEntity, VCNextEntity, VCZoomRegen

VCGetChamferDist1
VCSetChamferDist1

Version 1.2

Description A chamfer creates a line from a point on one line to a point on another line a specified distance from the real or projected intersection of those lines and trims each line to this additional line. Sets the first chamfer distance.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetChamferDist1(short* iError);
extern "C" void WINAPI VCGetChamferDist1BP(short* iError, double* dRet);
extern "C" void WINAPI VCSetChamferDist1(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetChamferDist1BP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetChamferDist1 Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetChamferDist1BP(var iError: Integer; var dRet: Double); far;
procedure VCSetChamferDist1(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - double value representing the distance

See Also VCGetChamferDist2, VCAutoFillet, VCFillet, VCGetFilletRad

VCGetChamferDist2
VCSetChamferDist2

Version 1.2

Description A chamfer creates a line from a point on one line to a point on another line a specified distance from the real or projected intersection of those lines, and trims each line to this additional lines. Sets the second chamfer distance.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetChamferDist2(short* iError);
extern "C" void WINAPI VCGetChamferDist2BP(short* iError, double* dRet);
extern "C" void WINAPI VCSetChamferDist2(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetChamferDist2BP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetChamferDist2 Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetChamferDist2BP(var iError: Integer; var dRet: Double); far;
procedure VCSetChamferDist2(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - double value representing the distance

See Also VCGetChamferDist1, VCGetAutoFillet, VCFillet, VCGetFilletRad

VCGetClosestSegment

Version 1.2

Description Finds the closest line segment to a specified point on a given entity.

Declaration

C/C++: extern "C" void WINAPI VCGetClosestSegment(short* iError, Point2D* p2d, ENTITYHANDLE lH, Point2D* dpP0, Point2D* dpP1);

Visual Basic: Declare Sub VCGetClosestSegment Lib "VCMAIN32.DLL" (iError As Integer, p2d As Point2D, ByVal lH As Long, dpP0 As Point2D, dpP1 As Point2D)

Delphi: procedure VCGetClosestSegment(var iError: Integer; var p2d: Point2D; lH: Longint; var dpP0: Point2D; var dpP1: Point2D); far;

Parameters
p2d - the point from which to find the closest segment.
lH - the entityhandle of the entity containing multiple segments.
dpP0 - the coordinates of the first end of the line.
dpP1 - the coordinates of the second end of the line.

Notes To reference the geometry of a continuous line, for example, it is sometimes necessary to know which segment of the entity to which the reference are made. Finding the perpendicular, for instance, with VCLinePerpPoint requires the two endpoints of the segment for which the perpendicular is to reference. VCGetClosestSegment provides this line segment and will allow these endpoints to be used in other calculations.

See Also VCGetUserToolLBDown, VCLinePerpPoint,

VCGetCMDId

Version 1.2

Description Returns the command id for a given native command.

Declaration

C/C++: extern "C" long WINAPI VCGetCMDId(short* iError, char* pNative);

Visual Basic: Declare Function VCGetCMDId Lib "VCMAIN32.DLL" (iError As Integer, ByVal pNative As String) As Long

Delphi: function VCGetCMDId(var iError: Integer; var pNative: PChar):Longint; far;

Parameters *pNative* - the native command name.

Returns - the command id.

Notes Many API calls require the command id of a command. If the native command name is known, then VCGetCMDId will return that value. The command id may change from one version of Corel Visual CADD to the next, so it is a good idea to always determine the command id at run time for any API calls. For a list of native commands, look in the NATIVE.CMD file or in Appendix A.

See Also VCGetCmdName, Corel Visual CADD Tool IDs

VCGetCmdName

Version 1.2.1

Description Returns the native command name for a given command id.

Declaration

C/C++: extern "C" short WINAPI VCGetCmdName(short* iError, short iCmdId, char* pName);

Visual Basic: Declare Function VCGetCmdName Lib "VCMAIN32.DLL" (iError As Integer, ByVal iCmdId As Integer, ByVal pName As String) As Integer

Delphi: function VCGetCmdName(var iError: Integer; iCmdId: Integer; var pName:

Parameters *iCmdId* - the command id.
pName - the returned native command name.
Returns - the number of characters in the returned pName.

Notes Many API calls return the command id of a command. However most command id's don't mean a lot to the average user. VCGetCmdName will return a more user friendly native command name.

See Also VCGetCMDId, Corel Visual CADD Tool IDs

VCGetCmdStr
VCSetCmdStr

Version 1.2

Description Retrieves the last command line input.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetCmdStr(short* iError, char* pS);
extern "C" void WINAPI VCSetCmdStr(short* iError, char* pS);
```

Visual Basic:
```
Declare Sub VCGetCmdStr Lib "VCMAIN32.DLL" ( iError As Integer, ByVal pS As String)
Declare Sub VCSetCmdStr Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String)
```

Delphi:
```
function VCGetCmdStr(var iError: Integer; var pS: String):Integer; far;
procedure VCSetCmdStr(var iError: Integer; var pS: PChar); far;
```

Parameters *pS* - the last command or coordinate entry processed by the Corel Visual CADD command parser.

Notes To initialize Windows messaging between Corel Visual CADD and an external application, the hWnd of some control or object must be sent to Visual CADD using VCSetAlertApp. When registering the hWnd the code must specify which messages the application will receive. These can be added together to get multiple messages. For example, an iCode of 12 would specify that the command line characters and abort messages would be sent to the external application. To handle these messages, the application must have code to handle a Windows message being sent whose hWnd is registered with VCSetAlertApp. In Visual BASIC, this is handled by supplying code in the mouse down event for the control specified for each mouse down message sent by Visual CADD. Corel Visual CADD is fairly intelligent about when to send this message and only send the message when a drawing point has been selected. This means that the user can issue snaps or use tracking without invoking the application code for the mouse down event. To retrieve the point the user selected in the drawing area, use VCGetUserToolLBDown, which sets a Point2D of the last point picked. When trapping the user input, register the control with an iCode of either 0 (all messages) or 8 (mouse down only) and add code to the external application for key press. When the key press code is activated, use VCGetCmdStr to retrieve the last command string from Corel Visual CADD. Once the key press has been determined, the application can act according to process the information or send it back for Corel Visual CADD to use with VCSetCmdStr. Once the application has completed the messaging, use VCClearAlertApp to remove the application from the messaging registry. For more information on iCode, please see Appendix C.

See Also VCClearAlertApp, VCGetUserToolLBDown, VCSetAlertApp, iCode, VCLButtonDown

{button ,AL(`Command Line Interaction;Creating a User Tool',0,`',`')} Task Guide Examples

VCGetCMPPath
VCSetCMPPath

Version 1.2

Description The default path for loading and saving Generic CADD components.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetCMPPath(short* iError, char* szPath);
extern "C" void WINAPI VCSetCMPPath(short* iError, char* szPath);
```

Visual Basic:
```
Declare Function VCGetCMPPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
As Integer
Declare Sub VCSetCMPPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
```

Delphi:
```
function VCGetCMPPath(var iError: Integer; var szPath: PChar):Integer; far;
procedure VCSetCMPPath(var iError: Integer; var szPath: PChar); far;
```

Parameters *sz* - string value for the path settings

See Also VCGetDWGPath, VCGetDXFPath, VCGetGCDPath, VCGetSYSPath, VCGetVCDPath, VCGetVCSPath, VCGetVCFPath

VCGetColorIndex
VCSetColorIndex

Version 1.2

Description The color for primary entity placements. Text, dimensions, hatches and fills each have their own property settings and are not affected by this subroutine.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetColorIndex(short* iError);
extern "C" void WINAPI VCSetColorIndex(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetColorIndex Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetColorIndex Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetColorIndex(var iError: Integer):Integer; far;
procedure VCSetColorIndex(var iError: Integer; i: Integer); far;
```

Parameters *i* - color index from 0 to 255

See Also VCGetLineTypeIndex, VCGetLineWidthIndex, VCGetLayerIndex, VCGetTextColor, VCGetDimItemColor,VCGetFillColor, VCGetHatchColor, VCGetDimItemColor

VCGetCommandAlias

Version 1.2

Description Returns the two-letter command for the specified command.

Declaration

C/C++: extern "C" short WINAPI VCGetCommandAlias(short i, char* szAlias);

Visual Basic: Declare Function VCGetCommandAlias Lib "VCMAIN32.DLL" (ByVal i As Integer, ByVal szAlias As String) As Integer

Delphi: function VCGetCommandAlias(i: Integer; var szAlias: String):Integer; far;

Parameters *i* - the command index of the desired command.
szAlias - the alias name of the command.

Notes Command aliases are simply the two-letter commands assigned to execute the command. While there is a default set of two-letter commands, they are user customizable by editing the text file ALIAS.CMD found in the Corel Visual CADD directory. Because of this, whenever sending commands to the command parser using VCChar it is generally a good idea to check the command alias of the command to prevent undesirable results. Although the aliases are traditionally called two-letter commands they can actually be up to three letters long as long as the first two letters do not conflict with an existing two-letter command.

See Also VCGetCommandCount, VCGetCommandDescription, VCGetCommandNative

{button ,AL(`Creating Command Aliases',0,`',`')} Task Guide Examples

VCGetCommandCount

Version	1.2
Description	Returns a count of all the available commands or tools.
Declaration	
C/C++:	extern "C" short WINAPI VCGetCommandCount();
Visual Basic:	Declare Function VCGetCommandCount Lib "VCMAIN32.DLL" () As Integer
Delphi:	function VCGetCommandCount:Integer; far;
Parameters	*Returns* - an integer representing the number of commands.
Notes	This can be used to determine the number of available commands at any time and to parse through each of them to determine their function.
See Also	VCGetCommandCount, VCGetCommandDescription, VCGetCommandNative

VCGetCommandDescription

Version 1.2

Description Returns the command description that appears in the status bar when the mouse moves over the command icon or menu item.

Declaration

C/C++: extern "C" short WINAPI VCGetCommandDescription(short i, char* szNative);

Visual Basic: Declare Function VCGetCommandDescription Lib "VCMAIN32.DLL" (ByVal i As Integer, ByVal szNative As String) As Integer

Delphi: function VCGetCommandDescription(i: Integer; var szNative: String):Integer; far;

Parameters *i* - the command index of the desired command.
szNative - the native command name.

Notes In Corel Visual CADD a prompt appears in the prompt area whenever the cursor passes over a button or a menu item describing what the command is or does. While these command descriptions will automatically be displayed in the prompt area, it may be helpful to retrieve the command description and display it elsewhere if a users may have a hard time seeing the prompt area.

See Also VCGetCommandCount, VCGetCommandDescription, VCGetCommandNative

VCGetCommandNative

Version 1.2

Description Returns the native command for the specified command.

Declaration

C/C++: extern "C" short WINAPI VCGetCommandNative(short i, char* szNative);

Visual Basic: Declare Function VCGetCommandNative Lib "VCMAIN32.DLL" (ByVal i As Integer, ByVal szNative As String) As Integer

Delphi: function VCGetCommandNative(i: Integer; var szNative: String):Integer; far;

Parameters

i - the command id of the desired command.
szNative - the native command name.

Notes Native commands are used in several user customizable files to specify which command is desired. These files include TOOLPAL.CST, MAINSBAR.CST, ALAIS.CMD, and all mouse popup menus. The native commands are not generally not used when programming with the API. However, they can be sent as macro commands in VCMacro or in VCSetFunkeyCmdString and can be useful to the Corel Visual CADD tool native command names.

See Also VCGetCommandCount, VCGetCommandDescription, VCGetCommandNative

{button ,AL(`Creating Command Aliases;Custom Commands;Custom Menus;Custom Mouse Menus;Custom Toolbars',0,`',`')} Task Guide Examples

VCGetConstPt
VCSetConstPt

Version 1.2

Description Option for displaying construction points. When working with entities, it is sometimes convenient to display the entity construction points to aid in snapping. Turning off the display will reduce the visual clutter and increase the speed of redraws. The number of construction points for an entity depend on the type of entity. A single line has two construction points, while a continuous Bezier curve can have many different construction points.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetConstPt(short* iError);
extern "C" void WINAPI VCSetConstPt(short* iError, vbool tf);
```

Visual Basic:
```
Declare Function VCGetConstPt Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetConstPt Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetConstPt(var iError: Integer):Integer; far;
procedure VCSetConstPt(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetHandlePt

{button ,AL(`Adding a Continuous Entity;Adding a Hatch/Fill Entity;Adding a Single Entity;Adding a Text Entity',0,`',`')} Task Guide Examples

VCGetCurEntAtbCount

Version 1.2

Description Attributes are non-graphical data that can be attached to a symbol. The attributes are made up of fields represented by a label and a value. The label is a name for the attribute field and is designated when creating the attribute. The value is the value of the attribute field and can be edited after creating the attribute. VCGetCurEntAtbCount returns the number of attributes attached to the current entity. To modify the attribute definition, use VCGetAtbDef* and VCSetAtbDefLabelValue.

Declaration

C/C++: extern "C" short WINAPI VCGetCurEntAtbCount(short* iError);

Visual Basic: Declare Function VCGetCurEntAtbCount Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCGetCurEntAtbCount(var iError: Integer):Integer; far;

Parameters *Return* - a count for the number of attached or embedded attributes.

See Also VCGetAtbDefLabel, VCGetAtbDefRecordCount, VCGetAtbDefValue, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecLabel, VCGetCurEntAtbRecValue, VCSetAtbDefLabelValue

{button ,AL(`Attribute Manipulation;Database Operations;Retrieving Attributes;Retrieving Entity Properties;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCGetCurEntAtbName

Version 1.2

Description Retrieves the internal name for the attached attribute on the current entity.

Declaration

C/C++: extern "C" short WINAPI VCGetCurEntAtbName(short* iError, short iWhichAtb, char* pStr);

Visual Basic: Declare Function VCGetCurEntAtbName Lib "VCMAIN32.DLL" (iError As Integer, ByVal iWhichAtb As Integer, ByVal pStr As String) As Integer

Delphi:

Parameters *iWhichAtb* - the index for the attribute.
PStr - the name of the attribute.
Returns - the length of the name.

Notes Several different attributes may be attached a symbol definition. This is reflected in the routine VCGetCurEntAtbCount, which counts the number of attributes attached to the current entity. VCGetCurEntAtbName allows an application to reference the name of each of these attached attributes. The information can then be modified or changed based on the attribute definition. To modify the attribute definition, use VCGetAtbDef* and VCSetAtbDefLabelValue.

See Also VCGetAtbDefLabel, VCGetAtbDefRecordCount, VCGetAtbDefValue, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecLabel, VCGetCurEntAtbRecValue, VCSetAtbDefLabelValue

{button ,AL(`Attribute Manipulation;Database Operations;Retrieving Attributes;Retrieving Entity Properties;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCGetCurEntAtbRecCount

Version 1.2

Description VCGetCurEntAtbRecCount returns the number of records in the attribute.

Declaration

C/C++: extern "C" short WINAPI VCGetCurEntAtbRecCount(short* iError, short iWhichAtb);

Visual Basic: Declare Function VCGetCurEntAtbRecCount Lib "VCMAIN32.DLL" (iError As Integer, ByVal iWhichAtb As Integer) As Integer

Delphi: function VCGetCurEntAtbRecCount(var iError: Integer; iWhichAtb: Integer):Integer; far;

Parameters *iWhichAtb* - the index for the attribute
Returns - a count for the number of records defining the attribute

Notes Attributes are non-graphical data that can be attached to a symbol. The attributes are made up of fields represented by a label and a value. The label is a name for the attribute field and is designated when creating the attribute. The value is the value of the attribute field and can be edited after creating the attribute. To modify the attribute definition, use VCGetAtbDef* and VCSetAtbDefLabelValue.

See Also VCGetAtbDefLabel, VCGetAtbDefRecordCount, VCGetAtbDefValue, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbCount, VCGetCurEntAtbRecLabel, VCGetCurEntAtbRecValue, VCSetAtbDefLabelValue

{button ,AL(`Attribute Manipulation;Database Operations;Retrieving Attributes;Retrieving Entity Properties;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCGetCurEntAtbRecLabel

Version 1.2

Description VCGetCurEntAtbRecLabel returns the label for the attribute at the field index.

Declaration

C/C++: extern "C" short WINAPI VCGetCurEntAtbRecLabel(short* iError, short iWhichAtb, short iWhichRec, char* pLabel);

Visual Basic: Declare Function VCGetCurEntAtbRecLabel Lib "VCMAIN32.DLL" (iError As Integer, ByVal iWhichAtb As Integer, ByVal iWhichRec As Integer, ByVal pLabel As String) As Integer

Delphi: function VCGetCurEntAtbRecLabel(var iError: Integer; iWhichAtb: Integer; iWhichRec: Integer; var pLabel: PChar):Integer; far;

Parameters *iWhichAtb* **-** attribute index
iWhichRec **-** the field index for the label
pLabel - the returned label

Notes Attributes are non-graphical data that can be attached to a symbol. The attributes are made up of fields represented by a label and a value. The label is a name for the attribute field and is designated when creating the attribute. The value is the value of the attribute field and can be edited after creating the attribute. To modify the attribute definition, use VCGetAtbDef* and VCSetAtbDefLabelValue.

See Also VCGetAtbDefLabel, VCGetAtbDefRecordCount, VCGetAtbDefValue, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbCount, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecValue , VCSetAtbDefLabelValue

{button ,AL(`Attribute Manipulation;Database Operations;Retrieving Attributes;Retrieving Entity Properties;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCGetCurEntAtbRecValue
VCSetCurEntAtbRecValue

Version 1.2

Description VCGetCurEntAtbRecValue returns the value for the attribute at the field index.

Declaration

C/C++: extern "C" short WINAPI VCGetCurEntAtbRecValue(short* iError, short iWhichAtb, short iWhichRec, char* pValue);
extern "C" void WINAPI VCSetCurEntAtbRecValue(short* iError, short iWhichAtb, short iWhichRec, char* pValue);

Visual Basic: Declare Function VCGetCurEntAtbRecValue Lib "VCMAIN32.DLL" (iError As Integer, ByVal iWhichAtb As Integer, ByVal iWhichRec As Integer, ByVal pValue As String) As Integer

Delphi: function VCGetCurEntAtbRecValue(var iError: Integer; iWhichAtb: Integer; iWhichRec: Integer; var pValue: PChar):Integer; far;

Parameters *iWhichAtb* - attribute index
iWhichRec - the field index for the label
pValue - the returned value

Notes Attributes are non-graphical data that can be attached to a symbol. The attributes are made up of fields represented by a label and a value. The label is a name for the attribute field and is designated when creating the attribute. The value is the value of the attribute field and can be edited after creating the attribute. To modify the attribute definition, use VCGetAtbDef* and VCSetAtbDefLabelValue.

See Also VCGetAtbDefLabel, VCGetAtbDefRecordCount, VCGetAtbDefValue, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbCount, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecLabel, VCSetAtbDefLabelValue

{button ,AL(`Attribute Manipulation;Database Operations;Retrieving Attributes;Retrieving Entity Properties;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCGetCurEntUserDataChunkSize

Version 1.2

Description Returns the size of the user data chunk specified by the index.

Declaration

C/C++: extern "C" short WINAPI VCGetCurEntUserDataChunkSize(short* iError, short iIndex);

Visual Basic: Declare Function VCGetCurEntUserDataChunkSize Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer) As Integer

Delphi: function VCGetCurEntUserDataChunkSize(var iError: Integer; iIndex:

Parameters *iIndex* - the index of the chunk user data.
Returns - the size in bytes of the specified chunk.

Notes When retrieving user data from drawing entities, the type of user data must be determined using VCGetCurrentEntityUserDataKind. If it happens to be a chunk it could be data of any size. To determine the size of the data in order to provide a correctly sized variable use VCGetCurEntUserDataChunkSize. After the variable has been created at the right size, the data can be retrieved with VCGetCurrentEntityUserDataChunk.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCGetCurrEntUserDataName

Version 1.2

Description Returns the User Data name for the entity information at the specified location.

Declaration

C/C++: extern "C" short WINAPI VCGetCurrEntUserDataName(short* iError, short iIndex, char* pName);

Visual Basic: Declare Function VCGetCurrEntUserDataName Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal pName As String) As Integer

Delphi: function VCGetCurrEntUserDataName(var iError: Integer; iIndex: Integer; pName: PChar):Integer; far;

Parameters *iIndex* - the index for the attached user data.
pName - the user data name.
Returns - the length of the string.

Notes User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of the C variable types double, float, long, or short. In addition to these types, a user defined type of "chunk" may also be stored. A chunk may be any size and is simply a pointer to a memory location. The size of the chunk is also passed so Corel Visual CADD can retrieve the appropriate amount of data from the specified memory location. Whenever using user data, an application must set a user data name in order to protect private data and to ensure that different applications do not interfere with the others data. VCSetUserDataName is provided for this purpose, while VCGetUserDataName checks the currently set user data name. The name needs to be set only one time before adding any user data. The VCAddCurrentEntityUserData* calls always append the new variable as the last user data variable. The VCSetCurrentEntityUserData* calls add the user data variable at the index specified in the call, provided that there are indeed that many indices already attached, and overwrite any existing user data at that index. As previously mentioned, user data may be attached to the drawing header. This is achieved by using VCSetHeaderUserData and then attaching the appropriate user data. Once VCNextEntity or any other current entity selections are used, the user data calls will again be used on the current entity.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID,VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCGetCurEntUserDataStringSize

Version 2.0

Description Retrieves the string size of a user data string value.

Declaration

C/C++ extern "C" short WINAPI VCGetCurEntUserDataStringSize(short* iError, short iIndex);

Visual Basic Declare Function VCGetCurEntUserDataStringSize Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer) As Integer

Delphi function VCGetCurEntUserDataStringSize(var iError: Integer; iIndex: Integer):Integer; far;

Parameters *iIndex* - the index of the chunk user data.
Returns - the size in bytes of the specified chunk.

Notes When retrieving user data from drawing entities, the type of user data must be determined using VCGetCurrentEntityUserDataKind. If it happens to be a chunk or string, it could be data of any size. To determine the size of the data in order to provide a correctly sized variable use VCGetCurEntUserDataChunkSize or VCGetCurEntUserDataStringSize. After the variable has been created at the right size, the data can be retrieved with VCGetCurrentEntityUserDataChunk or VCGetCurrentEntityUserDataString.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

VCGetCurrentEntity3DFlag0

Version 1.2.1

Description Determines if the current entity is a 3D entity.

Declaration

C/C++ extern "C" short WINAPI VCGetCurrentEntity3DFlag0(short* iError);

Visual Basic Declare Function VCGetCurrentEntity3DFlag0 Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi function VCGetCurrentEntity3DFlag0(var iError: Integer):Integer; far;

Parameters Returns - a flag whether the current entity is a 3D entity.
0 - it is not a 3D entity.
1 - it is a 3D entity.

Notes The Corel Visual CADD database supports several 3D entity types such as points, lines, continuous lines, polygons, symbols and blocks. When parsing the database an application can check if the active entity is a 3D entity.

See Also VCAcadReadWith3D, VCAddContinuousLine3DEntity, VCAddPoint3D, VCAddPolygon3D, VCAddSymbol3DEntity,

VCGetCurrentEntityArcData

Version 1.2

Description Retrieves relevant information about the geometry of an arc.

Declaration

C/C++: extern "C" void WINAPI VCGetCurrentEntityArcData(short* iError, Point2D* dpC, double* dRad, double* dStart, double* dSpan, double* dArcLength);

Visual Basic: Declare Sub VCGetCurrentEntityArcData Lib "VCMAIN32.DLL" (iError As Integer, dpC As Point2D, dRad As Double, dStart As Double, dSpan As Double, dArcLength As Double)

Delphi: procedure VCGetCurrentEntityArcData(var iError: Integer; var dpC: Point2D; var dRad: Double; var dStart: Double; var dSpan: Double; var dArcLength: Double); far;

Parameters
dpC - retrieves the center point of the arc.
dRad - retrieves the radius of the arc.
dStart - retrieves the start point as a radian measured from the 3 o'clock position.
dSpan - retrieves the end point as a radian measured from the 3 o'clock position.
dArcLength - retrieves the arc length of the arc span.

Notes Without this API it would be a matter of doing all the geometry calculations for arcs within external code routines. This provides all the basic geometry which would normally be provided in from the object information dialog. All angles in Corel Visual CADD are represented as radians not degrees. Therefor all angles will have to be converted to degrees if that is the applications preferred display format.

See Also VCGetCurrentEntityColor, VCGetCurrentEntityHandle, VCGetCurrentEntityKind, VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetCurrentEntityLineType, VCGetCurrentEntityLineWidth, VCSetCurrentEntity, VCAddArcEntity

VCGetCurrentEntityArea

Version 1.2

Description Calculates the area of the current entity using small differential line segments. In order to use VCGetCurrentEntityArea, the entity must be a closed bound entity, such as an unexploded rectangle, polygon, circle, or other object.

Declaration

C/C++: extern "C" void WINAPI VCGetCurrentEntityArea(short* iError, double dStep, double* dArea);

Visual Basic: Declare Sub VCGetCurrentEntityArea Lib "VCMAIN32.DLL" (iError As Integer, ByVal dStep As Double, dArea As Double)

Delphi: procedure VCGetCurrentEntityArea(var iError: Integer; dStep: Double; var

Parameters *dStep* - the length of the line segments to be used in calculating the area.

dArea - returned as the area of the entity.

Notes In cases where the area of an entity can't be easily calculated, VCGetCurrentEntityArea can calculate the area by constructing a polygon with sides of length dStep which approximates the area of the current entity. Obviously the smaller the step, the better the approximation however the slower the calculation will be.

See Also VCGetCurrentEntityDist, VCGetCurrentEntityLength, VCLineLength, VCMeasureArea, VCMeasureDistance,

VCGetCurrentEntityCloseContour
VCSetCurrentEntityCloseContour

Version 1.2

Description Joins the starting and ending points of continuous lines, double lines, and curves.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetCurrentEntityCloseContour(short* iError);
extern "C" void WINAPI VCSetCurrentEntityCloseContour(short* iError, vbool tf);
```

Visual Basic:
```
Declare Function VCGetCurrentEntityCloseContour Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetCurrentEntityCloseContour Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetCurrentEntityCloseContour(var iError: Integer):Boolean; far;
procedure VCSetCurrentEntityCloseContour(var iError: Integer; tf: Boolean); far;
```

Parameters

tf - toggle setting
1 - On (Checked)
0- Off (Unchecked)

Notes Use the Close Contour command to join the starting and ending points of multi-segmented lines or curves, when you want the connection to be trimmed (for straight segment objects) or smooth (for curves). For double lines, Close Contour joins and trims the starting and ending segments. For curves, the beginning and endpoints are joined, and the curve is made smooth at the joint. This command will also terminate a drawing command after the countour has been closed.

See Also VCGetAutoFillet

VCGetCurrentEntityColor

Version	1.2
Description	Retrieves the color index for the current entity.
Declaration	
C/C++:	extern "C" short WINAPI VCGetCurrentEntityColor(short* iError);
Visual Basic:	Declare Function VCGetCurrentEntityColor Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Delphi:	function VCGetCurrentEntityColor(var iError: Integer):Integer; far;
Parameters	*Returns* - the color index of the entity.
Notes	The entity is set as the current entity with VCSetCurrentEntity, VCFirstEntity, or VCNextEntity and will then allow the use of the VCGetCurrentEntity functions.
See Also	VCGetCurrentEntityHandle, VCGetCurrentEntityKind, VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetCurrentEntityLineType, VCGetCurrentEntityLineWidth, VCSetCurrentEntity, VCGetColorIndex

VCGetCurrentEntityDist

Version 1.2

Description Calculates the length of the current entity using line segments.

Declaration

C/C++: extern "C" void WINAPI VCGetCurrentEntityDist(short* iError, double dStep, double* dDist);

Visual Basic: Declare Sub VCGetCurrentEntityDist Lib "VCMAIN32.DLL" (iError As Integer, ByVal dStep As Double, dDist As Double)

Delphi: procedure VCGetCurrentEntityDist(var iError: Integer; dStep: Double; var

Parameters *dStep* - the length of the line segments to be used in calculating the area.

dDist - the returned length of the current entity.

Notes In cases where the length of an entity can't be easily calculated, VCGetCurrentEntityDist can calculate the area by constructing a continuous line with segments of length dStep which approximates the length of the current entity. The smaller the step, the better the approximation, however the slower the calculation will be.

See Also VCGetCurrentEntityArea, VCGetCurrentEntityArea, VCLineLenght, VCMeasureArea, VCMeasureDistance,

VCGetCurrentEntityHandle

Version 1.2

Description Retrieves a Corel Visual CADD entity handle so it may be retrieved at a later time.

Declaration

C/C++: extern "C" ENTITYHANDLE WINAPI VCGetCurrentEntityHandle(short* iError);

Visual Basic: Declare Function VCGetCurrentEntityHandle Lib "VCMAIN32.DLL" (iError As Integer) As Long

Delphi: function VCGetCurrentEntityHandle(var iError: Integer):Long; far;

Parameters *Returns* - the Corel Visual CADD entityhandle for the desired entity.

Notes The current entity is set with VCSetCurrentEntity, VCFirstEntity, VCNextEntity, or VCLastEntity. Each entity in Corel Visual CADD maintains a unique entity identifier (VCGetCurrentEntityUID) in order to track the entity. This is in addition to the dynamic entity handle (VCGetCurrentEntityHandle) which changes as entities are deleted and modified in the database. As entities are added to the drawing, both an entity handle and a UID are assigned to the entity. The entity handle will change as items are deleted and modified on the database while the UID will remain constant. Whenever linking entities to external databases or arrays, the application should utilize the UID due to its unchanging value with each entity. The entity handle is used when parsing the database or setting specific entities within the drawing session. The UID can should be audited prior to any external storage in order to ensure uniqueness in the ID.

See Also VCGetCurrentEntityColor, VCGetCurrentEntityUID, VCGetCurrentEntityKind, VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetCurrentEntityLineType, VCGetCurrentEntityLineWidth, VCGetCurrentEntityLineWidthValue, VCLastEntity, VCSetCurrentEntity

{button ,AL(`Duplicating an Entity;Duplicating an Entity with Transformation',0,`',`')} Task Guide Examples

VCGetCurrentEntityKind

Version 1.2

Description Retrieves the entity type from the current entity.

Declaration

C/C++: extern "C" short WINAPI VCGetCurrentEntityKind(short* iError);

Visual Basic: Declare Function VCGetCurrentEntityKind Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCGetCurrentEntityKind(var iError: Integer):Integer; far;

Parameters *Returns* - integer value representing the type of entity as follows. See Appendix A for a listing of entity types.

Notes The current entity is set with VCSetCurrentEntity, VCFirstEntity, VCNextEntity, or VCLastEntity

See Also VCFirstEntity, VCGetCurrentEntityColor, VCGetCurrentEntityUID, VCGetCurrentEntityHandle, VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetCurrentEntityLineType, VCGetCurrentEntityLineWidth, VCGetCurrentEntityLineWidthValue, VCLastEntity, VCNextEntity, VCSetCurrentEntity

{button ,AL(`Database Operations;Duplicating an Entity;Duplicating an Entity with Transformation;Parsing the Database',0,`',`')} Task Guide Examples

VCGetCurrentEntityLayer

Version 1.2

Description Retrieves the layer number of the current entity.

Declaration

C/C++: extern "C" short WINAPI VCGetCurrentEntityLayer(short* iError);

Visual Basic: Declare Function VCGetCurrentEntityLayer Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCGetCurrentEntityLayer(var iError: Integer):Integer; far;

Parameters *Returns* - the layer number.

Notes The current entity is set with VCSetCurrentEntity, VCFirstEntity, VCNextEntity, or VCLastEntity.

See Also VCFirstEntity, VCGetCurrentEntityColor, VCGetCurrentEntityUID, VCGetCurrentEntityHandle, VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetCurrentEntityLineType, VCGetCurrentEntityLineWidth, VCGetCurrentEntityLineWidthValue, VCLastEntity, VCNextEntity, VCSetCurrentEntity

{button ,AL(`Database Operations;Duplicating an Entity;Duplicating an Entity with Transformation;Parsing the Database',0,`',`')} Task Guide Examples

VCGetCurrentEntityLayerName

Version 1.2

Description Retrieves the layer name of the current entity.

Declaration

C/C++: extern "C" short WINAPI VCGetCurrentEntityLayerName(short* iError, char* pName);

Visual Basic: Declare Function VCGetCurrentEntityLayerName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pName As String) As Integer

Delphi: function VCGetCurrentEntityLayerName(var iError: Integer; var pName: String):Integer; far;

Parameters *pName* - set by the subroutine to be the name of the layer containing the entity.

Notes The current entity is set with VCSetCurrentEntity, VCFirstEntity, VCNextEntity or VCLastEntity. This allows subsequent use of the VCGetCurrentEntity* functions. If the current entity's layer is named, then VCGetCurrentEntityLayerName will return that string. If not, then VCGetCurrentEntityLayerName will return a 1 for iError and VCGetCurrentEntityLayer can be used to retrieve the layer index number.

See Also VCGetCurrentEntityColor, VCGetCurrentEntityHandle, VCGetCurrentEntityKind, VCGetCurrentEntityLayer,VCGetCurrentEntityLineType, VCGetCurrentEntityLineWidth, VCGetCurrentEntityLineWidthValue, VCLastEntity, VCSetCurrentEntity

VCGetCurrentEntityLength

Version 1.2

Description Retrieves the length of the current entity without point entry.

Declaration

C/C++: extern "C" void WINAPI VCGetCurrentEntityLength(short* iError, double dStep, double* dLength);

Visual Basic: Declare Sub VCGetCurrentEntityLength Lib "VCMAIN32.DLL" (iError As Integer, ByVal dStep As Double, dLength As Double)

Delphi: procedure VCGetCurrentEntityLength(var iError: Integer; dStep: Double; var dLength: Double); far;

Parameters *dStep* - the number of steps along the path.
dLength - the length of the entity as returned by Corel Visual CADD.

Notes In the case or non-linear entities, VCGetCurrentEntityLength provides an easy alternative to determine the path length of the entity without developing specific methods in external code for each entity. The length is actually calculated from small line segments that are calculated along the length of the entity at dStep increments. Although it is more accurate, more steps will also take more computation time and the accuracy required should be considered when determining a dStep value to be used. This routine differs from VCLineLength in that it does not accept input argument points and only calculates the length of the current entity measured directly from the first point to the last point of the entity.

See Also VCGetCurrentEntityColor, VCGetCurrentEntityHandle, VCGetCurrentEntityKind, VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetCurrentEntityLineType, VCGetCurrentEntityLineWidth, VCGetCurrentEntityLineWidthValue, VCLineLength, VCSetCurrentEntity

VCGetCurrentEntityLineType

Version	1.2
Description	Retrieves the line type number of the current entity.
Declaration	
C/C++:	extern "C" short WINAPI VCGetCurrentEntityLineType(short* iError);
Visual Basic:	Declare Function VCGetCurrentEntityLineType Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Delphi:	function VCGetCurrentEntityLineType(var iError: Integer):Integer; far;
Parameters	*Returns* - the line type number.
Notes	The current entity is set with VCSetCurrentEntity, VCFirstEntity, VCNextEntity, or VCLastEntity.
See Also	VCFirstEntity, VCGetCurrentEntityColor, VCGetCurrentEntityHandle, VCGetCurrentEntityKind, VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetCurrentEntityLineWidth, VCGetCurrentEntityLineWidthValue, VCLastEntity, VCNextEntity, VCSetCurrentEntity

VCGetCurrentEntityLineTypeName

Version 1.2

Description Retrieves the line type name of the current entity.

Declaration

C/C++: extern "C" short WINAPI VCGetCurrentEntityLineTypeName(short* iError, char* pName);

Visual Basic: Declare Function VCGetCurrentEntityLineTypeName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pName As String) As Integer

Delphi: function VCGetCurrentEntityLineTypeName(var iError: Integer; var pName: String):Integer; far;

Parameters *pName* - set by the subroutine to be the name of the line type of the entity.

Notes If the current entity's line type is named, VCGetCurrentEntityLineTypeName will return that string. If not, then VCGetCurrentEntityLineTypeName will return a 1 for iError and VCGetCurrentEntityLineType can be used to retrieve just the line type index number.

See Also VCFirstEntity, VCGetCurrentEntityColor, VCGetCurrentEntityHandle, VCGetCurrentEntityKind, VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetCurrentEntityLineWidth, VCGetCurrentEntityLineWidthValue, VCLastEntity, VCNextEntity, VCSetCurrentEntity

VCGetCurrentEntityLineWidth

Version	1.2
Description	Retrieves the line width of the current entity.
Declaration	
C/C++:	extern "C" short WINAPI VCGetCurrentEntityLineWidth(short* iError);
Visual Basic:	Declare Function VCGetCurrentEntityLineWidth Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Delphi:	function VCGetCurrentEntityLineWidth(var iError: Integer):Integer; far;
Parameters	*Returns* - the line width of the entity.
Notes	The entity whose information is desired must first be set. The current entity is set with VCSetCurrentEntity, VCFirstEntity, VCNextEntity, or VCLastEntity. This allows subsequent use of the VCGetCurrentEntity functions.
See Also	VCFirstEntity, VCGetCurrentEntityColor, VCGetCurrentEntityHandle, VCGetCurrentEntityKind, VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetCurrentEntityLineWidth, VCGetCurrentEntityLineWidthValue, VCLastEntity, VCNextEntity, VCSetCurrentEntity

VCGetCurrentEntityLineWidthValue

Version	2.0
Description	Specifies the real world line width for the current entity.
Declaration	
C/C++	extern "C" void WINAPI VCGetCurrentEntityLineWidthValue(short* iError, float* dV);
Visual Basic	Declare Sub VCGetCurrentEntityLineWidthValue Lib "VCMAIN32.DLL" (iError As Integer, dV As Double)
Delphi	procedure VCGetCurrentEntityLineWidthValue(var iError: Integer; var dV:
Parameters	dV - the real world line width
Notes	The entity whose information is desired must first be set. The current entity is set with VCSetCurrentEntity, VCFirstEntity, VCNextEntity, or VCLastEntity. This allows subsequent use of the VCGetCurrentEntity functions. VCGetCurrentEntityLineWidthValue retrieves the real world value
See Also	VCFirstEntity, VCGetCurrentEntityColor, VCGetCurrentEntityHandle, VCGetCurrentEntityKind, VCGetCurrentEntityLayer, VCGetCurrentEntityLayerName, VCGetCurrentEntityLineWidth, VCGetCurrentEntityLineWidthValue, VCLastEntity, VCNextEntity, VCSetCurrentEntity

VCGetCurrentEntityNormal3D

Version	1.2
Description	Returns the normal vector to the current 3D entity.
Declaration	
C/C++:	extern "C" void WINAPI VCGetCurrentEntityNormal3D(short* iError, Point3D* nV);
Visual Basic:	Declare Sub VCGetCurrentEntityNormal3D Lib "VCMAIN32.DLL" (iError As Integer, nV As Point3D)
Delphi:	procedure VCGetCurrentEntityNormal3D(var iError: Integer; var nV: Point3D); far;
Parameters	*nV* - the normal vector to the current entity.
Notes	Many calculations based on 3D geometry require the use of the normal vector. The normal vector is a perpendicular to the 3D face and is represented by x, y and z coordinates that define the direction of the vector.
See Also	VCChangeView3D, VCAddLine3D, VCAddPoint3D, VCSet3DDisplay, VCSet3DQShadeOptions

VCGetCurrentEntityPoint

Version 1.2

Description Retrieves the specified point from the current entity.

Declaration

C/C++: extern "C" Point2D WINAPI VCGetCurrentEntityPoint(short* iError, short iIndex);
extern "C" void WINAPI VCGetCurrentEntityPointBP(short* iError, short iIndex, Point2D* dpRet);

Visual Basic: Declare Sub VCGetCurrentEntityPointBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, dpRet As Point2d)

Delphi: proc function VCGetCurrentEntityPointCount(var iError: Integer):Integer; far; edure VCGetCurrentEntityPointBP(var iError: Integer; iIndex: Integer; var dpRet: Point2D); far;

Parameters *Returns* - the number of points defining the current entity.

Notes Any drawing entity is made up of construction points placed while constructing the entity or calculated from these placements. In order to retrieve any of these values from existing entities for use with any other constructions or external cataloging it may be necessary to use VCGetCurrentEntityPoint in order to retrieve this information. The current entity is set with VCSetCurrentEntity, VCFirstEntity, VCNextEntity, or VCLastEntity.

See Also VCFirstEntity, VCGetCurrentEntityPoint3D, VCLastEntity, VCSetCurrentEntity

VCGetCurrentEntityPoint3D

Version 1.2

Description Returns the specified point on a 3D entity.

Declaration

C/C++: extern "C" void WINAPI VCGetCurrentEntityPoint3D(short* iError, short iIndex, Point3D* dpRet);

Visual Basic: Declare Sub VCGetCurrentEntityPoint3D Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, dpRet As Point3D)

Delphi: procedure VCGetCurrentEntityPoint3D(var iError: Integer; iIndex: Integer; var dpRet: Point3D); far;

Parameters *iIndex* - the index for the point to retrieve.
DpRet - the returned 3D point.

Notes Any drawing entity is made up of construction points placed while constructing the entity or calculated from these placements. In order to retrieve any of these values from existing entities for use with any other constructions or external cataloging it may be necessary to use VCGetCurrentEntityPoint in order to retrieve this information. The current entity is set with VCSetCurrentEntity, VCFirstEntity, VCNextEntity, or VCLastEntity.

See Also VCAddLine3D, VCAddPoint3D, VCChangeView3D, VCFirstEntity, VCGetCurrentEntityPoint, VCGetCurrentEntityPointCount, VCLastEntity, VCNextEntity, VCSetCurrentEntity, VCSet3DDisplay, VCSet3DQShadeOptions

VCGetCurrentEntityPointCount

Version 1.2

Description Returns the number of points used in the definition of the current entity.

Declaration

C/C++: extern "C" short WINAPI VCGetCurrentEntityPointCount(short* iError);

Visual Basic: Declare Function VCGetCurrentEntityPointCount Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCGetCurrentEntityPointCount(var iError: Integer):Integer; far;

Parameters *Returns* - the number of points used to define the entity.

Notes Since continuous entities can be comprised of several construction points and not a preset amount of points, VCGetCurrentEntityPointCount provides the ability to determine the number of points prior to calling VCGetCurrentEntityPoint or VCGetCurrentEntityPoint3D which need a point index number in order to return the coordinates at each point. The current entity is set with VCSetCurrentEntity, VCFirstEntity, VCNextEntity, or VCLastEntity.

See Also VCFirstEntity, VCGetCurrentEntityPoint3D, VCGetCurrentEntityPoint,VCLastEntity, VCNextEntity, VCSetCurrentEntity

VCGetCurrentEntityUID
VCSetCurrentEntityUID

Version 2.0

Description Specifies the current entities UID(unique identifier).

Declaration

C/C++
```
extern "C" UID WINAPI VCGetCurrentEntityUID(short* iError);
extern "C" void WINAPI VCSetCurrentEntityUID(short* iError, UID uid);
```

Visual Basic
```
Declare Function VCGetCurrentEntityUID Lib "VCMAIN32.DLL" (iError As Integer) As Long
Declare Sub VCSetCurrentEntityUID Lib "VCMAIN32.DLL" (iError As Integer, ByVal uid As Long)
```

Delphi
```
function VCGetCurrentEntityUID(var iError: Integer):Longint; far;
procedure VCSetCurrentEntity3DFlag0(var iError: Integer; iFlag: Integer);
```

Parameters *uid* - the unique identifier for the current entiy

Notes Each entity in Corel Visual CADD maintains a unique entity identifier in order to track the entity. This is in addition to the dynamic entity handle which changes as entities are deleted and modified in the database. As entities are added to the drawing both an entity handle and a UID are assigned to the entity. The entity handle will change as items are deleted and modified on the database while the UID will remain constant. Whenever linking entities to external databases or arrays, the application should utilize the UID due to its unchanging value with each entity. The entity handle is used when parsing the database or setting specific entities within the drawing session. The UID can should be audited prior to any external storage in order to ensure uniqueness in the ID.

See Also VCGetCurrentEntityHandle, VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCGetCurrentEntityUserDataByte
VCSetCurrentEntityUserDataByte

Version 1.2

Description User data may be attached to any drawing entity or to the drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. This data can be assigned and retrieved from entities based on the data type and the specified index.

Declaration

C/C++: extern "C" BYTE WINAPI VCGetCurrentEntityUserDataByte(short* iError, short iIndex);
extern "C" void WINAPI VCSetCurrentEntityUserDataByte(short* iError, short iIndex, BYTE b);

Visual Basic: Declare Function VCGetCurrentEntityUserDataByte Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer) As Integer
Declare Sub VCSetCurrentEntityUserDataByte Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal b As Integer)

Delphi: function VCGetCurrentEntityUserDataByte(var iError: Integer; iIndex: Integer):Integer; far;
procedure VCSetCurrentEntityUserDataByte(var iError: Integer; iIndex: Integer; b: Integer); far;

Parameters *iIndex* - the index number within the current entity where the chunk should be stored.
b - Byte attached to the entity

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide
Examples

VCGetCurrentEntityUserDataChunk
VCSetCurrentEntityUserDataChunk

Version 1.2

Description User data may be attached to any drawing entity or to the drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. This data can be assigned and retrieved from entities based on the data type and the specified index. A Chunk is a piece of data un-associated with any data type. Chunks are most useful for assigning string data to an entities user data. The size of the chuck must be predetermined prior to calling this function in order for the function to know how much data to pull out of memory at the location specified by the pointer.

Declaration

C/C++: extern "C" void WINAPI VCGetCurrentEntityUserDataChunk(short* iError, short iIndex, char* p);
extern "C" void WINAPI VCSetCurrentEntityUserDataChunk(short* iError, short iIndex, void* p, short iSize);

Visual Basic: Declare Sub VCSetCurrentEntityUserDataChunk Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, p As Any, ByVal iSize As Integer)
Declare Sub VCSetCurrentEntityUserDataChunk Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, p As Any, ByVal iSize As Integer)

Delphi: procedure VCGetCurrentEntityUserDataChunk(var iError: Integer; iIndex: Integer; var p: String); far;
procedure VCSetCurrentEntityUserDataChunk(var iError: Integer; iIndex: Integer; var p: Pointer; iSize: Integer); far;

Parameters *Index* - the index number within the current entity where the chunk should be stored.
p - a pointer to a memory location where the data chunk is stored.
iSize - the size of the data chunk in bytes.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide
Examples

VCGetCurrentEntityUserDataCount

Version 1.2

Description Retrieves the count or number of indices of user data attached to the current entity.

Declaration

C/C++: extern "C" short WINAPI VCGetCurrentEntityUserDataCount(short* iError);

Visual Basic: Declare Function VCGetCurrentEntityUserDataCount Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCGetCurrentEntityUserDataCount(var iError: Integer):Integer; far;

Parameters *Returns* - a count of the attached User Data types

Notes The current entity is set with VCSetCurrentEntity, VCFirstEntity, or VCNextEntity.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCGetCurrentEntityUserDataDouble
VCSetCurrentEntityUserDataDouble

Version 1.2

Description User data may be attached to any drawing entity or to the drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. This data can be assigned and retrieved from entities based on the data type and the specified index.

Declaration

C/C++: extern "C" double WINAPI VCGetCurrentEntityUserDataDouble(short* iError, short iIndex);
extern "C" void WINAPI VCSetCurrentEntityUserDataDouble(short* iError, short iIndex, double dRet);

Visual Basic: Declare Sub VCGetCurrentEntUserDataDoubleBP Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, dRet As Double)
Declare Sub VCSetCurrentEntityUserDataDouble Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal d As Double)

Delphi: procedure VCGetCurrentEntUserDataDoubleBP(var iError: Integer; iIndex: Integer; var dRet: Double); far;
procedure VCSetCurrentEntityUserDataDouble(var iError: Integer; iIndex: Integer; dRet: Double); far;

Parameters *iIndex* - the index number within the current entity where the chunk should be stored.
d - double value attached to the entity

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide Examples

VCGetCurrentEntityUserDataFloat
VCSetCurrentEntityUserDataFloat

Version 1.2

Description User data may be attached to any drawing entity or to the drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. This data can be assigned and retrieved from entities based on the data type and the specified index.

Declaration

C/C++:
```
extern "C" float WINAPI VCGetCurrentEntityUserDataFloat(short* iError, short iIndex);
extern "C" void WINAPI VCSetCurrentEntityUserDataFloat(short* iError, short iIndex, float f);
```

Visual Basic:
```
Declare Function VCGetCurrentEntityUserDataFloat Lib "VCMAIN32.DLL" ( iError As Integer,
ByVal iIndex As Integer) As single
Declare Sub VCSetCurrentEntityUserDataFloat Lib "VCMAIN32.DLL" ( iError As Integer, ByVal
iIndex As Integer, ByVal f As Single)
```

Delphi:

Parameters *iIndex* - the index number within the current entity where the chunk should be stored.
d - value attached to the entity

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide
Examples

VCGetCurrentEntityUserDataKind

Version 1.2

Description Determines the record type of the specified index attached to the current entity.

Declaration

C/C++: extern "C" short WINAPI VCGetCurrentEntityUserDataKind(short* iError, short iIndex);

Visual Basic: Declare Function VCGetCurrentEntityUserDataKind Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer) As Integer

Delphi: function VCGetCurrentEntityUserDataKind(var iError: Integer; iIndex: Integer):Integer; far;

Parameters iIndex - the user data index to retrieve.
Returns - the data type for the specified user data.
1 - Byte
2 - Short
3 - Long
4 - Double
5 - Float
6 - Chunk

Notes User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of the C variable types double, float, long, or short. In addition to these types, a user defined type of "chunk" may also be stored. A chunk may be any size and is simply a pointer to a memory location. The size of the chunk is also passed so Corel Visual CADD can retrieve the appropriate amount of data from the specified memory location. Whenever using user data, an application must set a user data name in order to protect private data and to ensure that different applications do not interfere with the others data. VCSetUserDataName is provided for this purpose, while VCGetUserDataName checks the currently set user data name. The name must only be set one time before adding any user data. The VCAddCurrentEntityUserData* calls always append the new variable as the last user data variable. The VCSetCurrentEntityUserData* calls add the user data variable at the index specified in the call, provided that there are indeed that many indices already attached, and overwrite any existing user data at that index. As previously mentioned, user data may be attached to the drawing header. This is achieved by using VCSetHeaderUserData and then attaching the appropriate user data. Once VCNextEntity or any other current entity selections are used, the user data calls will again be used on the current entity.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize,VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide
Examples

VCGetCurrentEntityUserDataLong
VCSetCurrentEntityUserDataLong

Version 1.2

Description User data may be attached to any drawing entity or to the drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. This data can be assigned and retrieved from entities based on the data type and the specified index.

Declaration

C/C++:
```
extern "C" long WINAPI VCGetCurrentEntityUserDataLong(short* iError, short iIndex);
extern "C" void WINAPI VCSetCurrentEntityUserDataLong(short* iError, short iIndex, long l);
```

Visual Basic:
```
Declare Function VCGetCurrentEntityUserDataLong Lib "VCMAIN32.DLL" ( iError As Integer, ByVal iIndex As Integer) As long
Declare Sub VCSetCurrentEntityUserDataLong Lib "VCMAIN32.DLL" ( iError As Integer, ByVal iIndex As Integer, ByVal l As Long)
```

Delphi:
```
function VCGetCurrentEntityUserDataLong(var iError: Integer; iIndex: Integer):Long; far;
procedure VCSetCurrentEntityUserDataLong(var iError: Integer; iIndex: Integer; l: Longint); far;
```

Parameters *iIndex* - the index number within the current entity where the chunk should be stored.
d - value attached to the entity.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide
Examples

VCGetCurrentEntityUserDataShort
VCSetCurrentEntityUserDataShort

Version 1.2

Description User data may be attached to any drawing entity or to the drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. This data can be assigned and retrieved from entities based on the data type and the specified index.

Declaration

C/C++: extern "C" short WINAPI VCGetCurrentEntityUserDataShort(short* iError, short iIndex);
extern "C" void WINAPI VCSetCurrentEntityUserDataShort(short* iError, short iIndex, short s);

Visual Basic: Declare Function VCGetCurrentEntityUserDataShort Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer) As Integer
Declare Sub VCSetCurrentEntityUserDataShort Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal s As Integer)

Delphi: function VCGetCurrentEntityUserDataShort(var iError: Integer; iIndex:Integer):Integer; far;
procedure VCSetCurrentEntityUserDataShort(var iError: Integer; iIndex:Integer; s: Integer); far;

Parameters *iIndex* - the index number within the current entity where the chunk should be stored.
s - value attached to the entity

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID, VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntiytUserDataString, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide
Examples

VCGetCurrentEntityUserDataString
VCSetCurrentEntityUserDataString

Version 2.0

Description User data may be attached to any drawing entity or to the drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. This data can be assigned and retrieved from entities based on the data type and the specified index.

Declaration

C/C++ extern "C" void WINAPI VCGetCurrentEntityUserDataString(short* iError, short iIndex, char* str);

Visual Basic Declare Sub VCGetCurrentEntityUserDataString Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal str As String)

Delphi procedure VCGetCurrentEntityUserDataString(var iError: Integer; iIndex: Integer; str: PChar); far;

Parameters *iIndex* - the index number within the current entity where the stiring should be stored.
Str - the attached string

Notes

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort, VCGetUserDataName, VCGetCurrentEntityUID,VCGetCurrentEntityUserDataByte, VCGetCurrentEntityUserDataChunk, VCGetCurrentEntityUserDataCount, VCGetCurrentEntityUserDataDouble, VCGetCurrentEntityUserDataKind, VCGetCurrentEntityUserDataLong, VCGetCurrentEntityUserDataFloat, VCGetCurrentEntityUserDataShort, VCGetCurEntUserDataChunkSize, VCSetHeaderUserData

{button ,AL(`Attaching User Data;Database Operations;User Data Retrieval;User Data Tasks',0,`',`')} Task Guide
Examples

VCGetCurrentOleClassId

Version	2.0
Description	Retrieves the current OLE class ID.
Declaration	
C/C++	extern "C" long WINAPI VCGetCurentOleClassId(short* iError);
Visual Basic	Declare Function VCGetCurentOleClassId Lib "VCMAIN32.DLL" (iError As Integer) As Long
Delphi	function VCGetCurentOleClassId(var iError: Integer):Longint; far;
Parameters	*returns* - the current OLE class id.
Notes	An application can be created as an EXE, a Windows DLL or an OLE DLL. Each has advantages in functionality and interaction with the CAD engine. In addition, each is accessed through the Corel Visual CADD interface in different methods. An OLE DLL is a specialized link library containing methods and classes for controlling various operations. These DLL are specifically related to Visual Basic programmers. The OLE class allows a developer to create a class member function that can be directly run from the Corel Visual CADD interface allowing an application to take advantage of the performance increase associated with a DLL. In order to access this functionality the DLL and the class must be registered. VCCreateOLEClass registers the DLL and class. VCInvokeMethod will invoke the DLL method and VCDeleteOleClass will delete the registered DLL and class.
See Also	VCDeleteOleClass, VCOleClassMethodInvoke,VCCreateOleClass

VCGetCurrentPrinter

Version	2.0
Description	Specifies the current printer for the print routine.
Declaration	
C/C++	extern "C" short WINAPI VCGetCurrentPrinter(short* iError, char* szPrinter);
Visual Basic	Declare Function VCGetCurrentPrinter Lib "VCDLG32.DLL" (iError As Integer, ByVal szPrinter As String) As Integer
Delphi	function VCGetCurrentPrinter(var iError: Integer; szPrinter: PChar):Integer; far;
Parameters	*returns* - the lenght of the returned string. *szPrinter* - the printer name.
See Also	VCGetPrinterName, VCGetPrinterNameCount, VCGetPrintSettings

VCGetCurrentPoint

Version	1.2
Description	Returns the current point.
Declaration	
C/C++:	extern "C" void WINAPI VCGetCurrentPoint(short* iError, Point2D* dpP);
Visual Basic:	Declare Sub VCGetCurrentPoint Lib "VCMAIN32.DLL" (iError As Integer, dpP As Point2D)
Delphi:	procedure VCGetCurrentPoint(var iError: Integer; var dpP: Point2D); far;
Parameters	*dpP* - the current location of the drag rubberband.
Notes	When a user is constructing or modifying entities, Corel Visual CADD displays a dynamic feedback to preview what the change would look like if the point was placed where the cursor is currently sitting. This is called rubberbanding. In order to retrieve the current rubberband point while a user is constructing or editing any entities, VCGetCurrentPoint can be called and will return the current location of the rubberbanding cursor.
See Also	VCGetCurrentEntityPointCount, VCGetCurrentEntityPoint

VCGetCurrentUID

Version	2.0
Description	Retrieves the current UID for the next entity added to the database.
Declaration	
C/C++	extern "C" UID WINAPI VCGetCurrentUID(short* iError);
Visual Basic	Declare Function VCGetCurrentUID Lib "VCMAIN32.DLL" (iError As Integer) As Long
Delphi	function VCGetCurrentUID(var iError: Integer):Longint; far; procedure VCSetCurrentEntityUserDataString(var iError: Integer; iIndex:
Parameters	Returns the next UID.
Notes	Each entity in Corel Visual CADD maintains a unique entity identifier in order to track the entity. This is in addition to the dynamic entity handle which changes as entities are deleted and modified in the database. As entities are added to the drawing both an entity handle and a UID are assigned to the entity. The entity handle will change as items are deleted and modified on the database while the UID will remain constant. Whenever linking entities to external databases or arrays, the application should utilize the UID due to its unchanging value with each entity. The entity handle is used when parsing the database or setting specific entities within the drawing session. The UID can should be audited prior to any external storage in order to ensure uniqueness in the ID. VCGetCurrentUID differs from VCGetCurrentEntityUID in that VCGetCurrentUID give you the next available unique entity identifier available.
See Also	VCGetCurrentEntityHandle, VCGetCurrentEntityUID, VCGetUserDataName

VCGetCurrentUndoLevel

Version	2.0
Description	Retrieves the current undo level for placing entities.
Declaration	
C/C++	extern "C" short WINAPI VCGetCurrentUndoLevel(short* iError);
Visual Basic	Declare Function VCGetCurrentUndoLevel Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Delphi	function VCGetCurrentUndoLevel(var iError: Integer):Integer; far;
Parameters	Returns the current entity undo level.
Notes	Corel Visual CADD maintains a complete undo level for all entities added to the database. This information is then used when undo or redo operations are activated. Corel Visual CADD automatically increments the undo level as each command is completed the entity then takes on the last level after the operation is complete. When adding entities directly through the API, an application can monitor the undo level allowing for custom undo operations. Entities added using the API take on the active undo level but do not increment the level. This allow multiple add operation to be undone with a single operation. An application can bypass this functionality by utilizing VCBeginOperation and VCEndOperation to set application specific undo levels. The application can also monitor the current undo level in order to track the sequence an entity is added.
See Also	VCBeginOperation, VCAbortOperation

{button ,AL(`Creating a User Tool;Duplicating an Entity;Duplicating an Entity with Transformation',0,`',`')} Task Guide Examples

VCGetCurrentView
VCSetCurrentView

Version 2.0

Description Returns or sets the Corel Visual CADD view handle of the current drawing world

Declaration

C/C++
```
extern "C" void WINAPI VCGetCurrentView(short* iError, short* iView);
extern "C" void WINAPI VCSetCurrentView(short* iError, short iView);
```

Visual Basic
```
Declare Sub VCGetCurrentView Lib "VCMAIN32.DLL" (iError As Integer, iView As Integer)
Declare Sub VCSetCurrentView Lib "VCMAIN32.DLL" (iError As Integer, ByVal iView As Integer)
```

Delphi
```
procedure VCGetCurrentView(var iError: Integer; var iView: Integer); far;
procedure VCSetCurrentView(var iError: Integer; iView: Integer); far;
```

Parameters *iView* - the Corel Visual CADD world handle returned when VCNewWorld is used.

Notes Corel Visual CADD allows multiple views of the same drawing to appear. VCGetCurrentView helps you keep track of the number of instances of multiple views that are running. Returns a value between 0 and 63.

See Also VCIsCurrentWorldValid, VCNewWorld, VCDestroyWorld

VCGetCurrEntRFAbsName

Version 2.0

Description Returns the absolute file path and name for the current reference frame entity.

Declaration

C/C++ extern "C" short WINAPI VCGetCurrEntRFAbsName(short* iError, char* RetPath);

Visual Basic Declare Function VCGetCurrEntRFAbsName Lib "VCMAIN32.DLL" (iError As Integer, ByVal RetPath As String) As Integer

Delphi function VCGetCurrEntRFAbsName(var iError: Integer; RetPath: PChar):Integer;far;

Parameters *returns* - the length of the returned string.
RetPath - the absolute file name and path for the referenced entity.

Notes Reference frames allow external files to be linked into an existing drawing. When linked, the files are represented by a relative path between the current file location and the absolute path to the file. For example, if the current active drawing for an open VCD files is "C:\VCADD\SAMPLES\THISFILE.VCD" and a file is referenced into this drawing located at an absolute location of "C:\VCADD\LINKEDFILE.VCD" this routine will return the difference of the paths. In this case it will return " ..\" or indication that the linked file is located back one subdirectory. The routine can be used to retrieve the relative path for any given directory. Simply pass in a current directory(where the active drawing is) and the absolute path the linked file(file that is being referenced) and the routine will return the relative path for the directories.

See Also VCGetCurrEntRFAbsShortName, VCRelativePath

VCGetCurrEntRFAbsShortName

Version 2.0

Description Returns the absolute file path and name for the current reference frame entity.

Declaration

C/C++ extern "C" short WINAPI VCGetCurrEntRFAbsName(short* iError, char* RetPath);

Visual Basic Declare Function VCGetCurrEntRFAbsName Lib "VCMAIN32.DLL" (iError As Integer, ByVal RetPath As String) As Integer

Delphi function VCGetCurrEntRFAbsName(var iError: Integer; RetPath: PChar):Integer;far;

Parameters *returns* - the length of the returned string.
RetPath - the absolute file name and path for the referenced entity.

Notes Reference frames allow external files to be linked into an existing drawing. When linked, the files are represented by a relative path between the current file location and the absolute path to the file. For example, if the current active drawing for an open VCD files is "C:\VCADD\SAMPLES\THISFILE.VCD" and a file is referenced into this drawing located at an absolute location of "C:\VCADD\LINKEDFILE.VCD" this routine will return the difference of the paths. In this case it will return " ..\" or indication that the linked file is located back one subdirectory. The routine can be used to retrieve the relative path for any given directory. Simply pass in a current directory(where the active drawing is) and the absolute path the linked file(file that is being referenced) and the routine will return the relative path for the directories.

See Also VCGetCurrEntRFAbsName, VCRelativePath

VCGetCurrWorld
VCSetCurrWorld

Version 1.2

Description Returns or sets the Corel Visual CADD world handle of the current drawing world.

Declaration

C/C++:
```
extern "C" WORLDHANDLE WINAPI VCGetCurrWorld(void);
extern "C" void WINAPI VCSetCurrWorld(WORLDHANDLE hW);
```

Visual Basic:
```
Declare Function VCGetCurrWorld Lib "VCMAIN32.DLL" () As Long
Declare Sub VCSetCurrWorld Lib "VCMAIN32.DLL" (ByVal hW As Long)
```

Delphi:

Parameters *hW* - the Corel Visual CADD world handle returned when VCNewWorld is used.

Notes When using multiple drawings, particularly with MDI windows, it is necessary to know which world is current. Before making any changes to a drawing in which the current world is not explicitly known, VCGetCurrWorld should be used to verify that intended world is active and if not VCSetCurrWorld should be used to set the current world accordingly. The values for VCGetCurrWorld will range from 0 to 63.

See Also VCGetCurrentView, VCIsCurrentWorldValid, VCNewWorld, VCDestroyWorld

{button ,AL(`Creating a User Tool;Valid World Checking',0,`',`')} Task Guide Examples

VCGetCurrZoom

Version	1.2
Description	Returns the lower left and upper right coordinates of the current drawing view.
Declaration	
C/C++:	extern "C" void WINAPI VCGetCurrZoom(short* iError, Point2D* dpMin, Point2D* dpMax);
Visual Basic:	Declare Sub VCGetCurrZoom Lib "VCMAIN32.DLL" (iError As Integer, dpMin As Point2D, dpMax As Point2D)
Delphi:	procedure VCGetCurrZoom(var iError: Integer; var dpMin: Point2D; var dpMax:
Parameters	*dpMin* - the lower left corner of the view.
	dpMax - the upper right corner of the view.
Notes	If an application needs to determine the bounds of the current view in relation to coordinates in the drawing VCGetCurrrZoom will return two Point2D's containing the lower left and upper right corners of the drawing view.
See Also	VCZoomWindow

VCGetCursorFree
VCSetCursorFree

Version 1.2

Description Cursor free allows the cursor be free to move in any direction in ortho mode and allows the cursor to move unconstrained on the screen.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetCursorFree(short* iError);
extern "C" void WINAPI VCSetCursorFree(short* iError, vbool tf);
```

Visual Basic:
```
Declare Function VCGetCursorFree Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetCursorFree Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetCursorFree(var iError: Integer):Integer; far;
procedure VCSetCursorFree(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetOrthoMode

VCGetCursorColor
VCSetCursorColor

Version 1.2

Description The sets or gets the cursor color.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetCursorColor(short* iError);
extern "C" void WINAPI VCSetCursorColor(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetCursorColor Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetCursorColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetCursorColor(var iError: Integer):Integer; far;
procedure VCSetCursorColor(var iError: Integer; i: Integer); far;
```

Parameters *i* - the cursor color index from 0 to 15.

See Also VCGetBackgroundColor, VCGetCursorSize

VCGetCursorSize
VCSetCursorSize

Version 1.2

Description The cursor size in pixels.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetCursorSize(short* iError);
extern "C" void WINAPI VCSetCursorSize(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetCursorSize Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetCursorSize Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetCursorSize(var iError: Integer): Integer; far;
procedure VCSetCursorSize(var iError: Integer; i: Integer); far;
```

Parameters *i* - the cursor size in pixels

See Also VCGetVidTolerance

VCGetDatumMode
VCSetDatumMode

Version 2.0

Description Species the datum dimension mode. Datum dimensions are leader with X and Y coordinates attached for a location.

Declaration

C/C++
```
extern "C" short WINAPI VCGetDatumMode(short* iError);
extern "C" void WINAPI VCSetDatumMode(short* iError, short iMode);
```

Visual Basic
```
Declare Function VCGetDatumMode Lib "VCTOOL32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDatumMode Lib "VCTOOL32.DLL" (iError As Integer, ByVal iMode As Integer)
```

Delphi
```
function VCGetDatumMode(var iError: Integer):Integer; far;
procedure VCSetDatumMode(var iError: Integer; iMode: Integer); far;
```

Parameters iMode - the datum dimension mode.
0 - DATUMNONE
1 - DATUMXY
2 - DATUMX
3 - DATUMY

See Also VCGetDatumType, VCGetDatumBasePt

VCGetDatumType
VCSetDatumType

Version 2.0

Description Species the datum dimension mode. Datum dimensions are leader with X and Y coordinates attached for a location.

Declaration

C/C++
```
extern "C" short WINAPI VCGetDatumType(short* iError);
extern "C" void WINAPI VCSetDatumType(short* iError, short iDatumType);
```

Visual Basic
```
Declare Function VCGetDatumType Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDatumType Lib "VCMAIN32.DLL" (iError As Integer, ByVal iDatumType As Integer)
```

Delphi
```
function VCGetDatumType(var iError: Integer):Integer; far;
procedure VCSetDatumType(var iError: Integer; iDatumType: Integer); far;
```

Parameters iDatumType - the datum dimension type
0 - OFF
1 - XY
2 - X Only
3 - Y Only

See Also VCGetDatumBasePt, VCGetDatumMode

VCGetDefaultPrinter
VCSetDefaultPrinter

Version 2.0

Description Specifies the default printer.

Declaration

C/C++

```
extern "C" short WINAPI VCGetDefaultPrinter(short* iError, char* szDefaultPrinter);
extern "C" void WINAPI VCSetDefaultPrinter(short* iError, char* szDefaultPrinter);
```

Visual Basic

```
Declare Function VCGetDefaultPrinter Lib "VCDLG32.DLL" (iError As Integer, ByVal
szDefaultPrinter As String) As Integer
Declare Sub VCSetDefaultPrinter Lib "VCDLG32.DLL" (iError As Integer, ByVal szDefaultPrinter As
String)
```

Delphi

```
function VCGetDefaultPrinter(var iError: Integer; szDefaultPrinter: PChar):Integer; far;
procedure VCSetDefaultPrinter(var iError: Integer; szDefaultPrinter: PChar); far;
```

Parameters returns - the length of the returned string.
szDefaultPriner - the name of the default printer.

See Also VCGetCurrentPrinter, VCGetPrinterName, VCGetPrinterNameCount, VCGetPrintSettings

VCGetDefaultTool
VCSetDefaultTool

Version 1.2

Description The default drawing tool is pre-configured by the user and can be set to Single Line, Continuous Line or Selection.

Declaration

C/C++:
```
extern "C" WORD WINAPI VCGetDefaultTool(short* iError);
extern "C" void WINAPI VCSetDefaultTool(short* iError, WORD w);
```

Visual Basic:
```
Declare Function VCGetDefaultTool Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDefaultTool Lib "VCMAIN32.DLL" (iError As Integer, ByVal w As Integer)
```

Delphi:
```
function VCGetDefaultTool(var iError: Integer):Integer; far;
procedure VCSetDefaultTool(var iError: Integer; w: Integer); far;
```

Parameters *w* - the tool index
2102 - Single Line
2103 - Continuous Line
2449 - Select

See Also VCGetCursorSize, VCLineSingle, VCSelect

VCGetDimAngleFormat
VCSetDimAngleFormat

Version 2.0

Description Format for displaying angles as decimal degrees or degrees:minutes:seconds. If decimal degrees format is used, the number is decimal places displayed is determined by VCGetDimDecimalValue.

Declaration

C/C++

```
extern "C" short WINAPI VCGetDimAngleFormat(short* iError);
extern "C" void WINAPI VCSetDimAngleFormat(short* iError, short iF_);
```

Visual Basic

```
Declare Function VCGetDimAngleFormat Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Declare Sub VCSetDimAngleFormat Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi

```
function VCGetDimAngleFormat(var iError: Integer):Integer; far;
procedure VCSetDimAngleFormat(var iError: Integer; iF_: Integer); far;
```

Parameters iF - determines angular format to be used.
9 - Angle and Degrees.
10 - Degrees Minutes Seconds.

Notes

See Also VCGetDimDecimalValue, VCGetDisplayAngleFormat

VCGetDimArrowAngle
VCSetDimArrowAngle

Version 1.2

Description The dimension angle setting is used by all dimension arrow types except circular. As with all angular settings in Corel Visual CADD the value should be expressed in radians.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimArrowAngle(short* iError);
extern "C" void WINAPI VCGetDimArrowAngleBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimArrowAngle(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetDimArrowAngleBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetDimArrowAngle Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDimArrowAngleBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDimArrowAngle(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - double value representing the angle setting in radians

See Also VCGetDimArrowFlipDists, VCGetDimArrowLength, VCGetDimArrowMode, VCGetDimArrowType

VCGetDimArrowFlipDists
VCSetDimArrowFlipDists

Version 1.2

Description The length of the dimension line segment when arrowheads have been reversed. Flip reverses the direction of the dimension arrowheads so they point inward instead of outward. The dimension line is split and flipped to the outside of the dimensioned area.

Declaration

C/C++:
```
extern "C" void WINAPI VCGetDimArrowFlipDists(short* iError, double* d0, double* d1);
extern "C" void WINAPI VCSetDimArrowFlipDists(short* iError, double d0, double d1);
```

Visual Basic:
```
Declare Sub VCGetDimArrowFlipDists Lib "VCMAIN32.DLL" (iError As Integer, d0 As Double, d1 As Double)
Declare Sub VCSetDimArrowFlipDists Lib "VCMAIN32.DLL" (iError As Integer, ByVal d0 As Double, ByVal d1 As Double)
```

Delphi:
```
procedure VCGetDimArrowFlipDists(var iError: Integer; var d0: Double; var d1: Double); far;
procedure VCSetDimArrowFlipDists(var iError: Integer; d0: Double; d1: Double); far;
```

Parameters
d0 - left side flip distance
d1 - right side flip distance

See Also VCGetDimArrowAngle, VCGetDimArrowLength, VCGetDimArrowMode, VCGetDimArrowType

VCGetDimArrowLength
VCSetDimArrowLength

Version 1.2

Description Several settings are available for dimension arrows. These need to be set prior to placing the dimension into the drawing. The arrow length is analogous to the arrow size or scale.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimArrowLength(short* iError);
extern "C" void WINAPI VCGetDimArrowLengthBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimArrowLength(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetDimArrowLengthBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetDimArrowLength Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDimArrowLengthBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDimArrowLength(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the dimension arrow length

See Also VCGetDimArrowFlipDists, VCGetDimArrowAngle, VCGetDimArrowMode, VCGetDimArrowType

VCGetDimArrowMode
VCSetDimArrowMode

Version 1.2

Description Several settings are available for dimension arrows. These need to be set prior to placing the dimension into the drawing The arrow mode determines if the arrows are flipped to the outside or the inside of the extension lines.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimArrowMode(short* iError);
extern "C" void WINAPI VCSetDimArrowMode(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetDimArrowMode Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimArrowMode Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimArrowMode(var iError: Integer):Integer; far;
procedure VCSetDimArrowMode(var iError: Integer; b: Integer); far;
```

Parameters *b* - the state of the arrow flip.
0 - do not flip the dimension arrows to the outside of the dimension.
1 - flip the dimension arrows to the outside of the dimension.

See Also VCGetDimArrowFlipDists, VCGetDimArrowLength, VCGetDimArrowAngle, VCGetDimArrowType

VCGetDimArrowType
VCSetDimArrowType

Version 1.2

Description Several settings are available for dimension arrows. These need to be set prior to placing the dimension into the drawing. Corel Visual CADD allows several options for the dimension arrow type setting.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimArrowType(short* iError);
extern "C" void WINAPI VCSetDimArrowType(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetDimArrowType Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimArrowType Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimArrowType(var iError: Integer):Integer; far;
procedure VCSetDimArrowType(var iError: Integer; b: Integer); far;
```

Parameters *b* - the value of the arrow type.
0 - DIMARROWREGNOFILL
1 - DIMARROWREGFILLED
2 - DIMARROWREGOPEN
3 - DIMARROWNOTCHED
4 - DIMARROWSLASH
5 - DIMARROWCIRCLENOFILL
6 - DIMARROWCIRCLEFILL

See Also VCGetDimArrowFlipDists, VCGetDimArrowLength, VCGetDimArrowMode

VCGetDimDecimalValue
VCSetDimDecimalValue

Version 2.0

Description The number of digits displayed to the right of the decimal point.

Declaration

C/C++
```
extern "C" short WINAPI VCGetDimDecimalValue(short* iError);
extern "C" void WINAPI VCSetDimDecimalValue(short* iError, short iF_);
```

Visual Basic
```
Declare Function VCGetDimDecimalValue Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Declare Sub VCSetDimDecimalValue Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi
```
function VCGetDimDecimalValue(var iError: Integer):Integer; far;
procedure VCSetDimDecimalValue(var iError: Integer; iF_: Integer); far;
```

Parameters iF - the number of digits to use in the display. The valid range is 0 -8.

Notes Corel Visual CADD calculates and stores real numbers to a precision of 16 significant digits. Setting decimal places or fractions affects only how the numbers are displayed, not how they are calculated or stored.

See Also VCGetDisplayAngleFormat, VCGetDisplayDecimalValue, VCGetDisplayDistFormat, VCGetDisplayFractionalValue, VCGetDisplayShowLeadingZeros, VCGetDisplayShowUnits

VCGetDimDisplayItemCount

Version 1.2

Description Used to get the Dimension display item count.

Declaration

C/C++: extern "C" short WINAPI VCGetDimDisplayItemCount(short* iError);

Visual Basic: Declare Function VCGetDimDisplayItemCount Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCGetDimDisplayItemCount(var iError: Integer):Integer; far;

Parameters returns a count for the number of display items

Notes The elements that make up a dimension include the dimension line, left and right extension lines, left and right arrow and the dimension text. The API gives complete control over the visual properties of each of the dimension elements independent of each other. Changing the properties of dimension elements will not effect previously drawn dimensions. VCGetDimDisplayItemName returns the element name at the specified index. All dimension and leader extension settings must be set prior to creation of the dimension or leader. See the Corel Visual CADD reference manual for specific settings and what they do. Each setting has a "set" function to set/get API function call. For example, you can use VCGetDimItemColor to set the different dimension properties. Another use for the get functions is when querying specific settings of a dimension or leader. If you want to match all the dimension elements, VCMatchCurrentEntity is used to set all settings identical to the current entity. Each setting can then be extracted from the system settings. All dimension and leader settings must be set prior to creation of the dimension or leader. See the Corel Visual CADD reference manual for specific settings and what they do. Each setting has a "set" function to set the value and a "get" function to retrieve.

See Also VCGetDimDisplayItemName, VCGetDimItemColor, VCGetDimItemLineWidth, VCGetDimItemLineType, VCGetDimDisplayItemName, VCMatchCurrentEntity

VCGetDimDisplayItemName

Version 1.2

Description Returns the dimension display item name.

Declaration

C/C++: extern "C" short WINAPI VCGetDimDisplayItemName(short* iError, short i, char* pS);

Visual Basic: Declare Function VCGetDimDisplayItemName Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer, ByVal pS As String) As Integer

Delphi: function VCGetDimDisplayItemName(var iError: Integer; i: Integer; pS PChar):Integer; far;

Parameters *w* - the item index
pS - the dimension item name

Notes The elements that make up a dimension include the dimension line, left and right extension lines, left and right arrow and the dimension text. The API gives complete control over the visual properties of each of the dimension elements independent of each other. Changing the properties of dimension elements will not effect previously drawn dimensions. VCGetDimDisplayItemName returns the element name at the specified index.

See Also VCGetDimItemLineType, VCGetDimItemLineWidth, VCGetDimItemShow, VCGetDimItemColor

VCGetDimDistFormat
VCSetDimDistFormat

Version 2.0

Description Option to set or get the display dimension units.

Declaration

C/C++
```
extern "C" short WINAPI VCGetDimDistFormat(short* iError);
extern "C" void WINAPI VCSetDimDistFormat(short* iError, short iF_);
```

Visual Basic
```
Declare Function VCGetDimDistFormat Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimDistFormat Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi
```
function VCGetDimDistFormat(var iError: Integer):Integer; far;
procedure VCSetDimDistFormat(var iError: Integer; iF_: Integer); far;
```

Parameters if - the display format
0 - Decimal Inches
1 - Decimal Feet
2 - Decimal Feet & Inches
3 - Fractional Inches
4 - Fractional Feet
5 - Fractional Feet & Inches
6 - Millimeter
7 - Centimeter
8 - Meter

Notes In Corel Visual CADD 2.0, you can modify what units you want the dimensions to be displayed in by calling VCGetDimDistFormat. If you want top have multiple units displayed, such as decimal inches a meters, VCGetSecondaryDistFormat will allow you to specify what you want the secondary units to be.

See Also VCGetDimDecimalValue, VCGetDimFractionalValue, VCGetDisplayShowUnits, VCGetDisplayShowLeadingZeros, VCGetDisplayFractionalValue, VCGetDisplayDecimalValue, VCGetSecondaryDistFormat, VCGetUnitConversionFactor

VCGetDimFractionalValue
VCSetDimFractionalValue

Version 2.0

Description Returns an integer representing the denominator of the fractional display value. All decimal values will be rounded to the nearest fractional values represented by this denominator when displayed. This does not affect stored values, only the display of these values.

Declaration

C/C++
```
extern "C" short WINAPI VCGetDimFractionalValue(short* iError);
extern "C" void WINAPI VCSetDimFractionalValue(short* iError, short iF_);
```

Visual Basic
```
Declare Function VCGetDimFractionalValue Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimFractionalValue Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi
```
function VCGetDimFractionalValue(var iError: Integer):Integer; far;
procedure VCSetDimFractionalValue(var iError: Integer; iF_: Integer); far;
```

Parameters iF - determines the denominator of the fractional value to be used.
2 - 1/2.
4 - 1/4.
8 - 1/8.
16 - 1/16.
32 - 1/32.
64 - 1/64.

Notes The fractional values need to be used in conjunction with VCGetDimDistFormat: the units in VCGetDimDistFormat must be in Fractional Feet, Fractional Inches, or Fractional Feet & Inches.

See Also VCGetDimDistFormat, VCGetDisplayShowUnits, VCGetDisplayShowLeadingZeros, VCGetDisplayFractionalValue, VCGetDisplayDecimalValue, VCGetUnitConversionFactor

VCGetDimShowDash
VCSetDimShowDash

Version 2.0

Description Specifies if a dash is placed between fractional feet and inches values.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetDimShowDash(short* iError);
extern "C" void WINAPI VCSetDimShowDash(short* iError, vbool tf);
```

Visual Basic
```
Declare Function VCGetDimShowDash Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimShowDash Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetDimShowDash(var iError: Integer):Boolean; far;
procedure VCSetDimShowDash(var iError: Integer; tf: Boolean); far;
```

Parameters tf - show the dash in the dimension
0 - do not show the dash
1 - show the dash

See Also VCGetDimDistFormat, VCGetDisplayShowUnits, VCGetDisplayShowLeadingZeros, VCGetDisplayFractionalValue, VCGetDisplayDecimalValue, VCGetUnitConversionFactor, VCGetSecondaryDistFormat

VCGetDimShowFractions
VCSetDimShowFractions

Version 2.0

Description Dimension fractions can be displayed as a single character (¼) or multiple characters separated by a slash (1/4).

Declaration

C/C++
```
extern "C" short WINAPI VCGetDimShowFractions(short* iError);
extern "C" void WINAPI VCSetDimShowFractions(short* iError, short iF_);
```

Visual Basic
```
Declare Function VCGetDimShowFractions Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimShowFractions Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi
```
function VCGetDimShowFractions(var iError: Integer):Integer; far;
procedure VCSetDimShowFractions(var iError: Integer; iF_: Integer); far;
```

Parameters
tf - show the dimension fraction
0 - do not show the fraction
1 - show the fraction

Notes This option is available only for vector fonts which is determined with VCIsFontNameVT. Corel Visual CADD has more control over vector fonts than it does over True type fonts, so therefore it is best to use vector fonts when possible.

See Also VCGetDimFont, VCGetTextFontName, VCIsFontNameVText, VCIsTextFontVText

VCGetDimShowLeadingZeros
VCSetDimShowLeadingZeros

Version 2.0

Description When displaying decimal values between 1 and -1, it may be preferred to not display the leading zero - the single zero before the decimal point.

Declaration

C/C++
```
extern "C" short WINAPI VCGetDimShowLeadingZeros(short* iError);
extern "C" void WINAPI VCSetDimShowLeadingZeros(short* iError, short iF_);
```

Visual Basic
```
Declare Function VCGetDimShowLeadingZeros Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimShowLeadingZeros Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi
```
function VCGetDimShowLeadingZeros(var iError: Integer):Integer; far;
procedure VCSetDimShowLeadingZeros(var iError: Integer; iF_: Integer); far;
```

Parameters tf - show leading zeros
0 - do not show leading zeros.
1 - show leading zeros.

See Also VCGetDimFont, VCGetTextFontName, VCIsFontNameVText, VCIsTextFontVText

VCGetDimShowUnits
VCSetDimShowUnits

Version 2.0

Description Specifies if the abbreviation for the unit type is displayed after the number. If the units are Feet and Inches, the units are displayed regardless of this setting.

Declaration

C/C++
```
extern "C" short WINAPI VCGetDimShowUnits(short* iError);
extern "C" void WINAPI VCSetDimShowUnits(short* iError, short iF_);
```

Visual Basic
```
Declare Function VCGetDimShowUnits Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimShowUnits Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi
```
function VCGetDimShowUnits(var iError: Integer):Integer; far;
procedure VCSetDimShowUnits(var iError: Integer; iF_: Integer); far;
```

Parameters tf - show dimension units
0 - do not show dimesnion units.
1 - show dimension units.

Notes With Corel Visual CADD, you can specify if you want the units to be displayed or if you only want the dimension numbers to be displayed. The current units are set by VCGetDimDistFormat, such as meters, inches, or Fraction Feet. Remember that the units for feet and inches automatically appear, no matter what the settings are for VCGetDimShowUnits.

See Also VCGetDimDistFormat, VCGetDimShowDash, VCGetSecondaryDistFormat, VCGetDimFont, VCGetDimShowLeadingZeros, VCGetTextFontName

VCGetDimTextAspect
VCSetDimTextAspect

Version 2.0

Description Specifies the current text aspect ratio setting for dimensions. The text aspect ratio is the proportion of the text height to the text width.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimTextAspect(short* iError);
extern "C" void WINAPI VCGetDImTextAspectBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimTextAspect(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetDImTextAspectBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetDImTextAspect Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDImTextAspectBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDImTextAspect(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the current text aspect ratio.

See Also VCGetDimTextBold, VCGetDimTextCharSpace, VCGetDimTextColor, VCGetDimTextFontName, VCGetDimTextHeight, VCGetDimTextItalic, VCGetDimTextItalicValue, VCGetDimTextLayer, VCGetDimTextLineSpace, VCGetDimTextProSpacing, VCGetDimTextRot, VCGetDimTextString, VCGetDimTextUnderline

VCGetDimTextBold
VCSetDimTextBold

Version 2.0

Description Specifies if the dimension text is to be bold. The bold command only works with True type fonts.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetDimTextBold(short* iError);
extern "C" void WINAPI VCSetDimTextBold(short* iErrors, short i);
```

Visual Basic
```
Declare Function VCGetDimTextBold Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextBold Lib "VCMAIN32.DLL" (iErrors As Integer, ByVal i As Integer)
```

Delphi
```
function VCGetDimTextBold(var iError: Integer):Boolean; far;
procedure VCSetDimTextBold(var iErrors: Integer; i: Integer); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes Depending on what type of font is being used and how the font is defined, then you might be able to make the text look bold. Corel Visual CADD utilizes both TrueType Fonts and built in vector fonts. The vector fonts can be converted from other font formats such as .SHX and .FNT. When working with text entities it is important to understand the type of font being used. Certain settings such as Bold, Italic and Underline only effect TrueType Fonts while others such as Italic value are designed for vector fonts. To make a vector font look bold, change its line width to a large value. Therefore, when altering the settings of an existing text entity it is necessary to determine the type of font in order to apply the appropriate settings. VCIsFontNameVText determines if the specified font is a Corel Visual CADD vector font.

See Also VCGetDimFont, VCGetDimTextItalic, VCGetDimTextUnderline, VCIsFontNameVText

VCGetDimTextCharSpace
VCSetDimTextCharSpace

Version 2.0

Description Character spacing is the amount of space that appears between characters in a text string. It determines if the characters in a word are crowded or spread out. The value is a percentage of the characters height and applies only to vector fonts.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimTextCharSpace(short* iError);
extern "C" void WINAPI VCGetDimTextCharSpaceBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimTextCharSpace(short* iError, double dCharSpacing);
```

Visual Basic:
```
Declare Sub VCGetDimTextCharSpaceBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetDimTextCharSpace Lib "VCMAIN32.DLL" (iError As Integer, ByVal
dCharSpacing As Double)
```

Delphi:
```
procedure VCGetDimTextCharSpaceBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDimTextCharSpace(var iError: Integer; dCharSpacing: Double); far;
```

Parameters *dCharSpacing* - the charcter spacing as a decimal percentage (i.e. 1.5 is 150%)

See Also VCGetDimTextAspect, VCGetDimTextBold, VCGetDimTextHeight, VCGetDimTextItalic, VCGetDimTextItalicValue, VCGetDimTextLineSpace, VCGetDimTextProSpacing, VCGetDimTextRotationType, VCGetDimTextUnderline

VCGetDimTextFillVText
VCSetDimTextFillVText

Version 2.0

Description Specifies if vector fonts are filled in dimensions.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetDimTextFillVText(short* iError);
extern "C" void WINAPI VCSetDimTextFillVText(short* iError, vbool tf);
```

Visual Basic
```
Declare Function VCGetDimTextFillVText Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextFillVText Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetDimTextFillVText(var iError: Integer):Boolean; far;
procedure VCSetDimTextFillVText(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes Depending on what type of font is being used and how the font is defined, then you might be able to modify its appearance. Corel Visual CADD utilizes both TrueType Fonts and built in vector fonts. The vector fonts can be converted from other font formats such as .SHX and .FNT. When working with text entities it is important to understand the type of font being used. Certain settings such as Bold, Italic and Underline only effect TrueType Fonts while others such as Italic value are designed for vector fonts. VCGetDimTextFillVText will fill vector fonts that are closed outline fonts. Therefore, when altering the settings of an existing text entity it is necessary to determine the type of font in order to apply the appropriate settings. VCIsFontNameVText determines if the specified font is a Corel Visual CADD vector font.

See Also VCIsFontNameVText, VCGetDimFont

VCGetDimTextItalic
VCSetDimTextItalic

Version 2.0

Description Specifies if the test is to have an italic appearance. Will only work with True type fonts.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetDimTextItalic(short* iError);
extern "C" void WINAPI VCSetDimTextFillVText(short* iError, vbool tf);
```

Visual Basic
```
Declare Function VCGetDimTextItalic Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextItalic Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetDimTextItalic(var iError: Integer):Boolean; far;
procedure VCSetDimTextFillVText(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes Depending on what type of font is being used and how the font is defined, then you might be able to modify its appearance. Corel Visual CADD utilizes both TrueType Fonts and built in vector fonts. The vector fonts can be converted from other font formats such as .SHX and .FNT. When working with text entities it is important to understand the type of font being used. Certain settings such as Bold, Italic and Underline only effect TrueType Fonts while others such as Italic value are designed for vector fonts. VCGetDimTextItalic determines if the specified font is to be italicized or not (Remember that VCGetDimTextItalic only works with True type fonts). Therefore, when altering the settings of an existing text entity it is necessary to determine the type of font in order to apply the appropriate settings. VCIsFontNameVText determines if the specified font is a Corel Visual CADD vector font. To italicize vector fonts, use VCGetDimTextItalicAng.

See Also VCGetDimFont, VCGetDimTextBold, VCGetDimTextItalicAng, VCGetDimTextUnderline, VCIsFontNameVText

VCGetDimTextItalicAng
VCSetDimTextItalicAng

Version 2.0

Description Vector fonts can be slanted to emulate italics.

Declaration

C/C++:

```
extern "C" double WINAPI VCGetDimTextItalicValue(short* iError, double* dI);
extern "C" void WINAPI VCGetDimTextItalicValueBP(short* iError, double* dI);
extern "C" void WINAPI VCSetDimTextItalicValue(short* iError, double dI);
```

Visual Basic:

```
Declare Sub VCGetDimTextItalicValueBP Lib "VCMAIN32.DLL" (iError As Integer, dI As Double)
Declare Sub VCSetDimTextItalicValue Lib "VCMAIN32.DLL" (iError As Integer, ByVal dI As Double)
```

Delphi:

```
procedure VCGetDimTextItalicValueBP(var iError: Integer; var dI: Double); far;
procedure VCSetDimTextItalicValue(var iError: Integer; dI: Double); far;
```

Parameters *dI* - the angle in radians for the slant

Notes The number must range between 45 and -45 degrees. As with all angle functions, the angle is specified in radians. A negative number will slant the text backwards.

See Also VCGetDimTextAspect, VCGetDimTextBold, VCGetDimTextHeight, VCGetDimTextItalic, VCGetDimTextItalicValue, VCGetDimTextLineSpace, VCGetDimTextProSpacing, VCGetDimTextRotationType, VCGetDimTextUnderline

VCGetDimTextProSpacing
VCSetDimTextProSpacing

Version 2.0

Description Vector text character spacing can be forced to monospace or proportional spacing. Monospace is a characteristic of typewriter output and all characters will use the same amount of space regardless of their width and height.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetDimTextProSpacing(short* iError);
extern "C" void WINAPI VCSetDimTextProSpacing(short* iError, BOOL b);
```

Visual Basic:
```
Declare Function VCGetDimTextProSpacing Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextProSpacing Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimTextProSpacing(var iError: Integer):Boolean; far; external'VCMAIN';
procedure VCSetDimTextProSpacing(var iError: Integer; b: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDimTextAspect, VCGetDimTextBold, VCGetDimTextHeight, VCGetDimTextItalic, VCGetDimTextItalicValue, VCGetDimTextLineSpace, VCGetDimTextProSpacing, VCGetDimTextRotationType, VCGetDimTextUnderline

VCGetDimTextLineSpace
VCSetDimTextLineSpace

Version 2.0

Description The between text line spacing as a percentage of current text height.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimTextLineSpace(short* iError);
extern "C" void WINAPI VCGetDimTextLineSpaceBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimTextLineSpace(short* iError, double dLineSpacing);
```

Visual Basic:
```
Declare Sub VCGetDimTextLineSpaceBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetDimTextLineSpace Lib "VCMAIN32.DLL" (iError As Integer, ByVal dLineSpacing
As Double)
```

Delphi:
```
procedure VCGetDimTextLineSpaceBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDimTextLineSpace(var iError: Integer; dLineSpacing: Double); far;
```

Parameters *dRet* - spacing between the lines.

See Also VCGetDimTextAspect, VCGetDimTextBold, VCGetDimTextHeight, VCGetDimTextItalic, VCGetDimTextItalicValue, VCGetDimTextLineSpace, VCGetDimTextProSpacing, VCGetDimTextRotationType, VCGetDimTextUnderline

VCGetDimTextUnderline
VCSetDimTextUnderline

Version 2.0

Description Specifies if the dimension text is to be underline. Only works with True type fonts.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetDimTextUnderline(short* iError);
extern "C" void WINAPI VCSetDimTextUnderline(short* iError, vbool tf);
```

Visual Basic
```
Declare Function VCGetDimTextUnderline Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Declare Sub VCSetDimTextUnderline Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetDimTextUnderline(var iError: Integer):Boolean; far;
procedure VCSetDimTextUnderline(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes Depending on what type of font is being used and how the font is defined, then you might be able to modify its appearance. Corel Visual CADD utilizes both TrueType Fonts and built in vector fonts. The vector fonts can be converted from other font formats such as .SHX and .FNT. When working with text entities it is important to understand the type of font being used. Certain settings such as Bold, Italic and Underline only effect TrueType Fonts while others such as Italic value are designed for vector fonts. VCGetDimTextUnderline determines if the specified font is underlined or not (Remember that VCGetDimTextUnderline only works with True type fonts). Therefore, when altering the settings of an existing text entity it is necessary to determine the type of font in order to apply the appropriate settings. VCIsFontNameVText determines if the specified font is a Corel Visual CADD vector font.

See Also VCGetDimFont, VCGetDimTextBold, VCGetDimTextItalic, VCIsFontNameVText

VCGetDimExtAbove
VCSetDimExtAbove

Version 1.2

Description The dimension extension above distance is the length of the extension lines above the dimension line.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimExtAbove(short* iError);
extern "C" void WINAPI VCGetDimExtAboveBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimExtAbove(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetDimExtAboveBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetDimExtAbove Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDimExtAboveBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDimExtAbove(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the offset distance value.

See Also VCDimGetDimMode, VCGetDimUnitConversionFactor, VCDimGetDimExtStrerch, VCDimGetDimProximity, VCGetDimExtBelow, VCGetDimExtOffset

VCGetDimExtBelow
VCSetDimExtBelow

Version 1.2

Description The dimension extension below distance is the length of the extension lines below the dimension line when utilizing proximity fixed.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimExtBelow(short* iError);
extern "C" void WINAPI VCGetDimExtBelowBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimExtBelow(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetDimExtBelowBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetDimExtBelow Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDimExtBelowBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDimExtBelow(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the offset distance value.

See Also VCGetDimUnitConversionFactor, VCGetDimExtAbove, VCDimGetDimProximity, VCGetDimExtOffset

VCGetDimExtOffset
VCSetDimExtOffset

Version 1.2

Description The offset distance is the distance from the dimensioned point or object to the start of the extension line. Applies only if proximity fixed is on, then the dimension line is placed a fixed distance from the dimensioned object equal to the Below distance plus the Offset distance.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimExtOffset(short* iError);
extern "C" void WINAPI VCGetDimExtOffsetBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimExtOffset(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetDimExtOffsetBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetDimExtOffset Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDimExtOffsetBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDimExtOffset(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the offset distance value.

See Also VCGetDimUnitConversionFactor, VCGetDimExtAbove, VCGetDimExtBelow, VCDimGetDimProximity

VCGetDimFont
VCSetDimFont

Version 1.2

Description The font used for dimension placements.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimFont(short* iError, char* pS);
extern "C" void WINAPI VCSetDimFont(short* iError, char* pS);
```

Visual Basic:
```
Declare Function VCGetDimFont Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String) As Integer
Declare Sub VCSetDimFont Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String)
```

Delphi:
```
function VCGetDimFont(var iError: Integer; pS: PChar):Integer; far;
procedure VCSetDimFont(var iError: Integer; pS: PChar); far;
```

Parameters *pS* - the font name

Notes The available font names can be determined with VCGetFontName and VCGetFontNameCount.

See Also VCGetDimTextCentered, VCGetDimTextHeight, VCGetFontName, VCGetFontNameCount

VCGetDimItemColor
VCSetDimItemColor

Version 1.2

Description The elements that make up a dimension include the dimension line, left and right extension lines, left and right arrow and the dimension text. The API gives complete control over the visual properties of each of the dimension elements independent of each other. Changing the properties of dimension elements will not effect previously drawn dimensions.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimItemColor(short* iError, short i);
extern "C" void WINAPI VCSetDimItemColor(short* iError, short i, short j);
```

Visual Basic:
```
Declare Function VCGetDimItemColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer) As Integer
Declare Sub VCSetDimItemColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer, ByVal j As Integer)
```

Delphi:
```
function VCGetDimItemColor(var iError: Integer; i: Integer):Integer; far;
procedure VCSetDimItemColor(var iError: Integer; i: Integer; j: Integer);far;
```

Parameters
i - the dimension item number.
0 - Dimension Line
1- Left Arrow
2 - Right Arrow
3 - Left Extension
4 - Right Extension
5 - Dim Text
j - the color value from 0 to 255.

See Also VCGetDimItemLineWidth, VCGetDimItemLineType, VCGetDimDisplayItemName

VCGetDimItemLineType
VCSetDimItemLineType

Version 1.2

Description Used to get the line type of the selected dimension item. Each dimension item can be have a The elements that make up a dimension include the dimension line, left and right extension lines, left and right arrow and the dimension text. The API gives complete control over the visual properties of each of the dimension elements independent of each other. Changing the properties of dimension elements will not effect previously drawn dimensions.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimItemLineType(short* iError, short i);
extern "C" void WINAPI VCSetDimItemLineType(short* iError, short i, short j);
```

Visual Basic:
```
Declare Function VCGetDimItemLineType Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer) As Integer
Declare Sub VCSetDimItemLineType Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer, ByVal j As Integer)
```

Delphi:
```
function VCGetDimItemLineType(var iError: Integer; i: Integer):Integer; far;
procedure VCSetDimItemLineType(var iError: Integer; i: Integer; j: Integer); far;
```

Parameters

i - the dimension item number.
0 - Dimension Line
1- Left Arrow
2 - Right Arrow
3 - Left Extension
4 - Right Extension
5 - Dim Text
j - the linetype value.

See Also VCGetDimItemLineWidth, VCGetDimDisplayItemName, VCGetDimItemShow, VCGetDimItemLineType, VCGetDimItemColor

VCGetDimItemLineWidth
VCSetDimItemLineWidth

Version 1.2

Description The elements that make up a dimension include the dimension line, left and right extension lines, left and right arrow and the dimension text. The API gives complete control over the visual properties of each of the dimension elements independent of each other. Changing the properties of dimension elements will not effect previously drawn dimensions.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimItemLineWidth(short* iError, short i);
extern "C" void WINAPI VCSetDimItemLineWidth(short* iError, short i, short j);
```

Visual Basic:
```
Declare Function VCGetDimItemLineWidth Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer) As Integer
Declare Sub VCSetDimItemLineWidth Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer, ByVal j As Integer)
```

Delphi:
```
function VCGetDimItemLineWidth(var iError: Integer; i: Integer):Integer; far;
procedure VCSetDimItemLineWidth(var iError: Integer; i: Integer; j: Integer); far;
```

Parameters

i - the dimension item number.
0 - Dimension Line
1 - Left Arrow
2 - Right Arrow
3 - Left Extension
4 - Right Extension
5 - Dim Text
j - the width value.

See Also VCGetDimDisplayItemName, VCGetDimItemLineType, VCGetDimItemShow, VCGetDimItemLineType, VCGetDimItemColor

VCGetDimItemShow
VCSetDimItemShow

Version 1.2

Description The elements that make up a dimension include the dimension line, left and right extension lines, left and right arrow and the dimension text. The API gives complete control over the visual properties of each of the dimension elements independent of each other. Changing the properties of dimension elements will not effect previously drawn dimensions. Specifies if the selected dimension item is shown when placing or editing the dimension.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetDimItemShow(short* iError, short i);
extern "C" void WINAPI VCSetDimItemShow(short* iError, short i, vbool tf);
```

Visual Basic:
```
Declare Function VCGetDimItemShow Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
As Integer
Declare Sub VCSetDimItemShow Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer, ByVal
tf As Integer)
```

Delphi:
```
function VCGetDimItemShow(var iError: Integer; i: Integer):Boolean; far;
procedure VCSetDimItemShow(var iError: Integer; i: Integer; tf: Boolean); far;
```

Parameters
i - the dimension item number.
0 - Dimension Line
1 - Left Arrow
2 - Right Arrow
3 - Left Extension
4 - Right Extension
5 - Dim Text
tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDimItemLineWidth, VCGetDimItemLineType, VCGetDimDisplayItemName, VCGetDimItemColor

VCGetDimLayer
VCSetDimLayer

Version 1.2

Description Corel Visual CADD will maintain a layer index for dimensions independent of the current layer. Even though the layer dimension may be specified, VCGetDimUseDimLayer must be specified to activate the dimension layer.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimLayer(short* iError);
extern "C" void WINAPI VCSetDimLayer(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetDimLayer Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimLayer Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetDimLayer(var iError: Integer):Integer; far;
procedure VCSetDimLayer(var iError: Integer; i: Integer); far
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDimUseDimLayer

VCGetDimLineAngle
VCSetDimLineAngle

Version 1.2

Description The dimension direction is the orientation used when measuring a distance and drawing a dimension line.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimLineAngle(short* iError);
extern "C" void WINAPI VCGetDimLineAngleBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimLineAngle(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetDimLineAngleBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetDimLineAngle Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDimLineAngleBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDimLineAngle(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the value of the dimension line angle in radians.

Notes Measured distances are projected onto the dimension direction. *Horizontal* - Only the horizontal component of the entity is measured. *Vertical* - Only the vertical component of the entity is measured. *Aligned* - The dimension line is placed parallel to the entity. Aligned dimensions always represent the true length of the entity. *Angle* - Sets the dimension to a specified angle. The distance measured is the length of the entity projected onto the defined angle. VCGetDimLineAngle specifies the dimension angle.

See Also VCGetDimLineDirect

VCGetDimLineDirect
VCSetDimLineDirect

Version 1.2

Description The dimension direction is the orientation used when measuring a distance and drawing a dimension line.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimLineDirect(short* iError);
extern "C" void WINAPI VCSetDimLineDirect(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetDimLineDirect Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimLineDirect Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimLineDirect(var iError: Integer):Integer; far;
procedure VCSetDimLineDirect(var iError: Integer; b: Integer); far;
```

Parameters *b* - the value of the dimension line direction.
1 - DIMALIGNED
2 - DIMHORIZONTAL
3 - DIMVERTICAL
4 - DIMATANANGLE

Notes Measured distances are projected onto the dimension direction. *Horizontal* - Only the horizontal component of the entity is measured. *Vertical* - Only the vertical component of the entity is measured. *Aligned* - The dimension line is placed parallel to the entity. Aligned dimensions always represent the true length of the entity. *Angle* - Sets the dimension to a specified angle. The distance measured is the length of the entity projected onto the defined angle.

See Also VCGetDimLineAngleVCDimDirectionMode

VCGetDimLineText
VCSetDimLineText

Version 1.2

Description The location of the dimension line text.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimLineText(short* iError);
extern "C" void WINAPI VCSetDimLineText(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetDimLineText Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimLineText Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimLineText(var iError: Integer):Integer; far;
procedure VCSetDimLineText(var iError: Integer; b: Integer); far;
```

Parameters *b* - the value of the dimension line text.
0 - DIMTEXTINLINE
1 - DIMTEXTABOVELINE
2 - DIMTEXTFREEFLOAT

Notes *In Line* - Dimension text is inserted and centered in a break in the dimension line. The gap from the dimension line to the dimension text is equal to the Offset distance. *Above Line* - The dimension text is placed parallel to and offset from the dimension line. Automatically sets the dimension mode to Aligned mode. *Free Float* - Places the dimension text at the point specified in the VCAdd routines.

See Also VCGetDimTextOverwriteString, VCGetDimTextSuffixString, VCGetDimTextPrefixString, VCGetDimTextOverwrite, VCGetDimTextSuffix, VCGetDimTextPrefix, VCGetDimTextCentered, VCGetDimTextRotationType, VCGetDimTextSuffixString

VCGetDimTextCentered
VCSetDimTextCentered

Version 1.2

Description Dimension text is placed at the midpoint of the dimension line, regardless of the orientation or mode.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetDimTextCentered(short* iError);
extern "C" void WINAPI VCSetDimTextCentered(short* iError, vbool tf);
```

Visual Basic:
```
Declare Function VCGetDimTextCentered Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextCentered Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetDimTextCentered(var iError: Integer):Boolean; far;
procedure VCSetDimTextCentered(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDimTextHeight, VCGetDimFont, VCGetDimTextVertSpace, VCGetDimTextRotationType

VCGetDimTextHeight
VCSetDimTextHeight

Version 1.2

Description The dimension text height in inches.

Declaration

C/C++:

```
extern "C" double WINAPI VCGetDimTextHeight(short* iError);
extern "C" void WINAPI VCGetDimTextHeightBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimTextHeight(short* iError, double dRet);
```

Visual Basic:

```
Declare Sub VCGetDimTextHeightBP Lib "VCMAIN32.DLL" (iError As Integer, d As Double)
Declare Sub VCSetDimTextHeight Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:

```
procedure VCGetDimTextHeightBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDimTextHeight(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the value of the text height in inches.

See Also VCGetDimTextCentered, VCGetDimFont, VCGetDimTextVertSpace, VCGetDimHorizSpace, VCGetDimTextRotationType

VCGetDimTextHorizSpace
VCSetDimTextHorizSpace

Version 1.2

Description The dimension text horizontal spacing.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimTextHorizSpace(short* iError);
extern "C" void WINAPI VCGetDimTextHorizSpaceBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimTextHorizSpace(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetDimTextHorizSpaceBP Lib "VCMAIN32.DLL" (iError As Integer, d As Double)
Declare Sub VCSetDimTextHorizSpace Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDimTextHorizSpaceBP(var iError: Integer; var dRet: Double);
procedure VCSetDimTextHorizSpace(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the value of the inline spacing.

See Also VCGetDimTextCentered, VCGetDimTextHeight, VCGetDimFont, VCGetDimTextVertSpace, VCGetDimTextRotationType

VCGetDimTextOverwrite
VCSetDimTextOverwrite

Version 1.2

Description The dimension value is calculated automatically by Corel Visual CADD. The "Overwrite" option allows an application to completely replace the calculated dimension with an input string. VCSetDimTextOverwriteString sets the string while VCSetDimTextOverwrite tells Corel Visual CADD to replace the calculated value with the string.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetDimTextOverwrite(short* iError);
extern "C" void WINAPI VCSetDimTextOverwrite(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetDimTextOverwrite Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextOverwrite Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimTextOverwrite(var iError: Integer):Boolean; far;
procedure VCSetDimTextOverwrite(var iError: Integer; b: Integer); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDimTextOverwriteString, VCGetDimTextSuffixString, VCGetDimTextPrefixString, VCGetDimTextOverwrite, VCGetDimTextSuffix, VCGetDimTextPrefix, VCGetDimTextCentered, VCGetDimTextRotationType, VCGetDimTextSuffixString

VCGetDimTextOverwriteString
VCSetDimTextOverwriteString

Version 1.2

Description The dimension value is calculated automatically by Corel Visual CADD. The "Overwrite" option allows an application to completely replace the calculated dimension with an input string. VCSetDimTextOverwriteString sets the string while VCSetDimTextOverwrite tells Corel Visual CADD to replace the calculated value with the string.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimTextOverwriteString(short* iError, char* pB);
extern "C" void WINAPI VCSetDimTextOverwriteString(short* iError, char* pB);
```

Visual Basic:
```
Declare Function VCGetDimTextOverwriteString Lib "VCMAIN32.DLL" (iError As Integer, ByVal pB As String) As Integer
Declare Sub VCSetDimTextOverwriteString Lib "VCMAIN32.DLL" (iError As Integer, ByVal pB As String)
```

Delphi:
```
function VCGetDimTextOverwriteString(var iError: Integer; pB: PChar):Integer;
procedure VCSetDimTextOverwriteString(var iError: Integer; pB: PChar); far
```

Parameters *pB* - the value of the overwrite string.

See Also VCGetDimTextOverwriteString, VCGetDimTextSuffixString, VCGetDimTextPrefixString, VCGetDimTextOverwrite, VCGetDimTextSuffix, VCGetDimTextPrefix, VCGetDimTextCentered, VCGetDimTextRotationType, VCGetDimTextSuffixString

VCGetDimTextPrefix
VCSetDimTextPrefix

Version 1.2

Description The dimension settings allow a custom prefix or suffix to be added to the calculated dimension angle or distance without losing the associative property of the dimension.

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetDimTextPrefix(short* iError);
extern "C" void WINAPI VCSetDimTextPrefix(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetDimTextPrefix Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextPrefix Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimTextPrefix(var iError: Integer):Boolean; far;
procedure VCSetDimTextPrefix(var iError: Integer; b: Integer); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes The Strings can be set with VCSetDimTextSuffixString and VCSetDimTextPrefixString, while VCSetDimTextSuffix and VCSetDimTextPrefix indicate to use the strings in the dimension text.

See Also VCGetLeaderString, VCGetDimTextSuffixString, VCGetDimTextPrefixString, VCGetDimTextOverwrite, VCGetDimTextSuffix, VCGetDimTextPrefix

VCGetDimTextPrefixString
VCSetDimTextPrefixString

Version 1.2

Description The dimension settings allow a custom prefix or suffix to be added to the calculated dimension angle or distance without losing the associative property of the dimension.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimTextPrefixString(short* iError, char* pB);
extern "C" void WINAPI VCSetDimTextPrefixString(short* iError, char* pB);
```

Visual Basic:
```
Declare Function VCGetDimTextPrefixString Lib "VCMAIN32.DLL" (iError As Integer, ByVal pB As String) As Integer
Declare Sub VCSetDimTextPrefixString Lib "VCMAIN32.DLL" (iError As Integer, ByVal pB As String)
```

Delphi:
```
function VCGetDimTextPrefixString(var iError: Integer; pB: PChar):Integer;
procedure VCSetDimTextPrefixString(var iError: Integer; pB: PChar); far;
```

Parameters *pB* - the value of the prefix string.

Notes The Strings can be set with VCSetDimTextSuffixString and VCSetDimTextPrefixString, while VCSetDimTextSuffix and VCSetDimTextPrefix indicate to use the strings in the dimension text.

See Also VCGetLeaderString, VCGetDimTextSuffixString, VCGetDimTextPrefixString, VCGetDimTextOverwrite, VCGetDimTextSuffix, VCGetDimTextPrefix

VCGetDimTextRotationType
VCSetDimTextRotationType

Version 1.2

Description The dimension text orientation with respect to the dimension line.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimTextRotationType(short* iError);
extern "C" void WINAPI VCSetDimTextRotationType(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetDimTextRotationType Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextRotationType Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimTextRotationType(var iError: Integer):Integer; far;
procedure VCSetDimTextRotationType(var iError: Integer; b: Integer); far;
```

Parameters *b* - the value of the rotation type.
0 - Aligned
1 - Horizontal

Notes *Aligned* - The dimension text will be orientated parallel to the dimension line. This option is set automatically if the dimension text relationship to the dimension line is set to "Above".
Horizontal - The dimension text is placed horizontal regardless of the orientation of the dimension line. Applies only if the dimension text placement option is set to the "In Line" option.

See Also VCGetDimTextCentered, VCGetDimTextHeight, VCGetDimFont, VCGetDimTextVertSpace, VCGetDimTextHorizSpace

VCGetDimTextScale
VCSetDimTextScale

Version 1.2

Description A scaling factor that is applied to all dimensions to set the dimension text value.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimTextScale(short* iError);
extern "C" void WINAPI VCGetDimTextScaleBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimTextScale(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetDimTextScaleBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetDimTextScale Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDimTextScaleBP(var iError: Integer; var dRet: Double); far;
procedure VCSetDimTextScale(var iError: Integer; dRet: Double); far
```

Parameters *d* - the value of text scale.

Notes Can be used when details or drawings of mixed scales are plotted on the same sheet. This factor allows details to be blown up beyond "real world" size, and dimensioned correctly without having to reset the size related dimension properties.

See Also VCGetDimTextCentered, VCGetDimTextHeight, VCGetDimFont, VCGetDimTextVertSpace, VCGetDimTextHorizSpace, VCGetDimTextRotationType

VCGetDimTextSuffix
VCSetDimTextSuffix

Version 1.2

Description The dimension settings allow a custom prefix or suffix to be added to the calculated dimension angle or distance without losing the associative property of the dimension..

Declaration

C/C++:
```
extern "C" vbool WINAPI VCGetDimTextSuffix(short* iError);
extern "C" void WINAPI VCSetDimTextSuffix(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetDimTextSuffix Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextSuffix Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimTextSuffix(var iError: Integer):Boolean; far;
procedure VCSetDimTextSuffix(var iError: Integer; b: Integer); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes The Strings can be set with VCSetDimTextSuffixString and VCSetDimTextPrefixString, while VCSetDimTextSuffix and VCSetDimTextPrefix indicate to use the strings in the dimension text

See Also VCGetLeaderString, VCGetDimTextOverwrite, VCGetDimTextPrefix, VCGetDimTextPrefixString, VCGetDimTextSuffixString

VCGetDimTextSuffixString
VCSetDimTextSuffixString

Version 1.2

Description The dimension settings allow a custom prefix or suffix to be added to the calculated dimension angle or distance without losing the associative property of the dimension.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimTextSuffixString(short* iError, char* pB);
extern "C" void WINAPI VCSetDimTextSuffixString(short* iError, char* pB);
```

Visual Basic:
```
Declare Function VCGetDimTextSuffixString Lib "VCMAIN32.DLL" (iError As Integer, ByVal pB As String) As Integer
Declare Sub VCSetDimTextSuffixString Lib "VCMAIN32.DLL" (iError As Integer, ByVal pB As String)
```

Delphi:
```
function VCGetDimTextSuffixString(var iError: Integer; pB: PChar):Integer;
procedure VCSetDimTextSuffixString(var iError: Integer; pB: PChar); far
```

Parameters *pB* - the value of the suffix string.

Notes The Strings can be set with VCSetDimTextSuffixString and VCSetDimTextPrefixString, while VCSetDimTextSuffix and VCSetDimTextPrefix indicate to use the strings in the dimension text.

See Also VCGetLeaderString, VCGetDimTextOverwrite, VCGetDimTextPrefix, VCGetDimTextPrefixString, VCGetDimTextSuffix

VCGetDimTextTolDecimal
VCSetDimTextTolDecimal

Version 1.2

Description The number of decimal places to display on dimension tolerance values. Tolerances specify the allowable variations in a dimension and are often used in high precision work.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimTextTolDecimal(short* iError);
extern "C" void WINAPI VCSetDimTextTolDecimal(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetDimTextTolDecimal Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextTolDecimal Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimTextTolDecimal(var iError: Integer):Integer; far;
procedure VCSetDimTextTolDecimal(var iError: Integer; b: Integer); far
```

Parameters *b* - the number of decimal places

See Also VCGetDimTextTolLowerVal, VCGetDimTextTolType, VCGetDimTextTolUpperVal

VCGetDimTextTolLowerVal
VCSetDimTextTolLowerVal

Version 1.2

Description The maximum distance permitted for a lower tolerance variation in a dimension .Tolerances specify the allowable variations in a dimension and are often used in high precision work.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimTextTolLowerVal(short* iError);
extern "C" void WINAPI VCGetDimTextTolLowerValBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetDimTextTolLowerVal(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetDimTextTolLowerValBP Lib "VCMAIN32.DLL" (iError As Integer, d As Double)
Declare Sub VCSetDimTextTolLowerVal Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDimTextTolLowerValBP(var iError: Integer; var dRet: Double);
procedure VCSetDimTextTolLowerVal(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the value of the tolerance lower value setting.

See Also VCGetDimTextTolType, VCGetDimTextTolUpperVal, VCGetDimTextTolDecimal

VCGetDimTextTolType
VCSetDimTextTolType

Version 1.2

Description Tolerances specify the allowable variations in a dimension and are often used in high precision work. VCGetDimTextTolType sets what type of tolerance display is to be used.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDimTextTolType(short* iError);
extern "C" void WINAPI VCSetDimTextTolType(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetDimTextTolType Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimTextTolType Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetDimTextTolType(var iError: Integer):Integer; far;
procedure VCSetDimTextTolType(var iError: Integer; b: Integer); far;
```

Parameters *b* - the type of tolerance to be used on subsequent dimension placements.
0 - DIMNOTOLERANCE
1 - DIMSTACKEDMINMAX
2 - DIMSTACKEDVARIANCE
3 - DIMFIXEDVARIANCE

Notes The tolerance values can be shown in several methods: Stacked Variance - The calculated dimension is shown followed by allowable oversize tolerance "stacked" on top of the allowable undersize tolerance. Stacked min/max - The maximum allowable distance is stacked on top of the minimum allowable distance. The measured distance is not shown.

See Also VCGetDimTextTolLowerVal, VCGetDimTextTolUpperVal, VCGetDimTextTolDecimal

VCGetDimTextTolUpperVal
VCSetDimTextTolUpperVal

Version 1.2

Description The maximum distance permitted for a upper tolerance variation in a dimension .Tolerances specify the allowable variations in a dimension and are often used in high precision work.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetDimTextTolUpperVal(short* iError);
extern "C" void WINAPI VCGetDimTextTolUpperValBP(short* iError, double* d);
extern "C" void WINAPI VCSetDimTextTolUpperVal(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetDimTextTolUpperValBP Lib "VCMAIN32.DLL" (iError As Integer, d As Double)
Declare Sub VCSetDimTextTolUpperVal Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetDimTextTolUpperValBP(var iError: Integer; var dRet: Double);
procedure VCSetDimTextTolUpperVal(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the value of the tolerance upper value setting.

See Also VCGetDimTextTolLowerVal, VCGetDimTextTolType, VCGetDimTextTolDecimal

VCGetDimTextVertSpace
VCSetDimTextVertSpace

Version 1.2

Description The vertical spacing for dimension text display.

Declaration

C/C++:

```
extern "C" double WINAPI VCGetDimTextVertSpace(short* iError);
extern "C" void WINAPI VCGetDimTextVertSpaceBP(short* iError, double* d);
extern "C" void WINAPI VCSetDimTextVertSpace(short* iError, double d);
```

Visual Basic:

```
Declare Sub VCGetDimTextVertSpaceBP Lib "VCMAIN32.DLL" (iError As Integer, d As Double)
Declare Sub VCSetDimTextVertSpace Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:

```
procedure VCGetDimTextVertSpaceBP(var iError: Integer; var dRet: Double);
procedure VCSetDimTextVertSpace(var iError: Integer; dRet: Double); far
```

Parameters *d* - the value of the vertical spacing.

See Also VCGetDimTextCentered, VCGetDimTextHeight, VCGetDimFont, VCGetDimTextHorizSpace, VCGetDimTextRotationType

VCGetDimUnitConversionFactor

Version 2.0

Description Returns the conversion factor used by Corel Visual CADD to convert from the "inch" database to the current unit setting.

Declaration

C/C++ extern "C" void WINAPI VCGetDimUnitConversionFactor(short* iError, double* dRet);

Visual Basic Declare Sub VCGetDimUnitConversionFactor Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)

Delphi procedure VCGetDimUnitConversionFactor(var iError: Integer; var dRet: Double);

Parameters

Notes Since all data is currently stored in the Corel Visual CADD drawing database as inches, it is necessary to format any distances or areas in the units currently set in the program. VCGetDimUnitConversionFactor will find what the current units are and return a simple multiplier which will enable the conversion without having to case out each unit conversion in code.

See Also VCGetDisplayDistFormat, VCGetUnitConversionFactor

VCGetDimUseDimLayer
VCSetDimUseDimLayer

Version 1.2

Description Corel Visual CADD will maintain a layer index for dimensions independent of the current layer. Even though the layer dimension may be specified, VCGetDimUseDimLayer must be specified to activate the dimension layer. The dimension layer is set with VCSetDimLayer.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetDimUseDimLayer(short* iError);
extern "C" void WINAPI VCSetDimUseDimLayer(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetDimUseDimLayer Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimUseDimLayer Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetDimUseDimLayer(var iError: Integer):Boolean; far;
procedure VCSetDimUseDimLayer(var iError: Integer; tf: Boolean); far
```

Parameters *tf* **-** toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDimLayer

VCGetDisplayAngleFormat
VCSetDisplayAngleFormat

Version 1.2

Description Format for displaying angles as decimal degrees or degrees:minutes:seconds. If decimal degrees format is used, the number is decimal places displayed is determined by VCGetDisplayDecimalValue.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDisplayAngleFormat(short* iError);
extern "C" void WINAPI VCSetDisplayAngleFormat(short* iError, short iF_);
```

Visual Basic:
```
Declare Function VCGetDisplayAngleFormat Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDisplayAngleFormat Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi:
```
function VCGetDisplayAngleFormat(var iError: Integer):Integer; far;
procedure VCSetDisplayAngleFormat(var iError: Integer; iF_: Integer); far
```

Parameters *iF* - determines angular format to be used.
9 - Angle and Degrees.
10 - Degrees Minutes Seconds.

See Also VCGetDisplayDistFormat, VCGetDisplayShowUnits, VCGetDisplayShowLeadingZeros, VCGetDisplayFractionalValue, VCGetDisplayDecimalValue

VCGetDisplayDecimalValue
VCSetDisplayDecimalValue

Version 1.2

Description The number of digits displayed to the right of the decimal point.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDisplayDecimalValue(short* iError);
extern "C" void WINAPI VCSetDisplayDecimalValue(short* iError, short iF_);
```

Visual Basic:
```
Declare Function VCGetDisplayDecimalValue Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDisplayDecimalValue Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi:
```
function VCGetDisplayDecimalValue(var iError: Integer):Integer; far;
procedure VCSetDisplayDecimalValue(var iError: Integer; iF_: Integer); far
```

Parameters *iF* - the number of decimal places to be displayed.

Notes The valid range is 0 -8. Corel Visual CADD calculates and stores real numbers to a precision of 16 significant digits. Setting decimal places or fractions affects only how the numbers are displayed, not how they are calculated or stored. Corel Visual CADD seperates the number of decimals that are displayed and the number of decimals that are used in theVCGetDimDeci

See Also VCGetDimDecimalValue, VCGetDisplayDistFormat, VCGetDisplayAngleFormat, VCGetDisplayShowUnits, VCGetDisplayShowLeadingZeros, VCGetDisplayFractionalValue

VCGetDisplayDistFormat
VCSetDisplayDistFormat

Version 1.2

Description Option to set the units to display coordinates and distances on the screen and to draw dimensions.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDisplayDistFormat(short* iError);
extern "C" void WINAPI VCSetDisplayDistFormat(short* iError, short iF_);
```

Visual Basic:
```
Declare Function VCGetDisplayDistFormat Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDisplayDistFormat Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi:
```
function VCGetDisplayDistFormat(var iError: Integer):Integer; far;
procedure VCSetDisplayDistFormat(var iError: Integer; iF_: Integer); far;
```

Parameters *if* - the display format
0 - Decimal Inches
1 - Decimal Feet
2 - Decimal Feet & Inches
3 - Fractional Inches
4 - Fractional Feet
5 - Fractional Feet & Inches
6 - Millimeter
7 - Centimeter
8 - Meter

See Also VCGetDisplayAngleFormat, VCGetDisplayShowUnits, VCGetDisplayShowLeadingZeros, VCGetDisplayFractionalValue, VCGetDisplayDecimalValue

VCGetDisplayFractionalValue
VCSetDisplayFractionalValue

Version 1.2

Description Returns an integer representing the denominator of the fractional display value. All decimal values will be rounded to the nearest fractional values represented by this denominator when displayed. This does not affect stored values, only the display of these values.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDisplayFractionalValue(short* iError);
extern "C" void WINAPI VCSetDisplayFractionalValue(short* iError, short iF_);
```

Visual Basic:
```
Declare Function VCGetDisplayFractionalValue Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDisplayFractionalValue Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi:
```
function VCGetDisplayFractionalValue(var iError: Integer):Integer; far;
procedure VCSetDisplayFractionalValue(var iError: Integer; iF_: Integer);far
```

Parameters *iF* - determines the denominator of the fractional value to be used.
2 - 1/2.
4 - 1/4.
8 - 1/8.
16 - 1/16.
32 - 1/32.
64 - 1/64.

See Also VCGetDisplayDistFormat, VCGetDisplayAngleFormat, VCGetDisplayShowUnits, VCGetDisplayShowLeadingZeros, VCGetDisplayDecimalValue

VCGetDisplayShowFractions
VCSetDisplayShowFractions

Version 1.2

Description Dimension fractions can be displayed as a single character (¼) or multiple characters separated by a slash (1/4).

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDisplayShowFractions(short* iError);
extern "C" void WINAPI VCSetDisplayShowFractions(short* iError, short iF_);
```

Visual Basic:
```
Declare Function VCGetDisplayShowFractions Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDisplayShowFractions Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi:
```
function VCGetDisplayShowFractions(var iError: Integer):Integer; far;
procedure VCSetDisplayShowFractions(var iError: Integer; iF_: Integer); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes This option is available only for vector fonts which is determined with VCIsFontNameVText.

See Also VCGetDisplayDistFormat, VCGetDisplayShowUnits, VCGetDisplayShowLeadingZeros, VCGetDisplayFractionalValue, VCGetDisplayDecimalValue

VCGetDisplayShowLeadingZeros
VCSetDisplayShowLeadingZeros

Version 1.2

Description When displaying decimal values between 1 and -1, it may be preferred to not display the leading zero - the single zero before the decimal point.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDisplayShowLeadingZeros(short* iError);
extern "C" void WINAPI VCSetDisplayShowLeadingZeros(short* iError, short iF_);
```

Visual Basic:
```
Declare Function VCGetDisplayShowLeadingZeros Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDisplayShowLeadingZeros Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi:
```
function VCGetDisplayShowLeadingZeros(var iError: Integer):Integer; far;
procedure VCSetDisplayShowLeadingZeros(var iError: Integer; iF_: Integer);far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDisplayDistFormat, VCGetDisplayAngleFormat, VCGetDisplayShowUnits, VCGetDisplayFractionalValue, VCGetDisplayDecimalValue

VCGetDisplayShowUnits
VCSetDisplayShowUnits

Version 1.2

Description Specifies if the abbreviation for the unit type is displayed after the number. If the units are Feet and Inches, the units are displayed regardless of this setting.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDisplayShowUnits(short* iError);
extern "C" void WINAPI VCSetDisplayShowUnits(short* iError, short iF_);
```

Visual Basic:
```
Declare Function VCGetDisplayShowUnits Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDisplayShowUnits Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi:
```
function VCGetDisplayShowUnits(var iError: Integer):Integer; far;
procedure VCSetDisplayShowUnits(var iError: Integer; iF_: Integer); far
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDisplayDistFormat, VCGetDisplayAngleFormat, VCGetDisplayShowLeadingZeros, VCGetDisplayFractionalValue, , VCGetDisplayDecimalValue

VCGetDistanceHandle
VCSetDistanceHandle

Version 2.0

Description Specifies to which window handle is to display the current distance display values from Corel Visual CADD.

Declaration

C/C++
```
extern "C" long WINAPI VCGetDistanceHandle();
extern "C" void WINAPI VCSetDistanceHandle(long hWnd_);
```

Visual Basic
```
Declare Function VCGetDistanceHandle Lib "VCMAIN32.DLL" () As Long
Declare Sub VCSetDistanceHandle Lib "VCMAIN32.DLL" (ByVal hWnd_ As Long)
```

Delphi
```
function VCGetDistanceHandle:Longint; far;
procedure VCSetDistanceHandle(hWnd_: Longint); far;
```

Parameters *hWnd* - the Windows handle to display the distance entries.

Notes The Corel Visual CADD interface utilizes several status displays for the current user. These include the command prompt. An X Y display, a distance and angle value along with a selection count. When building a custom interface it is often desired to present this same information to the user. Instead of creating the status display in the application, the API allows for any Windows handle to be used to display the data. By Using the routines VCSetDistanceHandle, VCSetXYHandle and VCSetMessageHandle the application can quickly include the information into a custom interface.

See Also VCGetXYHandle, VCGetMessageHandle

VCGetDllRunCmdLine
VCSetDllRunCmdLine

Version 2.0

Description The command line for the DLL function.

Declaration

C/C++
```
extern "C" short WINAPI VCGetDllRunCmdLine(short* iError, char* szPath);
extern "C" void WINAPI VCSetDllRunCmdLine(short* iError, char* szPath);
```

Visual Basic
```
Declare Function VCGetDllRunCmdLine Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As
Declare Sub VCSetDllRunCmdLine Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As
String)String) As Integer
```

Delphi
```
function VCGetDllRunCmdLine(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetDllRunCmdLine(var iError: Integer; szPath: PChar); far;
```

Parameters *return* - the length of the return string.
szPath - the command line argument to pass to the DLL.

Notes Corel Visual CADD can run functions from within a DLL through the scripting language. This allows developers to create add on applications in a Windows DLL format and then simply reference functions contained in the DLL. Corel Visual CADD will load the DLL into memory and access the specified function. Generally, this is simply done through the Visual CADD interface with the Assign Script command or the CMDEXT file. Please refer to Customizing Corel Visual CADD for more information this. An application can also launch the routines through the API.

In order to access the DLL function, Corel Visual CADD must know the DLL name, the name of the function and any command line arguments required. The command line arguments can only be passed as a character string. The engine then uses this information to launch the specified function.

See Also VCGetDllRunFunction, VCGetDllRunName, VCGetOleDllClassName, VCGetOleDllFunctionCmdLine, VCGetOleDllFunctionName, VCGetOleDllName

VCGetDllRunFunction
VCSetDllRunFunction

Version 2.0

Description Specifies the DLL function name to be run.

Declaration

C/C++
```
extern "C" short WINAPI VCGetDllRunFunction(short* iError, char* szPath);
extern "C" void WINAPI VCSetDllRunFunction(short* iError, char* szPath);
```

Visual Basic
```
Declare Function VCGetDllRunFunction Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As
Declare Sub VCSetDllRunFunction Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As
String)String) As Integer
```

Delphi
```
function VCGetDllRunFunction(var iError: Integer; szPath: PChar):Integer;
procedure VCSetDllRunFunction(var iError: Integer; szPath: PChar); far;
```

Parameters
return - the length of the return string.
szPath - the command line argument to pass to the DLL.

Notes
Corel Visual CADD can run functions from within a DLL through the scripting language. This allows developers to create add on applications in a Windows DLL format and then simply reference functions contained in the DLL. Corel Visual CADD will load the DLL into memory and access the specified function. Generally, this is simply done through the Visual CADD interface with the Assign Script command or the CMDEXT file. Please refer to Customizing Corel Visual CADD for more information this. An application can also launch the routines through the API.

In order to access the DLL function, Corel Visual CADD must know the DLL name, the name of the function and any command line arguments required. The command line arguments can only be passed as a character string. The engine then uses this information to launch the specified function.

See Also VCGetDllRunCmdLine, VCGetDllRunName, VCGetOleDllClassName, VCGetOleDllFunctionCmdLine, VCGetOleDllFunctionName, VCGetOleDllName

VCGetDllRunName
VCSetDllRunName

Version 2.0

Description Specifies the DLL name where the function is located.

Declaration

C/C++
extern "C" short WINAPI VCGetDllRunName(short* iError, char* szPath);
extern "C" void WINAPI VCSetDllRunName(short* iError, char* szPath);

Visual Basic
Declare Function VCGetDllRunName Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
Declare Sub VCSetDllRunName Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)As Integer

Delphi
function VCGetDllRunName(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetDllRunFunction(var iError: Integer; szPath: PChar); far;

Parameters
return - the length of the return string.
szPath - the command line argument to pass to the DLL.

Notes Corel Visual CADD can run functions from within a DLL through the scripting language. This allows developers to create add on applications in a Windows DLL format and then simply reference functions contained in the DLL. Visual CADD will load the DLL into memory and access the specified function. Generally, this is simply done through the Corel Visual CADD interface with the Assign Script command or the CMDEXT file. Please refer to Customizing Corel Visual CADD for more information this. An application can also launch the routines through the API.

In order to access the DLL function, Corel Visual CADD must know the DLL name, the name of the function and any command line arguments required. The command line arguments can only be passed as a character string. The engine then uses this information to launch the specified function.

See Also VCGetDllRunCmdLine, VCGetDllRunFunction, VCGetOleDllClassName, VCGetOleDllFunctionCmdLine, VCGetOleDllFunctionName, VCGetOleDllName

VCGetDrawFBoundary
VCSetDrawFBoundary

Version 1.2

Description The fill boundary option determines whether the boundary that contains the hatch should be displayed as part of the hatch or be invisible.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetDrawFBoundary(short* iError);
extern "C" void WINAPI VCSetDrawFBoundary(short* iError, BOOL tfB);
```

Visual Basic:
```
Declare Function VCGetDrawFBoundary Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDrawFBoundary Lib "VCMAIN32.DLL" (iError As Integer, ByVal tfB As Integer)
```

Delphi:
```
function VCGetDrawFBoundary(var iError: Integer):Boolean; far;
procedure VCSetDrawFBoundary(var iError: Integer; tfB: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDrawHBoundary, VCGetFillColor, VCGetHatchColor, VCGetFillDisplay, VCGetHatchDisplay

VCGetDrawHBoundary
VCSetDrawHBoundary

Version 1.2

Description The hatch boundary option determines whether the boundary that contains the hatch should be displayed as part of the hatch or be invisible.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetDrawHBoundary(short* iError);
extern "C" void WINAPI VCSetDrawHBoundary(short* iError, BOOL tfB);
```

Visual Basic:
```
Declare Function VCGetDrawHBoundary Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDrawHBoundary Lib "VCMAIN32.DLL" (iError As Integer, ByVal tfB As Integer)
```

Delphi:
```
function VCGetDrawHBoundary(var iError: Integer):Boolean; far;
procedure VCSetDrawHBoundary(var iError: Integer; tfB: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDrawFBoundary, VCGetFillColor, VCGetHatchColor, VCGetFillDisplay, VCGetHatchDisplay

VCGetDrawingName
VCSetDrawingName

Version 1.2

Description The active drawing name as presented in the Corel Visual CADD caption bar.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDrawingName(char* pName);
extern "C" void WINAPI VCSetDrawingName(char* pName);
```

Visual Basic:
```
Declare Function VCGetDrawingName Lib "VCMAIN32.DLL" (ByVal pName As String) As Integer
Declare Sub VCSetDrawingName Lib "VCMAIN32.DLL" (ByVal pName As String)
```

Delphi:
```
function VCGetDrawingName(pName: PChar):Integer; far;
procedure VCSetDrawingName(pName: PChar); far;
```

Parameters *pName* - a string representing the path and name of the current drawing.

See Also VCSaveDrawing, VCLoadDrawing

VCGetDWGPath
VCSetDWGPath

Version 1.2

Description The default file path for opening and saving AutoCAD DWG drawing files.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDWGPath(short* iError, char* szPath);
extern "C" void WINAPI VCSetDWGPath(short* iError, char* szPath);
```

Visual Basic:
```
Declare Function VCGetDWGPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
As Integer
Declare Sub VCSetDWGPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
```

Delphi:
```
function VCGetDWGPath(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetDWGPath(var iError: Integer; szPath: PChar); far;
```

Parameters *szPath* - the file path

See Also VCGetDXFPath, VCGetGCDPath, VCGetSYSPath, VCGetVCDPath, VCGetVCSPath, VCGetCMPPath, VCGetVCFPath

VCGetDXFPath
VCSetDXFPath

Version 1.2

Description The default file path for opening and saving DXF drawing files.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetDXFPath(short* iError, char* szPath);
extern "C" void WINAPI VCSetDXFPath(short* iError, char* szPath);
```

Visual Basic:
```
Declare Function VCGetDXFPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
As Integer
Declare Sub VCSetDXFPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
```

Delphi:
```
function VCGetDXFPath(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetDXFPath(var iError: Integer; szPath: PChar); far;
```

Parameters *szPath* - the file path

See Also VCGetDWGPath, VCGetGCDPath, VCGetSYSPath, VCGetVCDPath, VCGetVCSPath, VCGetCMPPath, VCGetVCFPath

VCGetEntityContourCount

Version 1.2

Description Returns the number of contours contained in the specified entity definition.

Declaration

C/C++: extern "C" short WINAPI VCGetEntityContourCount(short* iError, ENTITYHANDLE lH);

Visual Basic: Declare Function VCGetEntityContourCount Lib "VCMAIN32.DLL" (iError As Integer, ByVal lH As Long) As Integer

Delphi: function VCGetEntityContourCount(var iError: Integer; lH: Longint):Integer;

Parameters *lH* - the Corel Visual CADD entity handle used to reference each entity in the drawing.
Returns - the number of contours contained in the entity definition.

Notes VCGetEntityContourCount provides a method to determine the number of contours that define the boundary for a hatch or fill. VCGetEntitySubEntityCount gives you the number of entities that are in each contour. For example, say you have an exploded rectangle with a hatch inside it. The VCGetEntityContourCount would return 1(the number of contours that define the hatch boundary), while VCGetEntitySubEntityCount will return a value of 4 (the number of entities that make up the contour boundary).

See Also VCGetEntitySubEntityCount

VCGetEntitySubEntityCount

Version 1.2

Description Returns the number of entities within the specified contour of the specified entity.

Declaration

C/C++: extern "C" short FAR WINAPI VCGetEntitySubEntityCount(short* iError, ENTITYHANDLE lH, short iContour);

Visual Basic: Declare Function VCGetEntitySubEntityCount Lib "VCMAIN32.DLL" (iError As Integer, ByVal lH As Long, ByVal iContour As Integer) As Integer

Delphi: function VCGetEntitySubEntityCount(var iError: Integer; lH: Longint; iContour Integer):Integer; far;

Parameters *lH* - the Corel Visual CADD entity handle used to reference each entity in the drawing.
iContour - the contour containing the desired entity count.
Returns - the number of entities within the contour.

Notes VCGetEntityContourCount provides a method to determine the number of contours that define the boundary for a hatch or fill. VCGetEntitySubEntityCount gives you the number of entities that are in each contour. For example, say you have an exploded rectangle with a hatch inside it. The VCGetEntityContourCount would return 1(the number of contours that define the hatch boundary), while VCGetEntitySubEntityCount will return a value of 4 (the number of entities that make up the contour boundary).

See Also VCGetEntityContourCount

VCGetEntityUndoLevel

Version	2.0
Description	Retrieves the undo level for the current entity.
Declaration	
C/C++	extern "C" short WINAPI VCGetEntityUndoLevel(short* iError, ENTITYHANDLE hE);
Visual Basic	Declare Function VCGetEntityUndoLevel Lib "VCMAIN32.DLL" (iError As Integer, ByVal hE As Long) As Integer
Delphi	function VCGetEntityUndoLevel(var iError: Integer; hE: Longint):Integer; far;
Parameters	hE- handle to the entity
Notes	Each entity in the database maintains a flag indicating the level of undo. Corel Visual CADD supports unlimited undo operations and maintains this capability through this flag. The flag value changes with any modification done on the entity, for example moving the entity. An application can check this flag prior to an operation to ensure the user has not changed or altered an entity outside the applications control.
See Also	VCAppExit, VCBeginOperation, VCEndOperation, VCIsRedoable, VCUndo

VCGetErasedEntityCount

Version	2.0
Description	Returns the number of erased entities in the drawing.
Declaration	
C/C++	extern "C" long WINAPI VCGetErasedEntityCount(short* iError);
Visual Basic	Declare Function VCGetErasedEntityCount Lib "VCMAIN32.DLL" (iError As Integer) As Long
Delphi	function VCGetErasedEntityCount(var iError: Integer):Longint; far;
Parameters	Returns the number of erased entities in the drawing.
Notes	Entities erased from the drawing are tagged but remain in the database to allow for undo levels. These entities are removed with a drawing save or pack data command.

See Also

VCGetExeName
VCSetExeName

Version 1.2

Description The current executable setting to be run when using the script "run" command or the API call VCRun. VCRun is used to run any external application from Corel Visual CADD.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetExeName(short* iError, char* pS);
extern "C" void WINAPI VCSetExeName(short* iError, char* sz);
```

Visual Basic:
```
Declare Function VCGetExeName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String) As Integer
Declare Sub VCSetExeName Lib "VCMAIN32.DLL" (iError As Integer, ByVal sz As String)
```

Delphi:
```
function VCGetExeName(var iError: Integer; pS: PChar):Integer; far;
procedure VCSetExeName(var iError: Integer; sz: PChar); far;
```

Parameters *z* - the executable string to be set.

See Also VCGetDllRunCmdLine, VCGetDllRunName, VCGetDllRunFunction, VCDllRun, VCRun

VCGetExplodeContinuousLines
VCSetExplodeContinuousLines

Version 1.2

Description Certain entities can be exploded or broken into individual segments as they are placed. Not all shapes or entities can be exploded into component parts. A circle, for example, would not be affected by this command.

Declaration

C/C++:
```
extern "C" BYTE WINAPI VCGetExplodeContinuousLines(short* iError);
extern "C" void WINAPI VCSetExplodeContinuousLines(short* iError, BYTE tf);
```

Visual Basic:
```
Declare Function VCGetExplodeContinuousLines Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetExplodeContinuousLines Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetExplodeContinuousLines(var iError: Integer):Integer; far;
procedure VCSetExplodeContinuousLines(var iError: Integer; tf: Integer); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCExplode, VCGetAutoFillet, VCLineContinuous

VCGetFillColor
VCSetFillColor

Version 1.2

Description The fill color used when filling a boundary.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetFillColor(short* iError);
extern "C" void WINAPI VCSetFillColor(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetFillColor Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetFillColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetFillColor(var iError: Integer):Integer; far;
procedure VCSetFillColor(var iError: Integer; i: Integer); far
```

Parameters *i* - the color setting.

See Also VCGetDrawFBoundary, VCGetDrawHBoundary, VCGetHatchColor, VCGetFillDisplay, VCGetHatchDisplay

{button ,AL(`Adding a Hatch/Fill Entity',0,`',`')} Task Guide Examples

VCGetFillDisplay
VCSetFillDisplay

Version 1.2

Description Determine whether fill entities are displayed on the screen as well as in print and plot routines. Turning off the display will reduce the visual clutter and increase the speed of redraws.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetFillDisplay(short* iError);
extern "C" void WINAPI VCSetFillDisplay(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetFillDisplay Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetFillDisplay Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetFillDisplay(var iError: Integer):Boolean; far;
procedure VCSetFillDisplay(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDrawFBoundary, VCGetDrawHBoundary, VCGetFillColor, VCGetHatchColor, VCGetHatchDisplay

{button ,AL(`Adding a Hatch/Fill Entity',0,`',`')} Task Guide Examples

VCGetFillVText
VCSetFillVText

Version 1.2

Description Specifies if the vector outline fonts are to be filled with the current text color.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetFillVText(short* iError);
extern "C" void WINAPI VCSetFillVText(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetFillVText Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetFillVText Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetFillVText(var iError: Integer):Boolean; far;
procedure VCSetFillVText(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetFontList, VCGetFontName, VCGetFontNameCount, VCGetTextColor, VCIsTextFontVText, VCIsFontNameVText

VCGetFilterColor
VCSetFilterColor

Version 1.2

Description Specifies the filter color index.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetFilterColor(short* iError);
extern "C" void WINAPI VCSetFilterColor(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetFilterColor Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetFilterColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetFilterColor(var iError: Integer):Integer; far;
procedure VCSetFilterColor(var iError: Integer; i: Integer); far;
```

Parameters *I* - color index

Notes The API allows an application to filter entities prior to making selections. can be set based on entity kind, layer, color, line type and line width.

See Also VCGetFilterKind, VCGetFilterKind2, VCGetFilterLayer, VCGetFilterLineType, VCGetFilterName, VCGetFilterWidth, VCSetFilterMatch, VCSetFilterActive

VCGetFilterKind
VCSetFilterKind

Version 1.2

Description The filter entity kind.

Declaration

C/C++:
```
extern "C" BYTE WINAPI VCGetFilterKind(short* iError);
extern "C" void WINAPI VCSetFilterKind(short* iError, BYTE b);
```

Visual Basic:
```
Declare Function VCGetFilterKind Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetFilterKind Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetFilterKind(var iError: Integer):Integer; far;
procedure VCSetFilterKind(var iError: Integer; b: Integer); far;
```

Parameters *b* - the filter kind. See Appendix A for the Entity Type.

Notes The API allows you to filter entities prior to making selections. By setting a selection criteria based on entity properties and settings, the selection routine will only "capture" those objects meeting the filter criteria. The filter criteria can be set based on entity kind, layer, color, line type and line width.

See Also VCGetFilterKind2, VCGetFilterLayer, VCGetFilterLineType, VCGetFilterName, VCGetFilterWidth, VCSetFilterMatch, VCSetFilterActive

VCGetFilterKind2
VCSetFilterKind2

Version 1.2

Description The second filter kind allows a more detailed search set for arcs and lines byspecifying elliptical arcs or continuous lines for example.

Declaration

C/C++:
```
extern "C" BYTE WINAPI VCGetFilterKind2(short* iError);
extern "C" void WINAPI VCSetFilterKind2(short* iError, BYTE b);
```

Visual Basic:
```
Declare Function VCGetFilterKind2 Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetFilterKind2 Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetFilterKind2(var iError: Integer):Integer; far;
procedure VCSetFilterKind2(var iError: Integer; b: Integer); far;
```

Parameters *b* - the filter kind.

Notes The API allows you to filter entities prior to making selections. By setting a selection criteria based on entity properties and settings, the selection routine will only "capture" those objects meeting the filter criteria. The filter criteria can be set based on entity kind, layer, color, line type and line width.

See Also VCGetFilterKind, VCGetFilterLayer, VCGetFilterLineType, VCGetFilterName, VCGetFilterWidth, VCSetFilterMatch, VCSetFilterActive

VCGetFilterLayer
VCSetFilterLayer

Version 1.2

Description The selection filter layer.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetFilterLayer(short* iError);
extern "C" void WINAPI VCSetFilterLayer(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetFilterLayer Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetFilterLayer Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetFilterLayer(var iError: Integer):Integer; far;
procedure VCSetFilterLayer(var iError: Integer; i: Integer); far
```

Parameters *I* - the layer index

Notes The API allows you to filter entities prior to making selections. By setting a selection criteria based on entity properties and settings, the selection routine will only "capture" those objects meeting the filter criteria. The filter criteria can be set based on entity kind, layer, color, line type and line width.

See Also VCGetFilterKind, VCGetFilterKind2, VCGetFilterLineType, VCGetFilterName, VCGetFilterWidth, VCSetFilterMatch, VCSetFilterActive

VCGetFilterLineType
VCSetFilterLineType

Version 1.2

Description The selection filter line type.

Declaration

C/C++:
```
extern "C" void WINAPI VCSetFilterLineType(short* iError, short l);
extern "C" void WINAPI VCSetFilterLineType(short* iError, short i);
```

Visual Basic:
```
Declare Sub VCSetFilterLineType Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
Declare Sub VCSetFilterLineType Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetFilterLineType(var iError: Integer):Integer; far; external 'VCMAIN'
procedure VCSetFilterLineType(var iError: Integer; i: Integer); far;
```

Parameters *l* - the line type index

Notes The API allows an application to filter entities prior to making selections. can be set based on entity kind, layer, color, line type and line width.

See Also VCGetFilterKind, VCGetFilterKind2, VCGetFilterLayer, VCGetFilterName, VCGetFilterWidth, VCSetFilterMatch, VCSetFilterActive

VCGetFilterName
VCSetFilterName

Version 1.2

Description Certain filter entity types, symbols and text, allow a name for the exact symbol or font to be specified.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetFilterName(short* iError, char* sz);
extern "C" void WINAPI VCSetFilterName(short* iError, char* sz);
```

Visual Basic:
```
Declare Function VCGetFilterName Lib "VCMAIN32.DLL" (iError As Integer, ByVal sz As String) As Integer
Declare Sub VCSetFilterName Lib "VCMAIN32.DLL" (iError As Integer, ByVal sz As String)
```

Delphi:
```
function VCGetFilterName(var iError: Integer; sz: PChar):Integer; far;
procedure VCSetFilterName(var iError: Integer; sz: PChar); far;
```

Parameters *sz* **-** the filter name

Notes The API allows you to filter entities prior to making selections. By setting a selection criteria based on entity properties and settings, the selection routine will only "capture" those objects meeting the filter criteria. The filter criteria can be set based on entity kind, layer, color, line type and line width.

See Also VCGetFilterColor, VCGetFilterKind, VCGetFilterKind2, VCGetFilterLayer, VCGetFilterLineType, VCGetFilterWidth

VCGetFilterWidth
VCSetFilterWidth

Version 1.2

Description The selection filter line width.

Declaration

C/C++:
```
extern "C" void WINAPI VCSetFilterWidth(short* iError, short I);
extern "C" void WINAPI VCSetFilterWidth(short* iError, short i);
```

Visual Basic:
```
Declare Sub VCSetFilterWidth Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
Declare Sub VCSetFilterWidth Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetFilterWidth(var iError: Integer):Integer; far;
procedure VCSetFilterWidth(var iError: Integer; i: Integer); far;
```

Parameters *I-* the filter line width

Notes The API allows you to filter entities prior to making selections. By setting a selection criteria based on entity properties and settings, the selection routine will only "capture" those objects meeting the filter criteria. The filter criteria can be set based on entity kind, layer, color, line type and line width.

See Also VCGetFilterKind, VCGetFilterKind2, VCGetFilterLayer, VCGetFilterLineType, VCGetFilterName, VCSetFilterMatch, VCSetFilterActive

VCGetFilletPreview
VCSetFilletPreview

Version 1.2

Description Specifies the current state of the fillet preview toggle.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetFilletPreview(short* iError);
extern "C" void WINAPI VCSetFilletPreview(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetFilletPreview Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetFilletPreview Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetFilletPreview(var iError: Integer):Boolean; far;
procedure VCSetFilletPreview(var iError: Integer; tf: Boolean); far
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes When manually filleting intersections of lines, Corel Visual CADD can dynamically preview all available fillets. While this is a user-friendly device, it can be unacceptably slow on some machines. Moreover, some external applications may not want the mechanics of the operation to be visible to the user.

See Also VCGetFilletPreview, VCGetFilletRad

VCGetFilletRad
VCSetFilletRad

Version 1.2

Description The fillet radius affects both the fillet command which fillets two non-parallel lines and the auto fillet command which fillets double lines and continuous lines as they are constructed.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetFilletRad(short* iError);
extern "C" void WINAPI VCGetFilletRadBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetFilletRad(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetFilletRadBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetFilletRad Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetFilletRadBP(var iError: Integer; var dRet: Double); far;
procedure VCSetFilletRad(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the desired fillet radius.

See Also VCGetFilletPreview, VCGetAutoFillet

VCGetFontList

Version	1.2
Description	Returns all of the available fonts as one comma delimited string.
Declaration	
C/C++:	extern "C" long WINAPI VCGetFontList(short* iError, char* list);
Visual Basic:	Declare Function VCGetFontList Lib "VCMAIN32.DLL" (iError As Integer, ByVal list As String) As Long
Delphi:	function VCGetFontList(var iError: Integer; list: PChar):Longint; far;
Parameters	*list* - the returned font list. Returns the number of characters in the font list.
Notes	Since it can be time consuming to cycle through all the fonts available and determine the names of each of them, VCGetFontList was provided so the entire list can be retrieved at one time and then parsed by internal code to separate individual names. The names are separated by commas.
See Also	VCGetFontName, VCGetFontNameCount

{button ,AL(`Adding a Text Entity',0,`',`')} Task Guide Examples

VCGetFontName

Version 1.2

Description Retrieves the name of the font specified by the supplied index.

Declaration

C/C++: extern "C" short WINAPI VCGetFontName(short* iError, short iIndex, char* s);

Visual Basic: Declare Function VCGetFontName Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal s As String) As Integer

Delphi: function VCGetFontName(var iError: Integer; iIndex: Integer; s PChar):Integer; far;

Parameters *iIndex* - the index number of the font whose name you want.
s - the string containing the name of the specified font.

Notes When determining all the fonts available to the user, the program must first determine how many fonts exist and then parse though each index to retrieve the font name with VCGetFontName.

See Also VCGetFontList, VCGetFontNameCount

VCGetFontNameCount

Version 1.2

Description Retrieves the number of fonts currently loaded into Corel Visual CADD including vector fonts and True Type fonts.

Declaration

C/C++: extern "C" short WINAPI VCGetFontNameCount(short* iError);

Visual Basic: Declare Function VCGetFontNameCount Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCGetFontNameCount(var iError: Integer):Integer; far;

Parameters *Returns* - the number of fonts currently available to Corel Visual CADD.

Notes When determining all the fonts available to the user, the program must first determine how many fonts exist and then parse though each index to retrieve the font name with VCGetFontName.

See Also VCGetFontList, VCGetFontName

VCGetFunkeyCmdString
VCSetFunkeyCmdString

Version 1.2

Description Once it has been determined using VCIsScriptAssigned if a script has been assigned to a key sequence, VCGetFunkeyCmdString will retrieve that string, allowing the application to append or edit the string and reassign it to the same key or an unused one using VCSetFunkeyCmdString.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetFunkeyCmdString(char* szCmd, short iShift, short nVKey);
extern "C" BOOL WINAPI VCSetFunkeyCmdString(char* szCmd, short iShift, short nVKey);
```

Visual Basic:
```
Declare Function VCGetFunkeyCmdString Lib "VCMAIN32.DLL" (ByVal szCmd As String, ByVal
iShift As Integer, ByVal nVKey As Integer) As Integer
Declare Function VCSetFunkeyCmdString Lib "VCMAIN32.DLL" (ByVal szCmd As String, ByVal
iShift As Integer, ByVal nVKey As Integer) As Integer
```

Delphi:
```
function VCGetFunkeyCmdString(szCmd: PChar; iShift: Integer; nVKey Integer):Integer; far;
function VCSetFunkeyCmdString(szCmd: PChar; iShift: Integer; nVKey: Integer):Boolean; far;
```

Parameters

szCmd - set by the procedure to be the script text.
iShift - determines the state of the modifier keys.
0 - none.
1 - shift.
2 - ctrl.
3 - alt.
nVKey - the ASCII code representing the desired key.

See Also VCGetCmdStr, VCIsScriptAssigned, VCMacro

VCGetGCDDefaultHatchName
VCSetGCDDefaultHatchName

Version 1.2

Description The default Corel Visual CADD hatch pattern name used to convert GCD hatch entities.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetGCDDefaultHatchName(short* iError, char* szName);
extern "C" void WINAPI VCSetGCDDefaultHatchName(short* iError, char* szName);
```

Visual Basic:
```
Declare Function VCGetGCDDefaultHatchName Lib "VCMAIN32.DLL" (iErr As Integer, ByVal szName As String) As Integer
Declare Sub VCSetGCDDefaultHatchName Lib "VCMAIN32.DLL" (iErr As Integer, ByVal szName As String)
```

Delphi:
```
function VCGetGCDDefaultHatchName(var iError: Integer; szName: PChar):Integer;
procedure VCSetGCDDefaultHatchName(var iError: Integer; szName: PChar); far;
```

Parameters *szName* - the default hatch pattern name.
Returns - the length of the current setting.

Notes The hatch patterns from the GCD format cannot be used directly in Corel Visual CADD. These patterns must converted to either symbols or to a default hatch pattern and changed in Corel Visual CADD. VCGetKeepGCDHatch specifies if hatches are converted to a default hatch or recreated as a symbol definition.

See Also VCAddHatchEntity, VCGetKeepGCDHatch, VCGetKeepGCDFontName, VCGetKeepAcadFontName

{button ,AL(`Adding a Hatch/Fill Entity',0,`',`')} Task Guide Examples

VCGetGCDPath
VCSetGCDPath

Version 1.2

Description The default path for loading and saving Generic CADD .GCD Drawings.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetGCDPath(short* iError, char* szPath);
extern "C" void WINAPI VCSetGCDPath(short* iError, char* szPath);
```

Visual Basic:
```
Declare Function VCGetGCDPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
As Integer
Declare Sub VCSetGCDPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
```

Delphi:
```
function VCGetGCDPath(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetGCDPath(var iError: Integer; szPath: PChar); far
```

Parameters *sz* - string value for the path settings

See Also VCGetDWGPath, VCGetDXFPath, VCGetSYSPath, VCGetVCDPath, VCGetVCSPath, VCGetCMPPath, VCGetVCFPath

VCGetGridDisplay
VCSetGridDisplay

Version 1.2

Description A reference grid can displayed and set as a snap to aid the in placing entities. The grid can have specified distances between horizontally and vertically placed grid points. VCGetGridDisplay specifies the grid points are visible on the screen.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetGridDisplay(short* iError);
extern "C" void WINAPI VCSetGridDisplay(short* iError, BOOL tfDisp);
```

Visual Basic:
```
Declare Function VCGetGridDisplay Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetGridDisplay Lib "VCMAIN32.DLL" (iError As Integer, ByVal tfDisp As Integer)
```

Delphi:
```
function VCGetGridDisplay(var iError: Integer):Boolean; far;
procedure VCSetGridDisplay(var iError: Integer; tfDisp: Boolean); far;
```

Parameters f - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetGridOrigin, VCGetGridSize, VCGetGridSnap

VCGetGridOrigin
VCSetGridOrigin

Version 1.2

Description A reference grid can displayed and set as a snap to aid the in placing entities.

Declaration

C/C++:
```
extern "C" Point2D WINAPI VCGetGridOrigin(short* iError);
extern "C" void WINAPI VCGetGridOriginBP(short* iError, Point2D* pRet);
extern "C" void WINAPI VCSetGridOrigin(short* iError, Point2D dpOrg);
extern "C" void WINAPI VCSetGridOriginBP(short* iError, Point2D* dpOrg_);
```

Visual Basic:
```
Declare Sub VCGetGridOriginBP Lib "VCMAIN32.DLL" (iError As Integer, pRet As Point2d)
Declare Sub VCSetGridOriginBP Lib "VCMAIN32.DLL" (iError As Integer, dpOrg_ As Point2d)
```

Delphi:
```
procedure VCGetGridOriginBP(var iError: Integer; var pRet: Point2D); far;
procedure VCSetGridOriginBP(var iError: Integer; var dpOrg_: Point2D); far;
```

Parameters *dpOrg* - the packed coordinate pair representing the desired grid setting

Notes The grid can have specified distances between horizontally and vertically placed grid points. The grid is aligned to the point set with VCSetGridOrigin or by default to the drawing origin.

See Also VCGetGridDisplay, VCGetGridSize, VCGetGridSnap

VCGetGridSize
VCSetGridSize

Version 1.2

Description A reference grid can displayed and set as a snap to aid the in placing entities.

Declaration

C/C++:
```
extern "C" Point2D WINAPI VCGetGridSize(short* iError);
extern "C" void WINAPI VCGetGridSizeBP(short* iError, Point2D* pRet);
extern "C" void WINAPI VCSetGridSize(short* iError, Point2D dpSize);
extern "C" void WINAPI VCSetGridSizeBP(short* iError, Point2D* dpSize_);
```

Visual Basic:
```
Declare Sub VCGetGridSizeBP Lib "VCMAIN32.DLL" (iError As Integer, pRet As Point2d)
Declare Sub VCSetGridSizeBP Lib "VCMAIN32.DLL" (iError As Integer, dpSize_ As Point2d)
```

Delphi:
```
procedure VCGetGridSizeBP(var iError: Integer; var pRet: Point2D); far;
procedure VCSetGridSizeBP(var iError: Integer; var dpSize_: Point2D); far;
```

Parameters *dpSize* - the Point2D structure containing the desired x and y scale

Notes The grid can have specified distances between horizontally and vertically placed grid points. The grid size or distance can be set to the desired value in both the X and Y direction with VCSetGridSize

See Also VCGetGridDisplay, VCGetGridOrigin, VCGetGridSnap

VCGetGridSnap
VCSetGridSnap

Version 1.2

Description A reference grid can be displayed and set as a snap to aid in the placing of entities.

Declaration

C/C++:
extern "C" BOOL WINAPI VCGetGridSnap(short* iError);
extern "C" void WINAPI VCSetGridSnap(short* iError, BOOL tfSnap);

Visual Basic:
Declare Function VCGetGridSnap Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetGridSnap Lib "VCMAIN32.DLL" (iError As Integer, ByVal tfSnap As Integer)

Delphi:
function VCGetGridSnap(var iError: Integer):Boolean; far;
procedure VCSetGridSnap(var iError: Integer; tfSnap: Boolean); far

Parameters
tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes The grid can have specified distances between horizontally and vertically placed grid points. When Snap Grid is ON, the cursor can move only from one grid point to another . The grid does not have to be visible for Snap Grid to be in effect.

See Also VCGetGridDisplay, VCGetGridOrigin, VCGetGridSize

VCGetHandlePt
VCSetHandlePt

Version 1.2

Description Option for displaying handle points.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetHandlePt(short* iError);
extern "C" void WINAPI VCSetHandlePt(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetHandlePt Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetHandlePt Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetHandlePt(var iError: Integer):Boolean; far;
procedure VCSetHandlePt(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes Each entity has handle points, which are essentially the endpoints. When working with entities it is sometimes convenient to display the entity handle points to aid in snapping. Turning off the display will reduce the visual clutter and increase the speed of redraws.

See Also VCGetConstPt

VCGetHatchColor
VCSetHatchColor

Version 1.2

Description The hatch color used when hatching a boundary.

Declaration

C/C++:
extern "C" short WINAPI VCGetHatchColor(short* iError);
extern "C" void WINAPI VCSetHatchColor(short* iError, short i);

Visual Basic:
Declare Function VCGetHatchColor Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetHatchColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)

Delphi:
function VCGetHatchColor(var iError: Integer):Integer; far;
procedure VCSetHatchColor(var iError: Integer; i: Integer); far;

Parameters *i* - the color setting.

See Also VCGetFillColor, VCGetHatchName, VCGetDrawHBoundary,VCGetHatchRot

{button ,AL(`Adding a Hatch/Fill Entity',0,`',`')} Task Guide Examples

VCGetHatchDisplay
VCSetHatchDisplay

Version 1.2

Description Determine whether hatch entities are displayed on the screen as well as in print and plot routines. Turning off the display will reduce the visual clutter and increase the speed of redraws.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetHatchDisplay(short* iError);
extern "C" void WINAPI VCSetHatchDisplay(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetHatchDisplay Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetHatchDisplay Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetHatchDisplay(var iError: Integer):Boolean; far;
procedure VCSetHatchDisplay(var iError: Integer; tf: Boolean); far; external'VCMAIN';
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetHatchName, VCGetDrawHBoundary,VCGetHatchRot

{button ,AL(`Adding a Hatch/Fill Entity',0,`',`')} Task Guide Examples

VCGetHatchName
VCSetHatchName

Version 1.2

Description The current hatch pattern name from the settings. VCGetHatchName retrieves the current hatch pattern setting. This call differs from VCGetSystemHatchName which returns the pattern name at the specified index.

Declaration

C/C++: extern "C" short WINAPI VCGetHatchName(short* iError, char* s);
extern "C" void WINAPI VCSetHatchName(short* iError, char* s);

Visual Basic: Declare Function VCGetHatchName Lib "VCMAIN32.DLL" (iError As Integer, ByVal s As String) As Integer
Declare Sub VCSetHatchName Lib "VCMAIN32.DLL" (iError As Integer, ByVal s As String)

Delphi: function VCGetHatchName(var iError: Integer; s: PChar):Integer; far;
procedure VCSetHatchName(var iError: Integer; s: PChar); far;

Parameters *s* - the string representing the current hatch pattern.

See Also VCGetSystemHatchName, VCGetDrawHBoundary,VCGetHatchRot

{button ,AL(`Adding a Hatch/Fill Entity',0,``',``')} Task Guide Examples

VCGetHatchRot
VCSetHatchRot

Version 1.2

Description The hatch rotation angle. As with all the angle settings, the angle value is in radians.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetHatchRot(short* iError);
extern "C" void WINAPI VCGetHatchRotBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetHatchRot(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetHatchRotBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetHatchRot Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetHatchRotBP(var iError: Integer; var dRet: Double); far;
procedure VCSetHatchRot(var iError: Integer; dRet: Double); far;
```

Parameters *d* - a double representing the hatch rotation angle in radians.

See Also VCGetHatchName, VCGetDrawHBoundary, VCGetHatchScale

{button ,AL(`Adding a Hatch/Fill Entity',0,`',`')} Task Guide Examples

VCGetHatchScale
VCSetHatchScale

Version 1.2

Description The multiplier used to scale the hatch definition to determine the size of the displayed hatch.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetHatchScale(short* iError);
extern "C" void WINAPI VCGetHatchScaleBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetHatchScale(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetHatchScaleBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetHatchScale Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetHatchScaleBP(var iError: Integer; var dRet: Double); far;
procedure VCSetHatchScale(var iError: Integer; dRet: Double); far;
```

Parameters *d* - a double representing the scale value.

See Also VCGetHatchName, VCGetDrawHBoundary,VCGetHatchRot

{button ,AL(`Adding a Hatch/Fill Entity',0,`',`')} Task Guide Examples

VCGetHighlight
VCSetHighlight

Version 1.2

Description Specifies if selected objects are highlighted in the selection color .

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetHighlight(short* iError);
extern "C" void WINAPI VCSetHighlight(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetHighlight Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetHighlight Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetHighlight(var iError: Integer):Boolean; far;
procedure VCSetHighlight(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

{button ,AL(`Parsing a Filtered Entity List;Parsing an On Screen List',0,`',`')} Task Guide Examples

VCGethWnd

Version	2.0
Description	Returns the Window handle for the given drawing.
Declaration	
C/C++	extern "C" long WINAPI VCGethWnd(WORLDHANDLE hW);
Visual Basic	Declare Function VCGethWnd Lib "VCMAIN32.DLL" (ByVal hW As Long) As Long
Delphi	function VCGethWnd(hW: Longint):Longint; far;
Parameters	hW - the WORLDHANDLE for the drawing. Return - the Window HWND for the window.
Notes	The API provides access to the Window handles for both the frame and the drawing. These handles can e used by other API to provide information to the window such as creating a child window from the Windows API. VCGethWnd returns the handle for individual MDI Windows based on the drawing index while VCGethWndFrame returns the handle for the entire Corel Visual CADD frame.
See Also	VCGetCurrWorld, VCGethWndFrame

VCGethWndFrame
VCSethWndFrame

Version 2.0

Description Returns the Window handle for the Corel Visual CADD frame.

Declaration

C/C++
```
extern "C" long WINAPI VCGethWndFrame();
extern "C" void WINAPI VCSethWndFrame(long hWnd);
```

Visual Basic
```
Declare Function VCGethWndFrame Lib "VCMAIN32.DLL" () As Long
Declare Sub VCSethWndFrame Lib "VCMAIN32.DLL" (ByVal hWnd As Long)
```

Delphi
```
function VCGethWndFrame:Longint; far;
procedure VCSethWndFrame(hWnd: Longint); far;
```

Parameters *Return* - the Window HWND for the window.

Notes The API provides access to the Window handles for both the frame and the drawing. These handles can e used by other API to provide information to the window such as creating a child window from the Windows API. VCGethWnd returns the handle for individual MDI Windows based on the drawing index while VCGethWndFrame returns the handle for the entire Corel Visual CADD frame.

See Also VCGetCurrWorld, VCGethWnd

VCGetIncSnap
VCSetIncSnap

Version 1.2

Description Specifies the increment snap option during ortho constrained operations..

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetIncSnap(short* iError);
extern "C" void WINAPI VCSetIncSnap(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetIncSnap Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetIncSnap Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetIncSnap(var iError: Integer):Integer; far;
procedure VCSetIncSnap(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes While in ortho mode there is the capability to have the cursor jump specific increments along the rubberband or drag line (ortho mode must be turned on for this call to work). This is called increment snap. This has an advantage over snap grid because it measures along the rubberband or drag line which will be at the ortho angle, not just along the horizontal and vertical

See Also VCGetIncSnapSize, VCGetOrthoMode,

VCGetIncSnapSize
VCSetIncSnapSize

Version 1.2

Description Specifies the distance between increment snaps during ortho constrained operations.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetIncSnapSize(short* iError);
extern "C" void WINAPI VCGetIncSnapSizeBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetIncSnapSize(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetIncSnapSizeBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetIncSnapSize Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetIncSnapSizeBP(var iError: Integer; var dRet: Double); far;
procedure VCSetIncSnapSize(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the distance between increment snaps.

Notes While in ortho mode there is the capability to have the cursor jump specific increments along the rubberband or drag line (ortho mode must be turned on for this API call to work). This is called increment snap. This has an advantage over snap grid because it measures along the rubberband or drag line which will be at the ortho angle not just along the horizontal and vertical.

See Also VCGetIncSnap, VCGetOrthoMode

VCGetInitCount

Version	1.2
Description	Returns the number of times that Corel Visual CADD has been initialized but not terminated.
Declaration	
C/C++:	extern "C" short WINAPI VCGetInitCount(void);
Visual Basic:	Declare Function VCGetInitCount Lib "VCMAIN32.DLL" () As Integer
Delphi:	function VCGetInitCount:Integer; far;
Parameters	*Returns* - the number of currently active instances of Corel Visual CADD.
Notes	When making tools or external applications that need Corel Visual CADD running, it is a good idea to check to see if any instances are currently available. VCGetInitCount will return the number of currently running instances of Corel Visual CADD, which a program can then use to decide if it should rely on one of the preexisting instances or spawn a new instance using VCInit.
See Also	VCInit, VCTerminate

{button ,AL(`Initialization Check',0,`',`')} Task Guide Examples

VCGetIsoMode
VCSetIsoMode

Version 2.0.1

Description Specifies the isometric grid mode.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetIsoMode(short* iError);
extern "C" void WINAPI VCSetIsoMode(short* iError, vbool tf);
```

Visual Basic
```
Declare Function VCGetIsoMode Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetIsoMode Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetIsoMode(var iError: Integer):Boolean; far;
procedure VCSetIsoMode(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes The IsoPlane will change the cursor to reflect the current plane and restrict the cursor movement to that 30/60/90 plane. This command is available only through the API.

See Also VCGetIsoPlane

VCGetIsoPlane
VCSetIsoPlane

Version 2.0.1

Description Specifies the isometric grid plane.

Declaration

C/C++
```
extern "C" short WINAPI VCGetIsoPlane(short* iError);
extern "C" void WINAPI VCSetIsoPlane(short* iError, short iPlane);
```

Visual Basic
```
Declare Function VCGetIsoPlane Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetIsoPlane Lib "VCMAIN32.DLL" (iError As Integer, ByVal iPlane As Integer)
```

Delphi
```
function VCGetIsoPlane(var iError: Integer):Integer; far;
procedure VCSetIsoPlane(var iError: Integer; iPlane: Integer); far;
```

Parameters iPlane - the plane for the isometric grid mode
0 - LEFT
1 - RIGHT
2 - TOP

Notes The IsoPlane will change the cursor to reflect the current plane and restrict the cursor movement to that 30/60/90 plane. This command is available only through the API.

See Also VCGetIsoMode

VCGetKeepAcadFontName
VCSetKeepAcadFontName

Version 1.2

Description When active, the current font mapping in the DWG Font tabs is overridden.

Declaration

C/C++: extern "C" BOOL WINAPI VCGetKeepAcadFontName(short* iError);
extern "C" void WINAPI VCSetKeepAcadFontName(short* iError, BOOL tf);

Visual Basic: Declare Function VCGetKeepAcadFontName Lib "VCMAIN32.DLL" (iErr As Integer) As Integer
extern "C" void WINAPI VCSetKeepAcadFontName(short* iError, BOOL tf);

Delphi: function VCGetKeepAcadFontName(var iError: Integer):Boolean; far;
procedure VCSetKeepAcadFontName(var iError: Integer; tf: Boolean); far;

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes All fonts are mapped to existing fonts of the same name. If a font of the same name does not exist, Corel Visual CADD will map the font to the default font.

See Also VCGetKeepGCDFontName, VCGetKeepGCDHatch, VCGetGCDDefaultHatchName, VCGetAcadImportUnit, VCSaveDrawing

VCGetKeepGCDFontName
VCSetKeepGCDFontName

Version 1.2

Description When active, the current font mapping in the GCD Font tabs is overridden.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetKeepGCDFontName(short* iError);
extern "C" void WINAPI VCSetKeepGCDFontName(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetKeepGCDFontName Lib "VCMAIN32.DLL" (iErr As Integer) As Integer
extern "C" void WINAPI VCSetKeepGCDFontName(short* iError, BOOL tf);
```

Delphi:
```
function VCGetKeepGCDFontName(var iError: Integer):Boolean; far;
procedure VCSetKeepGCDFontName(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes All fonts are mapped to existing fonts of the same name. If a font of the same name does not exist, Corel Visual CADD will map the font to the default font.

See Also VCGetKeepAcadFontName, VCGetKeepGCDHatch, VCGetGCDDefaultHatchName, VCGetAcadImportUnit

VCGetKeepGCDHatch
VCSetKeepGCDHatch

Version 1.2

Description Specifies if the Generic CAD hatch patterns are converted to symbols or to a default pattern.

Declaration

C/C++:
extern "C" BOOL WINAPI VCGetKeepGCDHatch(short* iError);
extern "C" void WINAPI VCSetKeepGCDHatch(short* iError, BOOL tf);

Visual Basic:
Declare Function VCGetKeepGCDHatch Lib "VCMAIN32.DLL" (iErr As Integer) As Integer
extern "C" void WINAPI VCSetKeepGCDHatch(short* iError, BOOL tf);

Delphi:
function VCGetKeepGCDHatch(var iError: Integer):Boolean; far;
procedure VCSetKeepGCDHatch(var iError: Integer; tf: Boolean); far;

Parameters *tf* - toggle setting
0 - use a default hatch pattern.
1- convert to a symbol definition.

Notes The hatch patterns from the GCD format can not be used directly in Corel Visual CADD. These patterns must be converted to either symbols or to a default hatch pattern when importing drawings into Corel Visual CADD. VCGetKeepGCDHatch specifies whether hatches are converted to a default hatch pattern or recreated as a symbol definition.

See Also VCGetKeepGCDFontName, VCGetKeepAcadFontName, VCGetGCDDefaultHatchName, VCGetAcadImportUnit

VCGetLastPoint
VCSetLastPoint

Version 1.2.1

Description Returns the last point used in the construction of any entity.

Declaration

C/C++:
```
extern "C" void FAR WINAPI VCGetLastPoint(short* iError, Point2D* dpP);
extern "C" void WINAPI VCSetLastPoint(short* iError, Point2D* dpP);
```

Visual Basic:
```
Declare Sub VCGetLastPoint Lib "VCMAIN32.DLL" ( iError As Integer, dpP As Point2D)
Declare Sub VCSetLastPoint Lib "VCMAIN32.DLL" (iErr As Integer, dpP As Point2D)
```

Delphi:
```
procedure VCGetLastPoint(var iError: Integer; var dpP: Point2D); far;
procedure VCSetLastPoint(var iError: Integer; var dpP: Point2D); far;
```

Parameters *dpP* - set to contain the coordinates of the last point selected.

Notes Similar to the Last Point snap and can be used from the API when constructing geometry, to reference the last mouse down point placed in the drawing.

See Also VCGetCurrentEntityPoint, VCGetCurrentPoint

VCGetLayerDisplay
VCSetLayerDisplay

Version 1.2

Description Drawing layers can be turned "off" to eliminate the layer from the screen.

Declaration

C/C++:
extern "C" BOOL WINAPI VCGetLayerDisplay(short* iError, short iIndex);
extern "C" void WINAPI VCSetLayerDisplay(short* iError, short iIndex, BOOL tf);

Visual Basic:
Declare Function VCGetLayerDisplay Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer) As Integer
Declare Sub VCSetLayerDisplay Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal tf As Integer)

Delphi:
function VCGetLayerDisplay(var iError: Integer; iIndex: Integer):Boolean; far;
procedure VCSetLayerDisplay(var iError: Integer; iIndex: Integer; tf: Boolean); far;

Parameters
iIndex - the layer index
tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes The entity data is still stored in the drawing and can be turned "on" to re-show the entities. By hiding layers, redraw times can be improved for only the desired details or layers.

See Also VCGetLayerRedraw, VCGetLayerDisplay, VCGetLayerIndex

VCGetLayerHasData

Version 1.2

Description Determines whether the specified layer contains any drawing data.

Declaration

C/C++: extern "C" vbool WINAPI VCGetLayerHasData(short* iError, short iIndex);

Visual Basic: Declare Function VCGetLayerHasData Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer) As Integer

Delphi: function VCGetLayerHasData(var iError: Integer; iIndex: Integer):Boolean; far;

Parameters *iIndex* - the layer number in question.
Returns - whether the layer contains data
0 - no data on layer.
1 - data on layer.

Notes When displaying layer information to the user, an application may need to display information about what is on the layer. This subroutine provides a mechanism for determining if anything is on a particular layer. This can also be useful in situations where a short layer list is required as in the Corel Visual CADD layer manager.

See Also VCGetLayerRedraw, VCGetLayerDisplay, VCGetLayerIndex

VCGetLayerIndex
VCSetLayerIndex

Version 1.2

Description Specifies the current layer property for all subsequent primary entity placements.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetLayerIndex(short* iError);
extern "C" void WINAPI VCSetLayerIndex(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetLayerIndex Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLayerIndex Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetLayerIndex(var iError: Integer):Integer; far;
procedure VCSetLayerIndex(var iError: Integer; i: Integer); far;
```

Parameters *i* - the desired layer index.

Notes All primary drawing entities have four specific properties associated with them. These are: color, layer, line type and line width. Each of these are set by an index, not by name. Text and Dimensions each have their own properties and as such are not set or retrieved using these functions but instead have their own similar functions.

See Also VCGetLineTypeIndex, VCGetLineWidthIndex, VCGetColorIndex, VCGetTextLayer, VCGetDimLayer

{button ,AL(`Adding a Single Entity',0,`',`')} Task Guide Examples

VCGetLayerIndexFromName

Version 1.2

Description Given a layer name, returns the index number for the named layer.

Declaration

C/C++: extern "C" short WINAPI VCGetLayerIndexFromName(short* iError, char* pName);

Visual Basic: Declare Function VCGetLayerIndexFromName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pName As String) As Integer

Delphi: function VCGetLayerIndexFromName(var iError: Integer; pName: PChar):Integer;

Parameters *pName* - the layer name of an existing layer.
Returns an integer from 0 to 511 representing the layer index number.

Notes In Corel Visual CADD, it is possible to name any of the 1024 supported layers. If an application requires a unknown layer number to be named a specific name, such as "electrical" this allows the application to locate that layer and retrieve the index of that layer for use in other functions requiring a layer number.

See Also VCGetLayerNameFromIndex, VCGetLayerIndex

VCGetLayerProperties
VCSetLayerProperties

Version 2.0.1

Description Specifies the layer properties for the given layer.

Declaration

C/C++

```
extern "C" vbool WINAPI VCGetLayerProperties(short* iError, short iLayer, short* iColor, short* iLtype, short* iWidth, float* fWidth);
extern "C" vbool WINAPI VCSetLayerProperties(short* iError, short iLayer, short iColor, short iLtype, short iWidth, float fWidth);
```

Visual Basic

```
Declare Function VCGetLayerProperties Lib "VCMAIN32.DLL" (iError As Integer, ByVal iLayer As Integer, iColor As Integer, iLtype As Integer, iWidth As Integer, fWidth As Double) As Integer
Declare Function VCSetLayerProperties Lib "VCMAIN32.DLL" (iError As Integer, ByVal iLayer As Integer, ByVal iColor As Integer, ByVal iLtype As Integer, ByVal iWidth As Integer, ByVal fWidth As Double) As Integer
```

Delphi

```
function VCGetLayerProperties(var iError: Integer; iLayer: Integer; var iColor: Integer; var iLtype: Integer; var iWidth: Integer; var fWidth: Double):Boolean; far;
function VCSetLayerProperties(var iError: Integer; iLayer: Integer; iColor:Integer; iLtype: Integer; iWidth: Integer; fWidth: Double):Boolean; far;
```

Parameters

iLayer - the index for the layer.
iColor - the color property assigned to the layer.
i*LType* - the line type assigned to the layer
iWidth - the line width index assigned to the layer.
f*Width* - the real world line width for the layer.
returns - the success of the function.
0 - FAILED
1 - PASSED

Notes Layer properties were introduced into v2.0.1 allowing properties to be assigned by layer rather than by entity. For example, a layer can be set so all entities drawn on the layer will be a specific color, line type and line width. This will override the current properties settings when active. VCGetUseByLayerProperties is used to determine if the layer has active property settings while VCSetUseByLayerProperties allows an application to choose which properties to use. VCSetLayerProperties will set the values for the layer and VCClearLayerProperties turns the capability off and clears all associated values. It is important to keep track of the state of layer properties when modifying entities in the drawing. For example, if you set the color index using VCSetColorIndex but the layer properties are enabled the proper color may not get applied. Therefore when attempting to control the properties of entities as they are placed it is imperative that the application monitor the setting for by layer control as the information is being supplied by the API.

See Also VCLayerHasProperties

VCGetLayerNameFromIndex

Version 1.2

Description Given a layer index number, retrieves the name associated with that layer.

Declaration

C/C++: extern "C" short WINAPI VCGetLayerNameFromIndex(short* iError, short iIndex, char* pName);

Visual Basic: Declare Sub VCGetLayerNameFromIndex Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal pName As String)

Delphi: function VCGetLayerNameFromIndex(var iError: Integer; iIndex: Integer; pName PChar):Integer; far;

Parameters *iIndex* - the number of the layer.
pName - set by the procedure as the name of the layer.

Notes Whenever displaying layers for user selection, it is important to display all named layers by their name as people recognize and will look for the named layers. VCGetLayerNameFromIndex provides this while allowing the internal code to still use the layer indices.

See Also VCGetLayerIndex, VCGetLayerIndexFromName

VCGetLayerRedraw
VCSetLayerRedraw

Version 1.2

Description Turning Redraw off causes Corel Visual CADD to wait for the Layer Manager Dialog box to be closed before it will hide or display the chosen layers.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetLayerRedraw(short* iError);
extern "C" void WINAPI VCSetLayerRedraw(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetLayerRedraw Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLayerRedraw Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetLayerRedraw(var iError: Integer):Boolean; far;
procedure VCSetLayerRedraw(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes Layer redraw provides visual feedback to the user as layers are displayed or hidden but can time consuming to wait for layer redraws before picking new layers.

See Also VCGetLayerRedraw, VCGetLayerDisplay, VCGetLayerIndex

VCGetLeaderArrowAngle
VCSetLeaderArrowAngle

Version 2.0

Description The dimension angle setting is used by all dimension arrow types except circular. As with all angular settings in Corel Visual CADD the value should be expressed in radians.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetLeaderArrowAngle(short* iError);
extern "C" void WINAPI VCGetLeaderArrowAngleBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetLeaderArrowAngle(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetLeaderArrowAngleBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetLeaderArrowAngle Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetLeaderArrowAngleBP(var iError: Integer; var dRet: Double); far;
procedure VCSetLeaderArrowAngle(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - double value representing the angle setting in radians

See Also VCGetLeaderArrowLength, VCGetLeaderArrowMode, VCGetLeaderArrowType

VCGetLeaderArrowLength
VCSetLeaderArrowLength

Version 2.0

Description Several settings are available for dimension arrows. These need to be set prior to placing the dimension into the drawing. The arrow length is analogous to the arrow size or scale.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetLeaderArrowLength(short* iError);
extern "C" void WINAPI VCGetLeaderArrowLengthBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetLeaderArrowLength(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetLeaderArrowLengthBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetLeaderArrowLength Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetLeaderArrowLengthBP(var iError: Integer; var dRet: Double); far;
procedure VCSetLeaderArrowLength(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the dimension arrow length

See Also VCGetLeaderArrowAngle, VCGetLeaderArrowMode, VCGetLeaderArrowType

VCGetLeaderArrowMode
VCSetLeaderArrowMode

Version 2.0

Description Several settings are available for dimension arrows. These need to be set prior to placing the dimension into the drawing The arrow mode determines if the arrows are flipped to the outside or the inside of the extension lines.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetLeaderArrowMode(short* iError);
extern "C" void WINAPI VCSetLeaderArrowMode(short* iError, short b);
```

Visual Basic:
```
Declare Function VCGetLeaderArrowMode Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLeaderArrowMode Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetLeaderArrowMode(var iError: Integer):Integer; far;
procedure VCSetLeaderArrowMode(var iError: Integer; b: Integer); far;
```

Parameters *b* - the state of the arrow flip.
0 - do not flip the dimension arrows to the outside of the dimension.
1 - flip the dimension arrows to the outside of the dimension.

See Also VCGetLeaderArrowLength, VCGetLeaderArrowAngle, VCGetLeaderArrowType

VCGetLeaderArrowType
VCSetLeaderArrowType

Version 2.0

Description Corel Visual CADD allows several options for the dimension arrow type setting.

Declaration

C/C++
```
extern "C" short WINAPI VCGetLeaderArrowType(short* iError);
extern "C" void WINAPI VCSetLeaderArrowType(short* iError, short b);
```

Visual Basic
```
Declare Function VCGetLeaderArrowType Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Declare Sub VCSetLeaderArrowType Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi
```
function VCGetLeaderArrowType(var iError: Integer):Integer; far;
procedure VCSetLeaderArrowType(var iError: Integer; b: Integer); far;
```

Parameters b - the value of the arrow type.
0 - DIMARROWREGNOFILL
1 - DIMARROWREGFILLED
2 - DIMARROWREGOPEN
3 - DIMARROWNOTCHED
4 - DIMARROWSLASH
5 - DIMARROWCIRCLENOFILL
6 - DIMARROWCIRCLEFILL

Notes Corel Visual CADD allows different dimension and leader sections to be edited. VCGetLeaderArrowType changes the arrow type that is used on the leader.

See Also VCGetLeaderArrowAngle, VCGetLeaderArrowLength, VCGetLeaderArrowMode, VCGetLeaderArrowType, VCGetLeaderShoulderLength

VCGetLeaderFontName
VCSetLeaderFontName

Version 2.0

Description The name of the font to be used for all for all subsequent text placements.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetLeaderTextFontName(short* iError, char* pS);
extern "C" void WINAPI VCSetLeaderTextFontName(short* iError, char* sz);
```

Visual Basic:
```
Declare Function VCGetLeaderTextFontName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String) As Integer
Declare Sub VCSetLeaderTextFontName Lib "VCMAIN32.DLL" (iError As Integer, ByVal sz As String)
```

Delphi:
```
function VCGetLeaderTextFontName(var iError: Integer; pS: PChar):Integer; far;
procedure VCSetLeaderTextFontName(var iError: Integer; sz: PChar); far
```

Parameters *pS* - the name of the current font.

See Also CGetLeaderFontName, VCGetLeaderString, VCGetLeaderTextAspect, VCGetLeaderTextBold, VCGetLeaderTextCharSpace, VCGetLeaderTextFillVText, VCGetLeadertTextHeight, VCGetLeaderTextItalic, VCGetLeaderTextItalicAng, VCGetLeaderTextLineSpace, VCGetLeaderTextProSpacing, VCGetLeaderTextUnderline, VCGetLeaderTextOffset

VCGetLeaderShoulderLength
VCSetLeaderShoulderLength

Version 1.2

Description The shoulder length of the leader specifies the length of the segment from the last placed leader point to the leader text.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetLeaderShoulderLength(short* iError);
extern "C" void WINAPI VCGetLeaderShoulderLengthBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetLeaderShoulderLength(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetLeaderShoulderLengthBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetLeaderShoulderLength Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetLeaderShoulderLengthBP(var iError: Integer; var dRet: Double); far;
procedure VCSetLeaderShoulderLength(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the shoulder length.

See Also VCGetLeaderString, VCGetLeaderTextOffset

VCGetLeaderString
VCSetLeaderString

Version 1.2

Description The leader string value for the current leader entity.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetLeaderString(short* iError, char* s);
extern "C" void WINAPI VCSetLeaderString(short* iError, char* s);
```

Visual Basic:
```
Declare Function VCGetLeaderString Lib "VCMAIN32.DLL" (iError As Integer, ByVal s As String) As Integer
Declare Sub VCSetLeaderString Lib "VCMAIN32.DLL" (iError As Integer, ByVal s As String)
```

Delphi:
```
function VCGetLeaderString(var iError: Integer; s: PChar):Integer; far;
procedure VCSetLeaderString(var iError: Integer; s: PChar); far;
```

Parameters *s* - the text string passed to the leader entity.

See Also CGetLeaderFontName, VCGetLeaderString, VCGetLeaderTextAspect, VCGetLeaderTextBold, VCGetLeaderTextCharSpace, VCGetLeaderTextFillVText, VCGetLeadertTextHeight, VCGetLeaderTextItalic, VCGetLeaderTextItalicAng, VCGetLeaderTextLineSpace, VCGetLeaderTextProSpacing, VCGetLeaderTextUnderline, VCGetLeaderTextOffset

VCGetLeaderTextAspect
VCSetLeaderTextAspect

Version 2.0

Description Specifies the current text aspect ratio setting. The text aspect ratio is the proportion of the text height to the text width.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetLeaderTextAspect(short* iError);
extern "C" void WINAPI VCGetLeaderTextAspectBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetLeaderTextAspect(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetLeaderTextAspectBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetLeaderTextAspect Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetLeaderTextAspectBP(var iError: Integer; var dRet: Double); far;
procedure VCSetLeaderTextAspect(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the current text aspect ratio.

See Also CGetLeaderFontName, VCGetLeaderString, VCGetLeaderTextAspect, VCGetLeaderTextBold, VCGetLeaderTextCharSpace, VCGetLeaderTextFillVText, VCGetLeadertTextHeight, VCGetLeaderTextItalic, VCGetLeaderTextItalicAng, VCGetLeaderTextLineSpace, VCGetLeaderTextProSpacing, VCGetLeaderTextUnderline, VCGetLeaderTextOffset

VCGetLeaderTextBold
VCSetLeaderTextBold

Version 2.0

Description Specifies the bold display option for TT Fonts with the leader command.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetLeaderTextBold(short* iError);
extern "C" void WINAPI VCSetLeaderTextBold(short* iErrors, short i);
```

Visual Basic
```
Declare Function VCGetLeaderTextBold Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLeaderTextBold Lib "VCMAIN32.DLL" (iErrors As Integer, ByVal i As Integer)
```

Delphi
```
function VCGetLeaderTextBold(var iError: Integer):Boolean; far;
procedure VCSetLeaderTextBold(var iErrors: Integer; i: Integer); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also CGetLeaderFontName, VCGetLeaderString, VCGetLeaderTextAspect, VCGetLeaderTextBold, VCGetLeaderTextCharSpace, VCGetLeaderTextFillVText, VCGetLeadertTextHeight, VCGetLeaderTextItalic, VCGetLeaderTextItalicAng, VCGetLeaderTextLineSpace, VCGetLeaderTextProSpacing, VCGetLeaderTextUnderline, VCGetLeaderTextOffset

VCGetLeaderTextCharSpace
VCSetLeaderTextCharSpace

Version 2.0

Description Character spacing is the amount of space that appears between characters in a text string. It determines if the characters in a word are crowded or spread out. The value is a percentage of the characters height and applies only to vector fonts.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetLeaderTextCharSpace(short* iError);
extern "C" void WINAPI VCGetLeaderTextCharSpaceBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetLeaderTextCharSpace(short* iError, double dCharSpacing);
```

Visual Basic:
```
Declare Sub VCGetLeaderTextCharSpaceBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetLeaderTextCharSpace Lib "VCMAIN32.DLL" (iError As Integer, ByVal dCharSpacing As Double)
```

Delphi:
```
procedure VCGetLeaderTextCharSpaceBP(var iError: Integer; var dRet: Double); far;
procedure VCSetLeaderTextCharSpace(var iError: Integer; dCharSpacing: Double); far;
```

Parameters *dCharSpacing* - the charcter spacing as a decimal percentage (i.e. 1.5 is 150%)

See Also VCGetLeaderTextAspect, VCGetLeaderTextBold, VCGetLeaderTextFontName, VCGetLeaderTextHeight, VCGetLeaderTextItalic, VCGetLeaderTextItalicValue, VCGetLeaderTextJustify, VCGetLeaderTextLineSpace, VCGetLeaderTextProSpacing, VCGetLeaderTextRot, VCGetLeaderString, VCGetLeaderTextUnderline

VCGetLeaderTextFillVText
VCSetLeaderTextFillVText

Version 2.0

Description Specifies if vector fonts are filled in dimensions.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetDimLeaderTextFillVLeaderText(short* iError);
extern "C" void WINAPI VCSetDimLeaderTextFillVLeaderText(short* iError, vbool tf);
```

Visual Basic
```
Declare Function VCGetDimLeaderTextFillVLeaderText Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetDimLeaderTextFillVLeaderText Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetDimLeaderTextFillVLeaderText(var iError: Integer):Boolean; far;
procedure VCSetDimLeaderTextFillVLeaderText(var iError: Integer; tf: Boolean); far;
```

Parameters

Notes Depending on what type of font is being used and how the font is defined, then you might be able to modify its appearance. Corel Visual CADD utilizes both TrueType Fonts and built in vector fonts. The vector fonts can be converted from other font formats such as .SHX and .FNT. When working with text entities it is important to understand the type of font being used. Certain settings such as Bold, Italic and Underline only effect TrueType Fonts while others such as Italic value are designed for vector fonts. VCGetDimTextFillVText will fill vector fonts that are closed outline fonts. Therefore, when altering the settings of an existing text entity it is necessary to determine the type of font in order to apply the appropriate settings. VCIsFontNameVText determines if the specified font is a Corel Visual CADD vector font.

See Also VCIsFontNameVLeaderText, VCGetDimFont

VCGetLeaderTextHeight
VCSetLeaderTextHeight

Version 2.0

Description Unlike most other Windows programs, Corel Visual CADD measures text height in real world units, specifically inches, instead of points.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetLeaderTextHeight(short* iError);
extern "C" void WINAPI VCGetLeaderTextHeightBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetLeaderTextHeight(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetLeaderTextHeightBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetLeaderTextHeight Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetLeaderTextHeightBP(var iError: Integer; var dRet: Double); far;
procedure VCSetLeaderTextHeight(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the text height.

See Also VCGetDimTextAspect, VCGetDimTextBold, VCGetDimTextHeight, VCGetDimTextItalic, VCGetDimTextItalicValue, VCGetDimTextLineSpace, VCGetDimTextProSpacing, VCGetDimTextRotationType, VCGetDimTextUnderline

VCGetLeaderTextItalic
VCSetLeaderTextItalic

Version 2.0

Description Specifies the italic display option for TT Fonts with the leader command.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetLeaderTextItalic(short* iError);
extern "C" void WINAPI VCSetLeaderTextFillVText(short* iError, vbool tf);
```

Visual Basic
```
Declare Function VCGetLeaderTextItalic Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLeaderTextItalic Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetLeaderTextItalic(var iError: Integer):Boolean; far;
procedure VCSetLeaderTextFillVText(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes

See Also CGetLeaderFontName, VCGetLeaderString, VCGetLeaderTextAspect, VCGetLeaderTextBold, VCGetLeaderTextCharSpace, VCGetLeaderTextFillVText, VCGetLeadertTextHeight, VCGetLeaderTextItalic, VCGetLeaderTextItalicAng, VCGetLeaderTextLineSpace, VCGetLeaderTextProSpacing, VCGetLeaderTextUnderline, VCGetLeaderTextOffset

VCGetLeaderTextItalicAng
VCSetLeaderTextItalicAng

Version 2.0

C/C++: extern "C" double WINAPI VCGetLeaderTextItalicAng(short* iError, double* dI);
extern "C" void WINAPI VCGetLeaderTextItalicAngBP(short* iError, double* dI);
extern "C" void WINAPI VCSetLeaderTextItalicAng(short* iError, double dI);

Visual Basic: Declare Sub VCGetLeaderTextItalicAngBP Lib "VCMAIN32.DLL" (iError As Integer, dI As Double)
Declare Sub VCSetLeaderTextItalicAng Lib "VCMAIN32.DLL" (iError As Integer, ByVal dI As Double)

Delphi: procedure VCGetLeaderTextItalicAngBP(var iError: Integer; var dI: Double); far;
procedure VCSetLeaderTextItalicAng(var iError: Integer; dI: Double); far;

Parameters *dI* - the angle in radians for the slant

Notes The number must range between 45 and -45 degrees. As with all angle functions, the angle is specified in radians. A negative number will slant the text backwards.

See Also VCGetDimTextAspect, VCGetDimTextBold, VCGetDimTextHeight, VCGetDimTextItalic, VCGetDimTextItalicValue, VCGetDimTextLineSpace, VCGetDimTextProSpacing, VCGetDimTextRotationType, VCGetDimTextUnderline

VCGetLeaderTextLineSpace
VCSetLeaderTextLineSpace

Version 2.0

Description The between text line VCGetDimTextLineSpacespacing as a percentage of current text height.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetLeaderTextLineSpace(short* iError);
extern "C" void WINAPI VCGetLeaderTextLineSpaceBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetLeaderTextLineSpace(short* iError, double dLineSpacing);
```

Visual Basic:
```
Declare Sub VCGetLeaderTextLineSpaceBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetLeaderTextLineSpace Lib "VCMAIN32.DLL" (iError As Integer, ByVal dLineSpacing As Double)
```

Delphi:
```
procedure VCGetLeaderTextLineSpaceBP(var iError: Integer; var dRet: Double); far;
procedure VCSetLeaderTextLineSpace(var iError: Integer; dLineSpacing: Double); far;
```

Parameters *dRet* - spacing between the lines.

See Also VCGetDimTextAspect, VCGetDimTextBold, VCGetDimTextHeight, VCGetDimTextItalic, VCGetDimTextItalicValue, VCGetDimTextLineSpace, VCGetDimTextProSpacing, VCGetDimTextRotationType, VCGetDimTextUnderline

VCGetLeaderTextProSpacing
VCSetLeaderTextProSpacing

Version 2.0

Description Vector text character spacing can be forced to monospace or proportional spacing. Monospace is a characteristic of typewriter output and all characters will use the same amount of space regardless of their width and height.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetLeaderTextProSpacing(short* iError);
extern "C" void WINAPI VCSetLeaderTextProSpacing(short* iError, vbool b);
```

Visual Basic
```
Declare Function VCGetLeaderTextProSpacing Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLeaderTextProSpacing Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi
```
function VCGetLeaderTextProSpacing(var iError: Integer):Boolean; far;
procedure VCSetLeaderTextProSpacing(var iError: Integer; b: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also CGetLeaderFontName, VCGetLeaderString, VCGetLeaderTextAspect, VCGetLeaderTextBold, VCGetLeaderTextCharSpace, VCGetLeaderTextFillVText, VCGetLeadertTextHeight, VCGetLeaderTextItalic, VCGetLeaderTextItalicAng, VCGetLeaderTextLineSpace, VCGetLeaderTextProSpacing, VCGetLeaderTextUnderline, VCGetLeaderTextOffset

VCGetLeaderTextUnderline
VCSetLeaderTextUnderline

Version 2.0

Description Specifies the underline display option for TT Fonts with the leader command.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetLeaderTextUnderline(short* iError);
extern "C" void WINAPI VCSetLeaderTextUnderline(short* iError, vbool tf);
```

Visual Basic
```
Declare Function VCGetLeaderTextUnderline Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLeaderTextUnderline Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetLeaderTextUnderline(var iError: Integer):Boolean; far;
procedure VCSetLeaderTextUnderline(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also CGetLeaderFontName, VCGetLeaderString, VCGetLeaderTextAspect, VCGetLeaderTextBold, VCGetLeaderTextCharSpace, VCGetLeaderTextFillVText, VCGetLeadertTextHeight, VCGetLeaderTextItalic, VCGetLeaderTextItalicAng, VCGetLeaderTextLineSpace, VCGetLeaderTextProSpacing, VCGetLeaderTextUnderline, VCGetLeaderTextOffset

VCGetLeaderTextOffset
VCSetLeaderTextOffset

Version 1.2

Description The distance leader text is offset from the leader shoulder.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetLeaderTextOffset(short* iError);
extern "C" void WINAPI VCSetLeaderTextOffset(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetLeaderTextOffsetBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetLeaderTextOffset Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetLeaderTextOffsetBP(var iError: Integer; var dRet: Double); far;
procedure VCSetLeaderTextOffset(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the offset distance.

See Also CGetLeaderFontName, VCGetLeaderString, VCGetLeaderTextAspect, VCGetLeaderTextBold, VCGetLeaderTextCharSpace, VCGetLeaderTextFillVText, VCGetLeadertTextHeight, VCGetLeaderTextItalic, VCGetLeaderTextItalicAng, VCGetLeaderTextLineSpace, VCGetLeaderTextProSpacing, VCGetLeaderTextUnderline, VCGetLeaderTextOffset

VCGetLineTypeDisplay
VCSetLineTypeDisplay

Version 1.2

Description Determine whether line types are displayed on the screen or if the entities are shown as solid lines. Turning off the display will reduce the visual clutter and increase the speed of redraws.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetLineTypeDisplay(short* iError);
extern "C" void WINAPI VCSetLineTypeDisplay(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetLineTypeDisplay Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLineTypeDisplay Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetLineTypeDisplay(var iError: Integer):Boolean; far;
procedure VCSetLineTypeDisplay(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetLineWidthDisplay, VCGetHandlePt, VCGetConstPt

VCGetLineTypeFromIndex

Version 1.2

Description Returns a line type definition array containing all segment lengths and its size.

Declaration

C/C++: extern "C" void WINAPI VCGetLineTypeFromIndex(short* iError, short iIndex, char* pName, short* bCode, short* iDashCount, double* pDashes);

Visual Basic: Declare Sub VCGetLineTypeFromIndex Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal pName As String, bCode As Integer, iDashCount As Integer, pDashes As Double)

Delphi: procedure VCGetLineTypeFromIndex(var iError: Integer; iIndex: Integer; pName PChar; var bCode: Integer; var iDashCount: Integer; var pDashes: Double);

Parameters *iIndex* - the line type number.
pName - assigned by the procedure as the line type name.
bCode determines whether the line is a world scale or device scale.
1 - world scale.
2 - device scale.
iDashCount - the number of dashes used and is the size of the pDashes array.
pDashes points to and array of doubles representing each dash length.

Notes Corel Visual CADD line types use either a world scale or a device scale. Device line types will always appear with the appropriate lengths regardless of the drawing view on screen or the print size. World scale line types will always be displayed and printed to scale, that is a 1" dash printed at ¼ scale will be ¼" long on paper. The pDashes array must contain dash lengths for the line type in order they are to be drawn in the line. A positive value indicates a displayed (or on) dash length while a negative value indicates a non-displayed (or off) dash length. These non-displayed dash lengths can be thought of as an offset length from the end of the last dash length to the beginning of the next dash length.

See Also VCAddLineType, VCGetLineTypeIndexFromName, VCGetLineTypeNameFromIndex

{button ,AL(`Adding a Single Entity',0,`',`')} Task Guide Examples

VCGetLineTypeIndex
VCSetLineTypeIndex

Version 1.2

Description Specifies the current line type property for all subsequent primary entity placements..

Declaration

C/C++:
```
extern "C" short WINAPI VCGetLineTypeIndex(short* iError);
extern "C" void WINAPI VCSetLineTypeIndex(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetLineTypeIndex Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLineTypeIndex Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetLineTypeIndex(var iError: Integer):Integer; far;
procedure VCSetLineTypeIndex(var iError: Integer; i: Integer); far;
```

Parameters *i* - the desired line type index.

Notes All primary drawing entities have four specific properties associated with them. These are: color, layer, line type and line width. Each of these are set by an index, not by name. Text and Dimensions each have their own properties and as such are not set or retrieved using these functions but instead have their own similar functions

See Also VCGetLineWidthIndex, VCGetLayerIndex, VCGetColorIndex

VCGetLineTypeIndexFromName

Version 1.2

Description Given a line type name, returns the index number for the named line type.

Declaration

C/C++: extern "C" short WINAPI VCGetLineTypeIndexFromName(short* iError, char* pName);

Visual Basic: Declare Function VCGetLineTypeIndexFromName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pName As String) As Integer

Delphi: function VCGetLineTypeIndexFromName(var iError: Integer; pName PChar):Integer; far;

Parameters *pName* - the line type name of an existing line type.
Returns an integer from 0 to 255 representing the line type index number.

Notes Using the LINETYPE.DEF text file present in the Corel Visual CADD directory, it is possible for the user to create and assign any line type definition to any line type index number. An application can also do the same thing by using VCAddLineType. Once a line type has been assigned, that line type number takes on the defining line type name. Using VCGetLineTypeIndexFromName, any application can retrieve the index number for use with other functions requiring a line type index.

See Also VCAddLineType, VCGetLineTypeFromIndex, VCGetLineTypeNameFromIndex

{button ,AL(`Adding a Single Entity',0,`',`')} Task Guide Examples

VCGetLineTypeNameFromIndex

Version 1.2

Description Given a line type index number, VCGetLineTypeNameFromIndex will return the name associated with that line type.

Declaration

C/C++: extern "C" short WINAPI VCGetLineTypeNameFromIndex(short* iError, short iIndex, char* pName);

Visual Basic: Declare Function VCGetLineTypeNameFromIndex Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal pName As String) As Integer

Delphi: function VCGetLineTypeNameFromIndex(var iError: Integer; iIndex: Integer;

Parameters *iIndex* - the line type number.
pName - assigned by the procedure as the line type name.

Notes Since line types are customizable by the user, there is often a significance to the names given to the lines. All line type names should be used whenever the user is given an option to choose line types.

See Also VCAddLIneType, VCGetLineTypeIndexFromName, VCGetLineTypeFromIndex

{button ,AL(`Adding a Single Entity',0,`',`')} Task Guide Examples

VCGetLineWidthDisplay
VCSetLineWidthDisplay

Version 1.2

Description Determine whether line widths are displayed on the screen or if the entities are shown as line width 0. Turning off the display will reduce the visual clutter and increase the speed of redraws.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetLineWidthDisplay(short* iError);
extern "C" void WINAPI VCSetLineWidthDisplay(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetLineWidthDisplay Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLineWidthDisplay Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetLineWidthDisplay(var iError: Integer):Boolean; far;
procedure VCSetLineWidthDisplay(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetLineTypeDisplay, VCGetHandlePt, VCGetConstPt, VCGetLineTypeNameFromIndex

VCGetLineWidthIndex
VCSetLineWidthIndex

Version 1.2

Description Specifies the current line width property for all subsequent primary entity placements.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetLineWidthIndex(short* iError);
extern "C" void WINAPI VCSetLineWidthIndex(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetLineWidthIndex Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetLineWidthIndex Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetLineWidthIndex(var iError: Integer):Integer; far;
procedure VCSetLineWidthIndex(var iError: Integer; i: Integer); far;
```

Parameters *i* - the desired line width index.

Notes All primary drawing entities have four specific properties associated with them. These are: color, layer, line type and line width. Each of these are set by an index, not by name. Text and Dimensions each have their own properties and as such are not set or retrieved using these functions but instead have their own similar functions.

See Also VCGetLineTypeIndex, VCGetLayerIndex, VCGetColorIndex

{button ,AL(`Adding a Continuous Entity;Adding a Single Entity',0,`',`')} Task Guide Examples

VCGetLineWidthValue
VCSetLineWidthValue

Version 2.0

Description Specifies the line width value for real world line weights.

Declaration

C/C++
```
extern "C" void WINAPI VCGetLineWidthValue(short* iError, float* dV);
extern "C" void WINAPI VCSetLineWidthValue(short* iError, float dV);
```

Visual Basic
```
Declare Sub VCGetLineWidthValue Lib "VCMAIN32.DLL" (iError As Integer, dV As Double)
Declare Sub VCSetLineWidthValue Lib "VCMAIN32.DLL" (iError As Integer, ByVal dV As Double)
```

Delphi
```
procedure VCGetLineWidthValue(var iError: Integer; var dV: Double); far;
procedure VCSetLineWidthValue(var iError: Integer; dV: Double); far;
```

Parameters *dV* - the value for the real world line width defintion.

Notes Corel Visual CADD provides a set of 16 predefined line widths based on screen units. Line widths can also be set in real world coordinates to reflect exact line weights for output. The real world line widths print and display at the width entered regardless of the scale.

See Also VCGetLineWidthIndex, VCGetLineTypeIndex, VCGetLayerIndex, VCGetColorIndex

VCGetLTScaleDevice
VCSetLTScaleDevice

Version 1.2

Description Specifies the line scale reference frame.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetLTScaleDevice(short* iError);
extern "C" void WINAPI VCSetLTScaleDevice(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetLTScaleDeviceBP Lib "VCMAIN32.DLL" (iError As Integer, pRet As Double)
Declare Sub VCSetLTScaleDevice Lib "VCMAIN32.DLL" (iError As Integer, ByVal dRet As Double)
```

Delphi:
```
procedure VCGetLTScaleDeviceBP(var iError: Integer; var pRet: Double); far;
procedure VCSetLTScaleDevice(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the scaling factor to apply

Notes There are two linetype reference frames for measuring the lengths of the solid and blank segments that make up a custom line type called World and Device. If the World option is chosen, then the segment lengths are measured in the same reference frame as the drawing objects themselves. Thus the apparent size of a world-reference pattern will change when you zoom in or out on-screen, or when you plot or print at different scales. If the Device option is chosen, then the segment lengths are measured in the reference frame of the computer screen, printer, or plotter. The apparent size of a device-reference pattern will remain constant on-screen and on paper regardless of the zoom factor or print scale. Both of these reference frames can be scaled to alter the line type from it's original definition.

See Also VCGetLTScaleWorld

VCGetLTScaleWorld
VCSetLTScaleWorld

Version 1.2

Description Specifies the line scale reference frame.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetLTScaleWorld(short* iError);
extern "C" void WINAPI VCSetLTScaleWorld(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetLTScaleWorldBP Lib "VCMAIN32.DLL" (iError As Integer, pRet As Double)
Declare Sub VCSetLTScaleWorld Lib "VCMAIN32.DLL" (iError As Integer, ByVal dRet As Double)
```

Delphi:
```
procedure VCGetLTScaleWorldBP(var iError: Integer; var pRet: Double); far;
procedure VCSetLTScaleWorld(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the scaling factor to apply

Notes There are two linetype reference frames for measuring the lengths of the solid and blank segments that make up a custom line type called World and Device. If the World option is chosen, then the segment lengths are measured in the same reference frame as the drawing objects themselves. Thus the apparent size of a world-reference pattern will change when you zoom in or out on-screen, or when you plot or print at different scales. If the Device option is chosen, then the segment lengths are measured in the reference frame of the computer screen, printer, or plotter. The apparent size of a device-reference pattern will remain constant on-screen and on paper regardless of the zoom factor or print scale. Both of these reference frames can be scaled to alter the line type from it's original definition.

See Also VCGetLTScaleDevice

VCGetMajorVersion

Version	1.2
Description	Returns the major version number of the current Corel Visual CADD program files.
Declaration	
C/C++:	extern "C" short WINAPI VCGetMajorVersion(void);
Visual Basic:	Declare Function VCGetMajorVersion Lib "VCMAIN32.DLL" () As Integer
Delphi:	function VCGetMajorVersion:Integer; far;
Parameters	*Returns* - major version number.
Notes	When running external applications with copies of Corel Visual CADD not provided with the application, it is a good idea to check the version to be sure that all API's needed by the application are supported in the version in use. Corel Visual CADD version numbers are broken into four parts. These are from most to least significance; Major, Minor, Dot and Internal. Major and minor are the most important and should always be checked. For example this API document was designed around Corel Visual CADD 2.0.1(major version 2, minor version 0, minor dot version 1).
See Also	VCGetMinorVersion, VCGetMinorDotVersion, VCGetMinorInternalVersion

{button ,AL(`Error Checking;Version checking',0,`',`')} Task Guide Examples

VCGetMDICount

Version	2.0
Description	Returns the number of active MDI(Multiple Document Interface) windows.
Declaration	
C/C++	extern "C" short WINAPI VCGetMDICount(short* iError);
Visual Basic	Declare Function VCGetMDICount Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Delphi	function VCGetMDICount(var iError: Integer):Integer; far;
Parameters	No additional parameters are used with this subroutine.
Notes	Corel Visual CADD supports the Multiple Document Interface(MDI) feature of Windows. MDI allows for multiple drawings to be opened in the same session. Each new drawing and each new drawing view are opened into their own active Window. These can then be manipulated as any other Window. However, Corel Visual CADD supports only 64 active drawing Windows whether they contain separate drawings or simply new views. VCGetMDICount returns the number of currently active worlds in the drawing session.
See Also	VCCreateMDIWindow

VCGetMenu
VCSetMenu

Version 1.2

Description Custom pull down menus are saved as ASCII text files with a *.MNU extension. These menus can be loaded into Corel Visual CADD.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetMenu(short* iError, char* sz);
extern "C" void WINAPI VCSetMenu(short* iError, char* sz);
```

Visual Basic:
```
Declare Function VCGetMenu Lib "VCMAIN32.DLL" (iError As Integer, ByVal sz As String) As Integer
Declare Sub VCSetMenu Lib "VCMAIN32.DLL" (iError As Integer, ByVal sz As String)
```

Delphi:
```
function VCGetMenu(var iError: Integer; sz: PChar):Integer; far;
procedure VCSetMenu(var iError: Integer; sz: PChar); far;
```

Parameters *sz* - the path and file name for the menu

See Also VCGetPopupButton

{button ,AL(`Custom Menus',0,`',`')} Task Guide Examples

VCGetMessageHandle
VCSetMessageHandle

Version	1.2
Description	Specifies the location by hWnd where Corel Visual CADD is to place the message string for each command.
Declaration	
C/C++:	extern "C" void WINAPI VCSetMessageHandle(HWND hWnd_);
Visual Basic:	Declare Sub VCSetMessageHandle Lib "VCMAIN32.DLL" (ByVal hWnd_ As Integer)
Delphi:	procedure VCSetMessageHandle(hWnd_: Integer); far;
Parameters	*hWnd* - the hWnd handle for the object to be used as the message area.
Notes	Every Windows object is reference by Windows using its hWnd. This is an integer and must be supplied to Corel Visual CADD in order for the messages to be routed to an external application.
See Also	VCSetDistanceHandle, VCSetAngleHandle, VCSetXYHandle

{button ,AL(`Utilizing a Custom Interface',0,`',`')} Task Guide Examples

VCGetMinorDotVersion

Version	1.2
Description	Returns the minor dot version number of the current Visual CADD program files.
Declaration	
C/C++:	extern "C" short WINAPI VCGetMinorDotVersion(void);
Visual Basic:	Declare Function VCGetMinorDotVersion Lib "VCMAIN32.DLL" () As Integer
Delphi:	function VCGetMinorDotVersion:Integer; far;
Parameters	returns - minor dot version number.
Notes	When running applications with Corel Visual CADD, it is a good idea to check the version to be sure that all API's needed by the application are supported in the version in use. Visual CADD version numbers are broken into four parts. These are from most to least significance; Major, Minor, Dot and Internal. Major and minor are the most important and should always be checked. For example this API document was designed around Numera Visual CADD 2.0.1(major version 2, minor version 0, minor dot version 1).
See Also	VCGetMinorVersion, VCGetMajorVersion, VCGetMinorInternalVersion

{button ,AL(`Error Checking;Valid World Checking',0,`',`')} Task Guide Examples

VCGetMinorInternalVersion

Version 1.2

Description Returns the minor internal version number of the current Visual CADD program files.

Declaration

C/C++: extern "C" short WINAPI VCGetMinorInternalVersion(void);

Visual Basic: Declare Function VCGetMinorInternalVersion Lib "VCMAIN32.DLL" () As Integer

Delphi: function VCGetMinorInternalVersion:Integer; far;

Parameters returns - minor internal version number.

Notes When running applications with copies of Corel Visual CADD, it is a good idea to check the version to be sure that all API's needed by the application are supported in the version in use. Visual CADD version numbers are broken into four parts. These are from most to least significance; Major, Minor, Dot and Internal. Major and minor are the most important and should always be checked. For example this API document was designed around Numera Visual CADD 2.0.1(major version 2, minor version 0, minor dot version 1).

See Also VCGetMinorVersion, VCGetMinorDotVersion, VCGetMajorVersion

{button ,AL(`Error Checking;Version checking',0,`',`')} Task Guide Examples

VCGetMinorVersion

Version 1.2

Description Returns the minor version number of the current Visual CADD program files.

Declaration

C/C++: extern "C" short WINAPI VCGetMinorVersion(void);

Visual Basic: Declare Function VCGetMinorVersion Lib "VCMAIN32.DLL" () As Integer

Delphi: function VCGetMinorVersion:Integer; far;

Parameters returns - minor version number.

Notes When running applications with Corel Visual CADD, it is a good idea to check the version to be sure that all API's needed by the application are supported in the version in use. Visual CADD version numbers are broken into four parts. These are from most to least significance; Major, Minor, Dot and Internal. Major and minor are the most important and should always be checked. For example this API document was designed around Numera Visual CADD 2.0.1(major version 2, minor version 0, minor dot version 1).

See Also VCGetMajorVersion, VCGetMinorDotVersion, VCGetMinorInternalVersion

{button ,AL(`Error Checking;Version checking',0,`',`')} Task Guide Examples

VCGetNumCopies
VCSetNumCopies

Version 1.2

Description Sets or receives the number of copies for the linear copy command.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetNumCopies(short* iError);
extern "C" void WINAPI VCSetNumCopies(short* iError, short i);
```

Visual Basic:
```
Declare Sub VCGetNumCopiesBP Lib "VCMAIN32.DLL" (iErr As Integer, dRet As Double)
Declare Sub VCSetNumCopies Lib "VCMAIN32.DLL" (iErr As Integer, ByVal i As Integer)
```

Delphi:
```
procedure VCGetNumCopiesBP(var iError: Integer; var dRet: Double); far;
procedure VCSetNumCopies(var iError: Integer; i: Integer); far;
```

Parameters I - the number of copies.

See Also VCGetNumRows, VCLinearCopy

VCGetNumRows
VCSetNumRows

Version 1.2

Description The number of rows used by the array copy command.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetNumRows(short* iError);
extern "C" void WINAPI VCSetNumRows(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetNumRows Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetNumRows Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetNumRows(var iError: Integer):Integer; far;
procedure VCSetNumRows(var iError: Integer; i: Integer); far;
```

Parameters *i* - the default number of rows.

See Also VCArrayCopy

VCGetOffsetDist
VCSetOffsetDist

Version 1.2

Description Specifies a fixed distance for the offset command.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetOffsetDist(short* iError);
extern "C" void WINAPI VCSetOffsetDist(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetOffsetDistBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetOffsetDist Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetOffsetDistBP(var iError: Integer; var dRet: Double); far;
procedure VCSetOffsetDist(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the desired offset distance.

Notes External applications that use the offset command possibly do not want the offset distance rubberband dynamically. When offset fixed is on the user simply picks which side of the entity to place the offset instead of side and distance. Specifies the distance to construct an offset from the entity if offset fixed is off.

See Also VCGetOffsetFixed, VCOffset

VCGetOffsetFixed
VCSetOffsetFixed

Version 1.2

Description Specifies the fixed distance offset setting.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetOffsetFixed(short* iError);
extern "C" void WINAPI VCSetOffsetFixed(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetOffsetFixed Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetOffsetFixed Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetOffsetFixed(var iError: Integer):Integer; far;
procedure VCSetOffsetFixed(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes External applications that use the offset command possibly do not want the offset distance rubberband dynamically. When offset fixed is off the user simply picks which side of the entity to place the offset instead of side and distance.

See Also VCGetOffsetDist, VCOffset

VCGetOleDllClassName
VCSetOleDllClassName

Version 2.0

Description Specifies the class name containing the function to execute from an OLE DLL.

Declaration

C/C++

```
extern "C" short WINAPI VCGetOleDllClassName(short* iError, char* szPath);
extern "C" void WINAPI VCSetOleDllClassName(short* iError, char* szPath);
```

Visual Basic

```
Declare Function VCGetOleDllClassName Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String) As Integer
Declare Sub VCSetOleDllClassName Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
```

Delphi

```
function VCGetOleDllClassName(var iError: Integer; szPath: PChar):Integer;
```

Parameters szPath - the OLE class name.
Return - the length of the returned string.

Notes Corel Visual CADD can run functions from within a DLL through the scripting language. This allows developers to create add on applications in a Windows DLL format and then simply reference functions contained in the DLL. Corel Visual CADD will load the DLL into memory and access the specified function. Generally, this is simply done through the Corel Visual CADD interface with the Assign Script command or the CMDEXT file. Please refer to Customizing Corel Visual CADD for more information this. An application can also launch the routines through the API.

In order to access the DLL function, Corel Visual CADD must know the DLL name, the name of the function and any command line arguments required. The command line arguments can only be passed as a character string. The engine then uses this information to launch the specified function.

OLE DLL are special cases of the standard DLL method. OLE DLL are created as exported class routines. Corel Visual CADD must handle this differently when accessing the functionality built into the DLL. This information is provided by an OLE DLL name, a class name containing the function, the function name and the command line argument for the function.

See Also VCGetDllRunCmdLine, VCGetDllRunFunction, VCGetDllRunName, VCGetOleDllFunctionCmdLine, VCGetOleDllFunctionName, VCGetOleDllName

VCGetOleDllFunctionCmdLine
VCSetOleDllFunctionCmdLine

Version 2.0

Description Specifies the command line for a function contained in the OLE DLL.

Declaration

C/C++

```
extern "C" short WINAPI VCGetOleDllFunctionCmdLine(short* iError, char* szPath);
extern "C" void WINAPI VCSetOleDllFunctionCmdLine(short* iError, char* szPath);
```

Visual Basic

```
Declare Function VCGetOleDllFunctionCmdLine Lib "VCMAIN32.DLL" (iError As Integer, ByVal
szPath as String) As Integer
Declare Sub VCSetOleDllFunctionCmdLine Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath
As String)
```

Delphi

```
function VCGetOleDllFunctionCmdLine(var iError: Integer; szPath:
```

Parameters *szPath* - the command line for the function.
Return - the length of the returned string.

Notes Corel Visual CADD can run functions from within a DLL through the scripting language. This allows developers to create add on applications in a Windows DLL format and then simply reference functions contained in the DLL. Corel Visual CADD will load the DLL into memory and access the specified function. Generally, this is simply done through the Corel Visual CADD interface with the Assign Script command or the CMDEXT file. Please refer to Customizing Corel Visual CADD for more information this. An application can also launch the routines through the API.

In order to access the DLL function, Corel Visual CADD must know the DLL name, the name of the function and any command line arguments required. The command line arguments can only be passed as a character string. The engine then uses this information to launch the specified function.

OLE DLL are special cases of the standard DLL method. OLE DLL are created as exported class routines. Corel Visual CADD must handle this differently when accessing the functionality built into the DLL. This information is provided by an OLE DLL name, a class name containing the function, the function name and the command line argument for the function.

See Also VCGetDllRunCmdLine, VCGetDllRunFunction, VCGetDllRunName, VCGetOleDllClassName, VCGetOleDllFunctionName, VCGetOleDllName

VCGetOleDllFunctionName
VCSetOleDllFunctionName

Version 2.0

Description The function name contained in the OLE DLL.

Declaration

C/C++
```
extern "C" short WINAPI VCGetOleDllFunctionName(short* iError, char* szPath);
extern "C" void WINAPI VCSetOleDllFunctionName(short* iError, char* szPath);
```

Visual Basic
```
Declare Function VCGetOleDllFunctionName Lib "VCMAIN32.DLL" (iError As Integer, ByVal
szPathAs String) As Integer
Declare Sub VCSetOleDllFunctionName Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As
String)
```

Delphi
```
function VCGetOleDllFunctionName(var iError: Integer; szPath: PChar):Integer;
```

Parameters szPath - the name of the function.
Return - the lenght of the returned string.

Notes Corel Visual CADD can run functions from within a DLL through the scripting language. This allows developers to create add on applications in a Windows DLL format and then simply reference functions contained in the DLL. Corel Visual CADD will load the DLL into memory and access the specified function. Generally, this is simply done through the Corel Visual CADD interface with the Assign Script command or the CMDEXT file. Please refer to Customizing Corel Visual CADD for more information this. An application can also launch the routines through the API.

In order to access the DLL function, Corel Visual CADD must know the DLL name, the name of the function and any command line arguments required. The command line arguments can only be passed as a character string. The engine then uses this information to launch the specified function.

OLE DLL are special cases of the standard DLL method. OLE DLL are created as exported class routines. Corel Visual CADD must handle this differently when accessing the functionality built into the DLL. This information is provided by an OLE DLL name, a class name containing the function, the function name and the command line argument for the function.

See Also VCGetDllRunCmdLine, VCGetDllRunFunction, VCGetDllRunName, VCGetOleDllClassName, VCGetOleDllFunctionCmdLine, VCGetOleDllName

VCGetOleDllName
VCSetOleDllName

Version 2.0

Description The OLE DLL name.

Declaration

C/C++
```
extern "C" short WINAPI VCGetOleDllName(short* iError, char* szPath);
extern "C" void WINAPI VCSetOleDllName(short* iError, char* szPath);
```

Visual Basic
```
Declare Function VCGetOleDllName Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String) As Integer
Declare Sub VCSetOleDllName Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String
```

Delphi
```
function VCGetOleDllName(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetOleDllClassName(var iError: Integer; szPath: PChar); far;
```

Parameters szPath - the OLE DLL name.
Return - the length of the returned string.

Notes Corel Visual CADD can run functions from within a DLL through the scripting language. This allows developers to create add on applications in a Windows DLL format and then simply reference functions contained in the DLL. Corel Visual CADD will load the DLL into memory and access the specified function. Generally, this is simply done through the Corel Visual CADD interface with the Assign Script command or the CMDEXT file. Please refer to Customizing Corel Visual CADD for more information this. An application can also launch the routines through the API.

In order to access the DLL function, Corel Visual CADD must know the DLL name, the name of the function and any command line arguments required. The command line arguments can only be passed as a character string. The engine then uses this information to launch the specified function.

OLE DLL are special cases of the standard DLL method. OLE DLL are created as exported class routines. Corel Visual CADD must handle this differently when accessing the functionality built into the DLL. This information is provided by an OLE DLL name, a class name containing the function, the function name and the command line argument for the function.

See Also VCGetDllRunCmdLine, VCGetDllRunFunction, VCGetDllRunName, VCGetOleDllClassName, VCGetOleDllFunctionCmdLine, VCGetOleDllFunctionName

VCGetOrthoAng
VCSetOrthoAng

Version 1.2

Description Specifies the ortho angle setting.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetOrthoAng(short* iError);
extern "C" void WINAPI VCGetOrthoAngBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetOrthoAng(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetOrthoAngBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetOrthoAng Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetOrthoAngBP(var iError: Integer; var dRet: Double); far;
procedure VCSetOrthoAng(var iError: Integer; dRet: Double); far;
```

Parameters *d* - ortho angle setting in radians

Notes Ortho mode is one the more useful features of Corel Visual CADD that constrains many construction and editing tools. This constraint only allows point placements along lines that lie at 90° increments from, or on, the ortho angle, from the first point placed in the command.

See Also VCGetOrthoMode

VCGetOrthoMode
VCSetOrthoMode

Version 1.2

Description Specifies the state of the ortho toggle.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetOrthoMode(short* iError);
extern "C" void WINAPI VCSetOrthoMode(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetOrthoMode Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetOrthoMode Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetOrthoMode(var iError: Integer):Integer; far;
procedure VCSetOrthoMode(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes Ortho mode is one the more useful features of Corel Visual CADD that constrains many construction and editing tools. This constraint only allows point placements along lines that lie at 90° increments from, or on, the ortho angle, from the first point placed in the command.

See Also VCGetOrthoAng

VCGetPlotterCount

Version 2.0

Description Retrieves the number of plotters currently available for the direct plot command.

Declaration

C/C++ extern "C" short WINAPI VCGetPlotterCount(short* iError);

Visual Basic Declare Function VCGetPlotterCount Lib "VCDLG32.DLL" (iError As Integer) As Integer

Delphi function VCGetPlotterCount(var iError: Integer):Integer; far;

Parameters *returns* - a count for the number of installed plotters.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

If a plotter is not supported by drivers provided, an application or end user may create a new driver form the plotters language control. This requires the user or application to name the new driver being created. The actual plotter language strings are then defined through the API or Corel Visual CADD interface.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDeInitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterCurrentLanguageName
VCSetPlotterCurrentLanguageName

Version 2.0

Description Specifies the current plotter language name.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterCurrentLanguageName(short* iError, char* szLanguageName);

Visual Basic Declare Sub VCSetPlotterCurrentLanguageName Lib "VCDLG32.DLL" (iError As Integer, ByVal szLanguageName As String)

Delphi procedure VCSetPlotterCurrentLanguageName(var iError: Integer; szLanguageName:

Parameters szLanguage - the name for the current plotter language

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs to be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDeInitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterCurrentPageSize
VCSetPlotterCurrentPageSize

Version 2.0

Description Sets the current page size for the direct plot routine.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterCurrentPageSize(short* iError, short iIndex);

Visual Basic Declare Sub VCSetPlotterCurrentPageSize Lib "VCDLG32.DLL" (iError As Integer, ByVal iIndex As Integer)

Delphi procedure VCSetPlotterCurrentPageSize(var iError: Integer; iIndex: Integer);

Parameters iIndex - the index specifying the current page size values

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine allows for custom page sizes to be defined with the VCAddPlotterPageSizeRoutine and by the user through the Corel Visual CADD interface. These can be removed from the interface by the user or through the API with VCRemovePLotterPageSize. Custom page sizes enhance the users control over vector output devices and allows the user or an application to set page parameters suited to a desired output.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDelnitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterCurrentPenMapName
VCSetPlotterCurrentPenMapName

Version 2.0

Description Specifies the current pen map name.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterCurrentPenMapName(short* iError, char* szPenMapName);

Visual Basic Declare Sub VCSetPlotterCurrentPenMapName Lib "VCDLG32.DLL" (iError As Integer, ByVal szPenMapName As String)

Delphi procedure VCSetPlotterCurrentPenMapName(var iError: Integer; szPenMapName:

Parameters szName - the name of the current plotter pen map

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDeInitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterDeInitString
VCSetPlotterDeInitString

Version 2.0

Description Describes the commands that are sent to the plotter after the plot is complete.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterDeInitString(short* iError, char* sz);

Visual Basic Declare Sub VCSetPlotterDeInitString Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String)

Delphi procedure VCSetPlotterDeInitString(var iError: Integer; sz: PChar); far;

Parameters sz - the de-initialization string for the plotter.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs t be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDeInitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterDelimiter
VCSetPlotterDelimiter

Version 2.0

Description Specifies the character that separates commands sent to the plotter. This field can be left blank.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterDelimiter(short* iError, char* sz);

Visual Basic Declare Sub VCSetPlotterDelimiter Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String)

Delphi procedure VCSetPlotterDelimiter(var iError: Integer; sz: PChar); far;

Parameters sz - the plotter delimiter string

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs t be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDelnitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterInitString
VCSetPlotterInitString

Version 2.0

Description Specifies the commands sent to the plotter to initialize the plot.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterInitString(short* iError, char* sz);

Visual Basic Declare Sub VCSetPlotterInitString Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String)

Delphi procedure VCSetPlotterInitString(var iError: Integer; sz: PChar); far;

Parameters sz - the plotter initialization string.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs t be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDelnitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterPenChangeString
VCSetPlotterPenChangeString

Version 2.0

Description Specifies the characters that signal the plotter to change to a different pen.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterPenChangeString(short* iError, char* sz);

Visual Basic Declare Sub VCSetPlotterPenChangeString Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String)

Delphi procedure VCSetPlotterPenChangeString(var iError: Integer; sz: PChar); far;

Parameters sz - the plotter pen change string.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs t be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDeInitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterPenDownString
VCSetPlotterPenDownString

Version 2.0

Description Specifies which characters lower the pen to the paper.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterPenDownString(short* iError, char* sz);

Visual Basic Declare Sub VCSetPlotterPenDownString Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String)

Delphi procedure VCSetPlotterPenDownString(var iError: Integer; sz: PChar); far;

Parameters sz - the plotter pen down string.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs t be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDeInitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterPenDrawString
VCSetPlotterPenDrawString

Version 2.0

Description Specifies the characters that signal the plotter to move the pen from one location to anther in the down position.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterPenDrawString(short* iError, char* sz);

Visual Basic Declare Sub VCSetPlotterPenDrawString Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String)

Delphi procedure VCSetPlotterPenDrawString(var iError: Integer; sz: PChar); far;

Parameters sz - the plotter pen draw string.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs t be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDeInitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterPenMoveString
VCSetPlotterPenMoveString

Version 2.0

Description Specifies the characters that signal the plotter to move from one location to another in the up position.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterPenMoveString(short* iError, char* sz);

Visual Basic Declare Sub VCSetPlotterPenMoveString Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String)

Delphi procedure VCSetPlotterPenMoveString(var iError: Integer; sz: PChar); far;

Parameters sz - the plotter pen move string.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs t be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDeInitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterPenSpeedString
VCSetPlotterPenSpeedString

Version 2.0

Description Sets the speed at which a pen moves across the paper.

Declaration

C/C++
```
extern "C" short WINAPI VCGetPlotterPenSpeedString(short* iError, char* sz);
extern "C" void WINAPI VCSetPlotterPenUpString(short* iError, char* sz);
```

Visual Basic
```
Declare Function VCGetPlotterPenSpeedString Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String) As Integer
Declare Sub VCSetPlotterPenSpeedString Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String)
```

Delphi
```
function VCGetPlotterPenSpeedString(var iError: Integer; sz: PChar):Integer;far;
procedure VCSetPlotterPenSpeedString(var iError: Integer; sz: PChar); far;
```

Parameters *szString* - the plotter speed in millimeters per second

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device. Pen speed is measured in millimeters per second. Specifying a high pen speed may result in damage to the pen tip.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDeInitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPlotterPenUpString
VCSetPlotterPenUpString

Version 2.0

Description Specifies which characters raise the pen from the paper.

Declaration

C/C++
```
extern "C" short WINAPI VCGetPlotterPenUpString(short* iError, char* sz);
extern "C" void WINAPI VCSetPlotterPenUpString(short* iError, char* sz);
```

Visual Basic
```
Declare Function VCGetPlotterPenUpString Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String) As Integer
Declare Sub VCSetPlotterPenUpString Lib "VCDLG32.DLL" (iError As Integer, ByVal sz As String)
```

Delphi
```
function VCGetPlotterPenUpString(var iError: Integer; sz: PChar):Integer;far;
procedure VCSetPlotterPenUpString(var iError: Integer; sz: PChar); far;
```

Parameters sz - string for the languages pen up commad

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs to be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type.

See Also VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName, VCGetPlotterDeInitString, VCGetPlotterDelimiter, VCGetPlotterInitString, VCGetPlotterLanguageCount, VCGetPlotterLanguageName, VCGetPlotterPageSize, VCGetPlotterPageSizeCount, VCGetPlotterPenChangeString, VCGetPlotterPenDownString, VCGetPlotterPenDrawString, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping, VCGetPlotterPenMoveString, VCGetPlotterPenSpeedString, VCGetPlotterPenUpString

VCGetPointDisplay
VCSetPointDisplay

Version 1.2

Description Determines if point entities are displayed on the screen. Turning off the display will reduce the visual clutter and increase the speed of redraws.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetPointDisplay(short* iError);
extern "C" void WINAPI VCSetPointDisplay(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetPointDisplay Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetPointDisplay Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetPointDisplay(var iError: Integer):Boolean; far;
procedure VCSetPointDisplay(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetConstPt, VCGetLinetypeDisplay, VCGetLinewidthDisplay

VCGetPopupButton
VCSetPopupButton

Version 1.2

Description Determines which button is used to activate the context sensitive pop-up menus.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetPopupButton(short* iError);
extern "C" void WINAPI VCSetPopupButton(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetPopupButton Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetPopupButton Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetPopupButton(var iError: Integer):Integer; far;
procedure VCSetPopupButton(var iError: Integer; i: Integer); far
```

Parameters No additional parameters are used with this subroutine.

See Also VCAddPopupCommand, VCDeletePopupMenu, VCGetMenu

{button ,AL(`Custom Mouse Menus',0,`',`')} Task Guide Examples

VCGetPreserveAcadColorNums
VCSetPreserveAcadColorNums

Version 1.2

Description Color translation variable for AutoCAD file conversion.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetPreserveAcadColorNums(short* iError);
extern "C" void WINAPI VCSetPreserveAcadColorNums(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetPreserveAcadColorNums Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetPreserveAcadColorNums Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetPreserveAcadColorNums(var iError: Integer):Boolean; far;
procedure VCSetPreserveAcadColorNums(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes Corel Visual CADD allows color numbers to be preserved in the translations (this option may be more important for users of pen plotters, even though this may cause object colors to change) or if the colors numbers should be changed so that the on-screen colors are preserved during the translation (this option should be selected if it is more important for the drawing to look the same after translation).

See Also VCAcadRead, VCGetKeepGCDFontName, VCGetKeepAcadFontName, VCGetGCDDefaultHatchName, VCGetAcadImportUnit

VCGetPrintSettings
VCSetPrintSettings

Version 2.0

Description Specifies the current print settings used by the Print command.

Declaration

C/C++
```
extern "C" void WINAPI VCGetPrintSettings(short* iError, PrintStruct* pSettings);
extern "C" void WINAPI VCSetPrintSettings(short* iError, PrintStruct* pSettings);
```

Visual Basic
```
Declare Sub VCGetPrintSettings Lib "VCDLG32.DLL" (iError As Integer, pSettings As PrintStruct)
Declare Sub VCSetPrintSettings Lib "VCDLG32.DLL" (iError As Integer, pSettings As PrintStruct)
```

Delphi
```
procedure VCGetPrintSettings(var iError: Integer; var pSettings: PrintStruct);far;
procedure VCSetPrintSettings(var iError: Integer; var pSettings: PrintStruct);far;
```

Parameters *pSettings* - the structure containing the print settings. See the Common Types Section for a detail on the Corel Visual CADD structure types.

Notes Corel Visual CADD contains both a Print and Plot command. The print command utilizes the standard Windows drivers for output to the device. The plot command is an internal routine allowing more control over vector output devices by bypassing the Windows drivers. Each of these commands maintain separate default settings for the print output such as scale, orientation and page size. These settings are maintained in a structure defined for Corel Visual CADD.

See Also VCGetPrinterName, VCGetPrinterNameCount, VCLoadPlotterDriver

VCGetPrinterName

Version	2.0
Description	Returns the name of the printer at the specified index.
Declaration	
C/C++	extern "C" short WINAPI VCGetPrinterName(short* iError, short iIndex, char* szPrinter);
Visual Basic	Declare Function VCGetPrinterName Lib "VCDLG32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal szPrinter As String) As Integer
Delphi	function VCGetPrinterName(var iError: Integer; iIndex: Integer; szPrinter:
Parameters	*iIndex* - index for the printer name to retrieve *szPrinter* - returned printer name
Notes	
See Also	VCGetPrintSettings, VCGetPrinterNameCount, VCLoadPlotterDriver

VCGetPrinterNameCount

Version	2.0
Description	Returns a count of the currently installed printers.
Declaration	
C/C++	extern "C" short WINAPI VCGetPrinterNameCount(short* iError);
Visual Basic	Declare Function VCGetPrinterNameCount Lib "VCDLG32.DLL" (iError As Integer) As Integer
Delphi	function VCGetPrinterNameCount(var iError: Integer):Integer; far;
Parameters	Returns a count for the number of installed printers.
Notes	
See Also	VCGetPrintSettings, VCGetPrinterNameCount, VCLoadPlotterDriver

VCGetPrompt
VCSetPrompt

Version 1.2

Description Specifies the prompt for a User Tool.

Declaration

C/C++: extern "C" short WINAPI VCGetPrompt(short iPromptIndex, char* szPrompt);
extern "C" BOOL WINAPI VCSetPrompt(short iPromptIndex, char* szPrompt);

Visual Basic: Declare Function VCGetPrompt Lib "VCTOOL32.DLL" (ByVal iPromptIndex As Integer, ByVal szPrompt As String) As Integer
Declare Function VCSetPrompt Lib "VCTOOL32.DLL" (ByVal iPromptIndex As Integer, ByVal szPrompt As String) As Integer

Delphi: function VCGetPrompt(iPromptIndex: Integer; szPrompt: PChar):Integer; far;
function VCSetPrompt(iPromptIndex: Integer; szPrompt: Pchar); Integer; far;

Parameters *iPromptIndex* - the step number to which the prompt is assigned.
szPrompt - a string representing the prompt to be displayed.

Notes When custom tools are created, prompts should always be displayed to the user in order to explain what steps or input is required at each step. The first prompt is set with VCSetUserTool and all subsequent prompts should be set with VCSetPrompt. VCGetPrompt will conversely return any of the existing prompts.

See Also VCSetAlertApp, VCGetUserToolLBDown

VCGetQuickSearch
VCSetQuickSearch

Version 1.2

Description The Quick Search toggle enables a faster search method for objects in the drawing..

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetQuickSearch(short* iError);
extern "C" void WINAPI VCSetQuickSearch(short* iError, BOOL tfQS);
```

Visual Basic:
```
Declare Function VCGetQuickSearch Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetQuickSearch Lib "VCMAIN32.DLL" (iError As Integer, ByVal tfQS As Integer)
```

Delphi:
```
function VCGetQuickSearch(var iError: Integer):Boolean; far;
procedure VCSetQuickSearch(var iError: Integer; tfQS: Boolean); far
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes With Quick Search toggled ON Corel Visual CADD will select the first object it finds within the search tolerance - not necessarily the nearest object but the first object in the database that is within the tolerance. Quick Search is most useful when your drawing is very large and you are zoomed in far enough not to have too many competing points in the area around the cursor. If Backward Redraw is ON, the first object that Quick Search will find will actually be the most recent object placed within the tolerance

See Also VCGetCursorSize

VCGetRadCopies
VCSetRadCopies

Version 1.2

Description Specifies the number of copies used in a radial copy command.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetRadCopies(short* iError);
extern "C" void WINAPI VCSetRadCopies(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetRadCopies Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetRadCopies Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetRadCopies(var iError: Integer):Integer; far;
procedure VCSetRadCopies(var iError: Integer; i: Integer); far;
```

Parameters I - number of radial copies.

See Also VCGetNumCopies

VCGetReadOnly
VCSetReadOnly

Version 2.0

Description Specifies the read only property for the active drawing.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetReadOnly(short* iError, short* iReadOnly);
extern "C" void WINAPI VCSetReadOnly(short* iError, short iReadOnly);
```

Visual Basic
```
Declare Function VCGetReadOnly Lib "VCMAIN32.DLL" (iError As Integer, iReadOnly As Integer)
As Integer
Declare Sub VCSetReadOnly Lib "VCMAIN32.DLL" (iError As Integer, ByVal iReadOnly As Integer)
```

Delphi
```
function VCGetReadOnly(var iError: Integer; var iReadOnly: Integer):Boolean;
```

Parameters *iReadOnly* - toggle indicating the read only state of the active drawing
0 - drawing is read only
1 - drawing is not read only

Notes Corel Visual CADD supports file locking an read only access for enhanced network support. This toggle sets or retrieves the current Read Only state for the active drawing.

See Also

VCGetRefFrame
VCSetRefFrame

Version 2.0

Description Specifies the frame coordinates of the bounding rectangle.

Declaration

C/C++
```
extern "C" void WINAPI VCGetRefFrame(short* iError, Point2D* dpLL, Point2D* dpUR);
extern "C" void WINAPI VCSetRefFrameNameDlg(short* iError);
```

Visual Basic
```
Declare Sub VCGetRefFrame Lib "VCMAIN32.DLL" (iError As Integer, dpLL As Point2D, dpUR As Point2D)
Declare Sub VCSetRefFrameNameDlg Lib "VCDLG32.DLL" (iError As Integer)
```

Delphi
```
procedure VCGetRefFrame(var iError: Integer; var dpLL: Point2D; var dpUR:
```

Parameters *dpLL* - the Point2D coordinate pair containing the lower left corner of the reference frame
dpUR - the Point2D coordinate pair containing the upper right corner of the reference frame

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetRefFrameColor
VCSetRefFrameColor

Version 2.0

Description Specifies the reference frame color.

Declaration

C/C++
```
extern "C" short WINAPI VCGetRefFrameColor(short* iError);
extern "C" void WINAPI VCSetRefFrameColor(short* iError, short iC);
```

Visual Basic
```
Declare Function VCGetRefFrameColor Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetRefFrameColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal iC As Integer)
```

Delphi
```
function VCGetRefFrameColor(var iError: Integer):Integer; far;
procedure VCSetReadOnly(var iError: Integer; iReadOnly: Integer); far;
```

Parameters *iC* - the color of the reference frame boundary

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetRefFrameDrawBoundary
VCSetRefFrameDrawBoundary

Version 2.0

Description Specifies if the reference frame boundary is displayed and printed in the drawing.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetRefFrameDrawBoundary(short* iError);
extern "C" void WINAPI VCSetRefFrameDrawBoundary(short* iError, vbool vb);
```

Visual Basic
```
Declare Function VCGetRefFrameDrawBoundary Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetRefFrameDrawBoundary Lib "VCMAIN32.DLL" (iError As Integer, ByVal vb As Integer)
```

Delphi
```
function VCGetRefFrameDrawBoundary(var iError: Integer):Boolean; far;
procedure VCSetRefFrameDrawBoundary(var iError: Integer; vb: Boolean); far;
```

Parameters vb - flag for displaying the reference frame boundary.
0 - do not show the boundary.
1 - show the boundary.

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetRefFrameIsDataBound
VCSetRefFrameIsDataBound

Version 2.0

Description Specifies if the reference frame information is bound to the current drawing.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetRefFrameIsDataBound(short* iError);
extern "C" void WINAPI VCSetRefFrameIsDataBound(short* iError, BOOL tf);
```

Visual Basic
```
Declare Function VCGetRefFrameIsDataBound Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetRefFrameIsDataBound Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetRefFrameIsDataBound(var iError: Integer):Boolean; far;
procedure VCSetRefFrameIsDataBound(var iError: Integer; tf: Boolean); far;
```

Parameters tf - determines if data in the reference is bound to the active file.
0 - the data is not bound.
1 - the data is bound.

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetRefFrameIsDynamic

Version 2.0

Description Determines if the reference frame is dynamic.

Declaration

C/C++ extern "C" vbool WINAPI VCGetRefFrameIsDynamic(short* iError);

Visual Basic Declare Function VCGetRefFrameIsDynamic Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi function VCGetRefFrameIsDynamic(var iError: Integer):Boolean; far;

Parameters return - value for the reference frame.
0 - the reference frame is not dynamic.
1 - the reference frame is dynamic.

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetRefFrameLineWidth
VCSetRefFrameLineWidth

Version 2.0

Description Specifies the line width for the reference frame.

Declaration

C/C++
```
extern "C" short WINAPI VCGetRefFrameLineWidth(short* iError);
extern "C" void WINAPI VCSetRefFrameLineWidth(short* iError, short iC);
```

Visual Basic
```
Declare Function VCGetRefFrameLineWidth Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetRefFrameLineWidth Lib "VCMAIN32.DLL" (iError As Integer, ByVal iC As Integer)
```

Delphi
```
function VCGetRefFrameLineWidth(var iError: Integer):Integer; far;
procedure VCSetRefFrameLineWidth(var iError: Integer; iC: Integer); far;
```

Parameters iC - the line width index value from 0 - 16.

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetRefFrameName
VCSetRefFrameName

Version 2.0

Description Specifies the file pointed to by the reference frame entity.

Declaration

C/C++
```
extern "C" short WINAPI VCGetRefFrameName(short* iError, char* s);
extern "C" void WINAPI VCSetRefFrameNameDlg(short* iError);
```

Visual Basic
```
Declare Function VCGetRefFrameName Lib "VCMAIN32.DLL" (iError As Integer, ByVal s As String)
As Integer
Declare Sub VCSetRefFrameNameDlg Lib "VCDLG32.DLL" (iError As Integer)
```

Delphi
```
function VCGetRefFrameName(var iError: Integer; s: PChar):Integer; far;
procedure VCSetRefFrameNameDlg(var iError: Integer); far;
```

Parameters s - the name of the reference frame.
returns - the length of the returned string.

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetRefFrameOffset
VCSetRefFrameOffset

Version 2.0

Description Specifies the reference frame offset.

Declaration

C/C++
```
extern "C" void WINAPI VCGetRefFrameOffset(short* iError, Point2D* dpOffset);
extern "C" void WINAPI VCSetRefFrameOffset(short* iError, Point2D* dpOffset);
```

Visual Basic
```
Declare Sub VCGetRefFrameOffset Lib "VCMAIN32.DLL" (iError As Integer, dpOffset As Point2D)
Declare Sub VCSetRefFrameOffset Lib "VCMAIN32.DLL" (iError As Integer, dpOffset As Point2D)
```

Delphi
```
procedure VCGetRefFrameOffset(var iError: Integer; var dpOffset: Point2D);
```

Parameters dpOffset -

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetRefFrameRot
VCSetRefFrameRot

Version 2.0

Description Specifies the reference frame rotation relative to the x plane.

Declaration

C/C++
```
extern "C" void WINAPI VCGetRefFrameRot(short* iError, double* dR);
extern "C" void WINAPI VCSetRefFrameRot(short* iError, double* dR);
```

Visual Basic
```
Declare Sub VCGetRefFrameRot Lib "VCMAIN32.DLL" (iError As Integer, dR As Double)
Declare Sub VCSetRefFrameRot Lib "VCMAIN32.DLL" (iError As Integer, dR As Double)
```

Delphi
```
procedure VCGetRefFrameRot(var iError: Integer; var dR: Double); far;
procedure VCSetRefFrameOffset(var iError: Integer; var dpOffset: Point2D);
```

Parameters dR - the rotation for the reference frame in radians.

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetRefFrameScale
VCSetRefFrameScale

Version 2.0

Description Specifies the reference frame scale.

Declaration

C/C++
```
extern "C" void WINAPI VCGetRefFrameScale(short* iError, Point2D* dpP);
extern "C" void WINAPI VCSetRefFrameScale(short* iError, Point2D* dpP);
```

Visual Basic
```
Declare Sub VCGetRefFrameScale Lib "VCMAIN32.DLL" (iError As Integer, dpP As Point2D)
Declare Sub VCSetRefFrameScale Lib "VCMAIN32.DLL" (iError As Integer, dpP As Point2D)
```

Delphi
```
procedure VCGetRefFrameScale(var iError: Integer; var dpP: Point2D); far;
procedure VCSetRefFrameScale(var iError: Integer; var dpP: Point2D); far;
```

Parameters dpP - the Point2D structure containing the X and Y scale values.

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetRefFrameViewWidthHeight
VCSetRefFrameViewWidthHeight

Version 2.0

Description Specifies the reference frame height and width of a reference frame.

Declaration

C/C++
```
extern "C" void WINAPI VCGetRefFrameViewWidthHeight(short* iError, Point2D* dpWidthHeight);
extern "C" void WINAPI VCSetRefFrameViewWidthHeight(short* iError, Point2D* dpWidthHeight);
```

Visual Basic
```
Declare Sub VCGetRefFrameViewWidthHeight Lib "VCMAIN32.DLL" (iError As Integer,
dpWidthHeight As Point2D)
Declare Sub VCSetRefFrameViewWidthHeight Lib "VCMAIN32.DLL" (iError As Integer,
dpWidthHeight As Point2D)
```

Delphi
```
procedure VCGetRefFrameViewWidthHeight(var iError: Integer; var dpWidthHeight:
```

Parameters dpP - the Point2D structure containing the Widht (X) and Height (Y) values.

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

See Also VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameColor, VCGetRefFrameDrawBoundary, VCGetRefFrameIsDynamic, VCGetRefFrameLineWidth, VCGetRefFrameOffset, VCGetRefFrameRot, VCGetRefFrameScale, VCGetRefFrameViewWidthHeight

VCGetReplaceWithSymbol
VCSetReplaceWithSymbol

Version 1.2

Description In creating a symbol entity, the selected entities can be replaced by the newly defined symbol definition.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetReplaceWithSymbol(short* iError);
extern "C" void WINAPI VCSetReplaceWithSymbol(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetReplaceWithSymbol Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetReplaceWithSymbol Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetReplaceWithSymbol(var iError: Integer):Boolean; far;
procedure VCSetReplaceWithSymbol(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCSymbolCreate

VCGetRibalogSize

Version 1.2

Description Returns the upper left corner and height width of the ribalogs area.

Declaration

C/C++: extern "C" void WINAPI VCGetRibalogSize(short* iError, iPoint2D* ipOrg, iPoint2D* ipSize);

Visual Basic: Declare Sub VCGetRibalogSize Lib "VCDLG32.DLL" (iError As Integer, ipOrg As iPoint2D, ipSize As iPoint2D)

Delphi: procedure VCGetRibalogSize(var iError: Integer; var ipOrg: iPoint2D; ipSize: iPoint2D); far;

Parameters *ipOrg* - screen coordinates for upper left corner.
iPSize - height and width in screen coordinates for the ribalog area.

Notes Corel Visual CADD uses ribalogs to gather user input during the drawing session. In order to create a look and feel similar to the Corel Visual CADD interface, an application can create ribalogs for displaying information. VCGetRibalogSize returns the screen coordinates for the upper left corner and the height and width available for display. A form or dialog can then be formatted to fit inside the space.

See Also VCGetStatusBarSize

VCGetRPolyInscribe
VCSetRPolyInscribe

Version 1.2

Description Specifies the state of the inscribe toggle for regular polygons.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetRPolyInscribe(short* iError);
extern "C" void WINAPI VCSetRPolyInscribe(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetRPolyInscribe Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetRPolyInscribe Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetRPolyInscribe(var iError: Integer):Integer; far;
procedure VCSetRPolyInscribe(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes Regular polygons can be created one of two ways, either inscribed or circumscribed. Inscribed creation forces the second placement point on one of the vertices and circumscribed creation forces the second placement point on the middle of one of the sides.

See Also VCGetRPolySides

VCGetRPolySides
VCSetRPolySides

Version 1.2

Description Specifies the default setting for the number of sides in a regular polygon construction..

Declaration

C/C++:
```
extern "C" short WINAPI VCGetRPolySides(short* iError);
extern "C" void WINAPI VCSetRPolySides(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetRPolySides Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetRPolySides Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetRPolySides(var iError: Integer):Integer; far;
procedure VCSetRPolySides(var iError: Integer; i: Integer); far;
```

Parameters *I* - setting for number of regular polygon sides.

Notes Regular polygons are actually continuous lines and are not considered polygons by Corel Visual CADD once created. Because of this, it is not possible to change the number of sides without recreating the polygon. It is therefore necessary to set the number of sides before creating the polygon

See Also VCGetRPolyInscribe

VCGetRubberBandColor
VCSetRubberBandColor

Version 1.2

Description The rubberbanding display color.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetRubberBandColor(short* iError);
extern "C" void WINAPI VCSetRubberBandColor(short* iError, short i);
```

Visual Basic:
```
Declare Function VCGetRubberBandColor Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetRubberBandColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
```

Delphi:
```
function VCGetRubberBandColor(var iError: Integer):Integer; far;
procedure VCSetRubberBandColor(var iError: Integer; i: Integer); far;
```

Parameters *i* - the rubberbanding color

See Also VCGetCursorColor, VCGetBackgroundColor

VCGetSaveEnvOnExit
VCSetSaveEnvOnExit

Version 1.2

Description Specifies if the current settings are to be saved as the default for other drawing session on close.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetSaveEnvOnExit(short* iError);
extern "C" void WINAPI VCSetSaveEnvOnExit(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetSaveEnvOnExit Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetSaveEnvOnExit Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetSaveEnvOnExit(var iError: Integer):Boolean; far;
procedure VCSetSaveEnvOnExit(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCSaveStyle

VCGetSavePaths
VCSetSavePaths

Version 1.2

Description Determines if the same file paths are used the next time a file is opened.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetSavePaths(short* iError);
extern "C" void WINAPI VCSetSavePaths(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetSavePaths Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetSavePaths Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetSavePaths(var iError: Integer):Boolean; far;
procedure VCSetSavePaths(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDWGPath, VCGetDXFPath, VCGetGCDPath, VCGetSYSPath, VCGetVCDPath, VCGetVCSPath, VCGetCMPPath, VCGetVCFPath

VCGetScaleX
VCSetScaleX

Version 1.2

Description Specifies x scale factor for the scale modify command. The scale modify command uses both a x and y scale factor to differentially scale a selected entity or group of entities.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetScaleX(short* iError);
extern "C" void WINAPI VCGetScaleXBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetScaleX(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetScaleXBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetScaleX Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetScaleXBP(var iError: Integer; var dRet: Double); far;
procedure VCSetScaleX(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the x scale factor currently set.

See Also VCGetScaleXY, VCGetScaleY

VCGetScaleXY
VCSetScaleXY

Version 1.2

Description Specifies both the x and y scale factor for the scale modify command. The scale modify command uses both a x and y scale factor to differentially scale a selected entity or group of entities.

Declaration

C/C++:
```
extern "C" Point2D WINAPI VCGetScaleXY(short* iError);
extern "C" void WINAPI VCGetScaleXYBP(short* iError, Point2D* pRet);
extern "C" void WINAPI VCSetScaleXY(short* iError, Point2D p);
```

Visual Basic:
```
Declare Sub VCGetScaleXYBP Lib "VCMAIN32.DLL" (iError As Integer, pRet As Point2d)
Declare Sub VCSetScaleXY Lib "VCMAIN32.DLL" (iError As Integer, p As Point2d)
```

Delphi:
```
procedure VCGetScaleXYBP(var iError: Integer; var pRet: Point2D); far;
procedure VCSetScaleXY(var iError: Integer; dRet: Poin2D); far;
```

Parameters p contains the x and y values for the x and y scale.

See Also VCGetScaleX, VCGetScaleY

VCGetScaleY
VCSetScaleY

Version 1.2

Description Specifies y scale factor for the scale modify command. The scale modify command uses both a x and y scale factor to differentially scale a selected entity or group of entities.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetScaleY(short* iError);
extern "C" void WINAPI VCGetScaleYBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetScaleY(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetScaleYBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetScaleY Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetScaleYBP(var iError: Integer; var dRet: Double); far;
procedure VCSetScaleY(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the y scale factor to be set.

See Also VCGetScaleX, VCGetScaleXY

VCGetSCRPath
VCSetSCRPath

Version 1.2

Description The default file path for the script files. Scripts are macros used in the Corel Visual CADD interface to automate common tasks. The scripts are saved in a text file SCRIPT.DEF located in the script path.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetSCRPath(short* iError, char* szPath);
extern "C" void WINAPI VCSetSCRPath(short* iError, char* szPath);
```

Visual Basic:
```
Declare Function VCGetSCRPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
As Integer
procedure VCSetSCRPath(var iError: Integer; szPath: PChar); far;
```

Delphi:
```
function VCGetSCRPath(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetSCRPath(var iError: Integer; szPath: PChar); far;
```

Parameters *Path* - string returned containing the current symbol path.

See Also VCGetDWGPath, VCGetDXFPath, VCGetGCDPath, VCGetSYSPath, VCGetVCDPath, VCGetVCSPath, VCGetCMPPath, VCGetVCFPath

VCGetSecondaryDistFormat
VCSetSecondaryDistFormat

Version 2.0

Description Specifies dimensions are displayed using both the primary and secondary units.

Declaration

C/C++
```
extern "C" short WINAPI VCGetSecondaryDistFormat(short* iError);
extern "C" void WINAPI VCSetSecondaryDistFormat(short* iError, short iF_);
```

Visual Basic
```
Declare Function VCGetSecondaryDistFormat Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetSecondaryDistFormat Lib "VCMAIN32.DLL" (iError As Integer, ByVal iF_ As Integer)
```

Delphi
```
function VCGetSecondaryDistFormat(var iError: Integer):Integer; far;
```

Parameters iF - flag indicating the distance display format.
0 - do not use both the primary and secondary units in the dimension string.
1 - use both the primary and secondary units in the dimension string.

Notes

See Also

VCGetShiftClick
VCSetShiftClick

Version 1.2

Description Specifies whether or not pressing the Shift key while clicking the right mouse button activates the pop-up menus.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetShiftClick(short* iError);
extern "C" void WINAPI VCSetShiftClick(short* iError, BOOL tfShift);
```

Visual Basic:
```
Declare Function VCGetShiftClick Lib "VCMAIN32.DLL" (iErr As Integer) As Integer
Declare Sub VCSetShiftClick Lib "VCMAIN32.DLL" (iErr As Integer, ByVal tfShift As Integer)
```

Delphi:
```
function VCGetShiftClick(var iError: Integer):Boolean; far;
procedure VCSetShiftClick(var iError: Integer; tfShift: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetPopupButton

VCGetShortLayerList
VCSetShortLayerList

Version 1.2

Description Displays only layers that have been named or have data on them in Layer Manager.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetShortLayerList(short* iError);
extern "C" void WINAPI VCSetShortLayerList(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetShortLayerList Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetShortLayerList Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetShortLayerList(var iError: Integer):Boolean; far;
procedure VCSetShortLayerList(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetLayerDisplay, VCGetAllLayerEd, VCGetAllLayerSnap, VCGetLayerIndex

VCGetShowDrag
VCSetShowDrag

Version 1.2

Description Specifies if selected objects will visually drag across the screen during movement, placement and copy operations.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetShowDrag(short* iError);
extern "C" void WINAPI VCSetShowDrag(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetShowDrag Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetShowDrag Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetShowDrag(var iError: Integer):Boolean; far;
procedure VCSetShowDrag(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetConstPt, VCGetShowTangentPoints

VCGetShowTangentPoints
VCSetShowTangentPoints

Version 1.2

Description Option for displaying tangent points.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetShowTangentPoints(short* iError);
extern "C" void WINAPI VCSetShowTangentPoints(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetShowTangentPoints Lib "VCMAIN32.DLL" (iErr As Integer) As Integer
Declare Sub VCSetShowTangentPoints Lib "VCMAIN32.DLL" (iErr As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetShowTangentPoints(var iError: Integer):Boolean; far;
procedure VCSetShowTangentPoints(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

Notes When working with entities, it is sometimes convenient to display the entity construction points to aid in snapping. Turning off the display will reduce the visual clutter and increase the speed of redraws.

See Also VCGetConstPt

VCGetSingleUnitFrac
VCSetSingleUnitFrac

Version 1.2

Description Dimension fractions can be displayed as a single character (¼) or multiple characters separated by a slash (1/4). This option is available only for vector fonts which is determined with VCIsFontNameVText.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetSingleUnitFrac(short* iError);
extern "C" void WINAPI VCSetSingleUnitFrac(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetSingleUnitFrac Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetSingleUnitFrac Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetSingleUnitFrac(var iError: Integer):Boolean; far;
procedure VCSetSingleUnitFrac(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetDisplayFractionalValue

VCGetSnapPercentVal
VCSetSnapPercentVal

Version 1.2

Description The default value for the snap percent command.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetSnapPercentVal(short* iError);
extern "C" void WINAPI VCGetSnapPercentValBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetSnapPercentVal(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetSnapPercentValBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetSnapPercentVal Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetSnapPercentValBP(var iError: Integer; var dRet: Double); far;
procedure VCSetSnapPercentVal(var iError: Integer; dRet: Double); far;
```

Parameters *d* - the desired snap percentage value.

Notes Snap percent typically requires the user to enter a percentage value along the entity selected to snap. However, as with many commands, the interface is not available through the API. It is therefor necessary to preset this value using VCSetSnapPercentVal. This setting can also be retrieved with VCGetSnapPercentVal. This value can be above 100 or below 0 and will thus snap beyond the end of the entity. The end selected closest to is the 0 percent end.

See Also VCSnapPercent

VCGetSolid
VCSetSolid

Version 1.2

Description Specifies the current state of the auto fill of double lines. It is possible to automatically fill between double lines as they are placed using the current fill color.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetSolid(short* iError);
extern "C" void WINAPI VCSetSolid(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetSolid Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetSolid Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetSolid(var iError: Integer):Integer; far;
procedure VCSetSolid(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetFillColor, VCGetAutoFillet

VCGetSpanAngle
VCSetSpanAngle

Version 1.2

Description Specifies the default span angle for the radial copy command.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetSpanAngle(short* iError);
extern "C" void WINAPI VCGetSpanAngleBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetSpanAngle(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetSpanAngleBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetSpanAngle Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetSpanAngleBP(var iError: Integer; var dRet: Double); far;
procedure VCSetSpanAngle(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the current default span angle

Notes Many of the modify commands require user input in order for them to function correctly. In order to allow modifications to entities without displaying the interface, it was necessary to allow preset values for all the modify prompts.

See Also VCGetRadCopies

VCGetSpecificPrinter
VCSetSpecificPrinter

Version 2.0

Description Specifies a specific plotter or printer.

Declaration

C/C++
```
extern "C" short WINAPI VCGetSpecificPrinter(short* iError, char* szSpecificPrinter);
extern "C" void WINAPI VCSetSpecificPrinter(short* iError, char* szSpecificPrinter);
```

Visual Basic
```
Declare Function VCGetSpecificPrinter Lib "VCDLG32.DLL" (iError As Integer, ByVal
szSpecificPrinter As String) As Integer
Declare Sub VCSetSpecificPrinter Lib "VCDLG32.DLL" (iError As Integer, ByVal szSpecificPrinter
As String)
```

Delphi
```
function VCGetSpecificPrinter(var iError: Integer; szSpecificPrinter:PChar):Integer; far;
procedure VCSetSpecificPrinter(var iError: Integer; szSpecificPrinter: PChar);far;
```

Parameters *szSpecificPrinter* - the name of the printer.
Returns an the length of the string.

Notes

See Also

VCGetStatusBarSize

Version 1.2

Description Return the upper left and height and width of the status bar in screen coordinates.

Declaration

C/C++: extern "C" void WINAPI VCGetStatusBarSize(short* iError, iPoint2D* ipOrg, iPoint2D* ipSize);

Visual Basic: Declare Sub VCGetStatusBarSize Lib "VCDLG32.DLL" (iError As Integer, ipOrg As iPoint2D, ipSize As iPoint2D)

Delphi: procedure VCGetStatusBarSize(var iError: Integer; var ipOrg: iPoint2D; ipSize: iPoint2D); far;

Parameters iPOrg - screen coordinate for the upper left corner.
iPSize - screen units for the height and width.

Notes Corel Visual CADD utilizes the status bar to display details during the drawing session. VCGetStatusBarSize returns the size of the bar allowing the external application to create a custom status bar displaying information relevant to the application.

See Also VCGetRibalogSize

VCGetSymAutoExplode
VCSetSymAutoExplode

Version 1.2

Description Specifies if a symbol is automatically exploded when placed.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetSymAutoExplode(short* iError);
extern "C" void WINAPI VCSetSymAutoExplode(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetSymAutoExplode Lib "VCMAIN32.DLL" (iErr As Integer) As Integer
Declare Sub VCSetSymAutoExplode Lib "VCMAIN32.DLL" (iErr As Integer, ByVal tf As Integer
```

Delphi:
```
function VCGetSymAutoExplode(var iError: Integer):Boolean; far;
procedure VCSetSymAutoExplode(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetSymExplode

VCGetSymExplode
VCSetSymExplode

Version 1.2

Description Returns the option for layer control when exploding a symbol.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetSymExplode(short* iError);
extern "C" void WINAPI VCSetSymExplode(short* iError, short iEx);
```

Visual Basic:
```
Declare Function VCGetSymExplode Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetSymExplode Lib "VCMAIN32.DLL" (iError As Integer, ByVal iEx As Integer)
```

Delphi:
```
function VCGetSymExplode(var iError: Integer):Integer; far;
procedure VCSetSymExplode(var iError: Integer; iEx: Integer); far;
```

Parameters *iEx* - index for placement of symbol entities
0 - Individual Layer
1 - Placement Layer
2 - Current Layer

Notes When a symbol is exploded, three options are available for placing the resulting entities. *Placement Layer* - all objects that make up exploded symbols are placed on the same layer in the drawing as the symbol. *Current Layer* - all objects that make up exploded symbols are assigned to the layer that is current when the symbol is exploded. *Individual Layers* **-** each object within exploded symbols revert to the layer that was current when the object was drawn, prior to creation of the symbol

See Also VCGetSymName, VCGetSymbolName, VCGetSymScale, VCGetSymRot

VCGetSymName
VCSetSymName

Version 1.2

Description The symbol name for a symbol creation or the name of the currently selected symbol.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetSymName(short* iError, char* pS);
extern "C" void WINAPI VCSetSymName(short* iErroR, char* pS);
```

Visual Basic:
```
Declare Sub VCSetSymName Lib "VCMAIN32.DLL" ( iError As Integer, ByVal sz As String)
Declare Function VCGetSymName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String) As Integer
```

Delphi:
```
function VCGetSymName(var iError: Integer; pS: PChar):Integer; far;
procedure VCSetSymName(var iError: Integer; sz: PChar); far;
```

Parameters *pS* - the current symbol name.

See Also VCGetSymbolName, VCGetSymScale, VCGetSymRot, VCGetSymExplode

VCGetSymRot
VCSetSymRot

Version 1.2

Description Specifies the currently set symbol rotation.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetSymRot(short* iError);
extern "C" void WINAPI VCGetSymRotBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetSymRot(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetSymRotBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetSymRot Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetSymRotBP(var iError: Integer; var dRet: Double); far;
procedure VCSetSymRot(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the current symbol rotation in radians

Notes When placing symbol via the API, it may be necessary to adjust the symbol rotation before placement. If an external symbol interface is provided these also allow the interface to adjust the symbol rotation.

See Also VCGetSymName, VCGetSymbolName, VCGetSymScale, VCGetSymExplode

VCGetSymScale
VCSetSymScale

Version 1.2

Description Specifies the x and y symbol scale factor.

Declaration

C/C++:
```
extern "C" Point2D WINAPI VCGetSymScale(short* iError);
extern "C" void WINAPI VCGetSymScaleBP(short* iError, Point2D* pRet);
extern "C" void WINAPI VCSetSymScale(short* iError, Point2D p);
```

Visual Basic:
```
Declare Sub VCGetSymScaleBP Lib "VCMAIN32.DLL" (iError As Integer, pRet As Point2d)
Declare Sub VCSetSymScale Lib "VCMAIN32.DLL" (iError As Integer, p As Point2d)
```

Delphi:
```
procedure VCGetSymScaleBP(var iError: Integer; var pRet: Point2D); far;
procedure VCsetSymScale(var iError: Integer; dRet: Point2D); far;
```

Parameters *p* - the x and y values for the x and y scale

Notes When placing symbol via the API, it may be necessary to adjust the symbol scale before placement. If an external symbol interface is provided these also allow the interface to adjust the symbol scale.

See Also VCGetSymName, VCGetSymbolName, VCGetSymRot, VCGetSymExplode, VCGetSymScaleX, VCGetSymScaleY

VCGetSymScaleX
VCSetSymScaleX

Version 1.2

Description Specifies the x symbol scale factor.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetSymScaleX(short* iError);
extern "C" void WINAPI VCGetSymScaleXBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetSymScaleX(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetSymScaleXBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetSymScaleX Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetSymScaleXBP(var iError: Integer; var dRet: Double); far;
procedure VCSetSymScaleX(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the current x symbol scale factor.

Notes When placing symbol via the API, it may be necessary to adjust the symbol scale before placement. If an external symbol interface is provided these also allow the interface to adjust the symbol scale.

See Also VCGetSymName, VCGetSymbolName, VCGetSymScale, VCGetSymRot, VCGetSymExplode, VCGetSymScaleY

VCGetSymScaleY
VCSetSymScaleY

Version 1.2

Description Specifies the y symbol scale factor.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetSymScaleY(short* iError);
extern "C" void WINAPI VCGetSymScaleYBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetSymScaleY(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetSymScaleYBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetSymScaleY Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetSymScaleYBP(var iError: Integer; var dRet: Double); far;
procedure VCSetSymScaleY(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the current y symbol scale factor.

Notes When placing symbol via the API, it may be necessary to adjust the symbol scale before placement. If an external symbol interface is provided these also allow the interface to adjust the symbol scale.

See Also VCGetSymAutoExplode, VCGetSymName, VCGetSymbolName, VCGetSymScale, VCGetSymRot, VCGetSymExplode, VCGetSymScaleX

VCGetSymSnap
VCSetSymSnap

Version 1.2

Description Although symbol are considered a single entity, the ability to snap near point and closest point is still provided if desired via the symbol snap toggle. When on, the user can snap to entities within a symbol definition without exploding the symbol.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetSymSnap(short* iError);
extern "C" void WINAPI VCSetSymSnap(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Sub VCSetSymSnap Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
Declare Function VCGetSymSnap Lib "VCMAIN32.DLL" (iError As Integer) As Integer
```

Delphi:
```
function VCGetSymSnap(var iError: Integer):Integer; far;
procedure VCSetSymSnap(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetSymAutoExplode, VCGetSymName, VCGetSymbolName, VCGetSymScale, VCGetSymRot, VCGetSymExplode, VCGetSymScaleX, VCGetSymScaleY

VCGetSymbolDefCount

Version 1.2

Description Returns the number of symbol definitions in the current Corel Visual CADD session regardless of placements.

Declaration

C/C++: extern "C" short WINAPI VCGetSymbolDefCount(void);

Visual Basic: Declare Function VCGetSymbolDefCount Lib "VCMAIN32.DLL" () As Integer

Delphi: function VCGetSymbolDefCount:Integer; far;

Parameters *Returns* - the number of symbol definitions loaded.

Notes Often it is necessary to directly access each symbol, and to do this the application must know how many symbol definitions exist in order to parse through each definition.
VCGetSymbolDefCount will return how many definitions but not how many placements of each.

See Also VCGetSymbolDefEntityCount, VCGetSymbolIndex

VCGetSymbolDefEntityCount

Version 1.2

Description Retrieves the number of entities in a symbol definition.

Declaration

C/C++: extern "C" short WINAPI VCGetSymbolDefEntityCount(short i);

Visual Basic: Declare Function VCGetSymbolDefEntityCount Lib "VCMAIN32.DLL" (ByVal i As Integer) As Integer

Delphi: function VCGetSymbolDefEntityCount(i: Integer):Integer; far;

Parameters *Returns* - the number of entities.

Notes A symbol definition is simply a set of entities with their own properties that are defined as part of a symbol definition. This function will return the number of entities contained within the symbol definition whether for informational purposes or for an index to use in another procedure.

See Also VCGetSymbolDefCount, VCGetSymbolIndex, VCSetSymbolSection

VCGetSymbolIndex

Version 1.2

Description Returns an index of the specified symbol name for use in other symbol functions or subroutines.

Declaration

C/C++: extern "C" short WINAPI VCGetSymbolIndex(short* iError, char* pName);

Visual Basic: Declare Function VCGetSymbolIndex Lib "VCMAIN32.DLL" (iError As Integer, ByVal pName As String) As Integer

Delphi: function VCGetSymbolIndex(var iError: Integer; pName: PChar):Integer; far;

Parameters *pName* - the name of the symbol.
Returns - the symbol index number.

Notes Several subroutines use the symbol index in order to add entities to a symbol definition. These include VCAddLineEntity, VCAddCircleEntity, VCAddPointEntity as well as several others. In order to add these entities into a symbol definition and thus create a symbol from the external application, it is necessary to know the symbol index. VCGetSymbolIndex provides this. VCCreateSymbolDef must first be used to create a symbol definition to get a index or add entities, unless the symbol is loaded previously via Corel Visual CADD.

See Also VCGetSymbolDefCount, VCGetSymbolIndex, VCGetSymbolName, VCGetSymName, VCSaveVCS

VCGetSymbolInternalName

Version 1.2.1

Description Returns the internal name of a symbol name.

Declaration

C/C++: extern "C" short WINAPI VCGetSymbolInternalName(short* iError, char* pFileName, char* pReturn);

Visual Basic: Declare Function VCGetSymbolInternalName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pFileName As String, ByVal pReturn As String) As Integer

Delphi: function VCGetSymbolInternalName(var iError: Integer; pFileName: PChar; pReturn: PChar):Integer; far;

Parameters *pFileName* - the filename of the desired symbol.
pReturn - the returned internal symbol name.
Returns - the number of characters in pReturn.

Notes Although symbol filenames can only be eight characters long internal names can be larger. These internal names are stored in the symbol files and are what is displayed when selecting symbols to be placed within the drawing

See Also VCGetSymName, VCGetSymbolName

VCGetSymbolName

Version 1.2

Description Retrieves a symbol name from its symbol index.

Declaration

C/C++: extern "C" short WINAPI VCGetSymbolName(char* pName, short i);

Visual Basic: Declare Function VCGetSymbolName Lib "VCMAIN32.DLL" (ByVal pName As String, ByVal i As Integer) As Integer

Delphi: function VCGetSymbolName(pName: PChar; i: Integer):Integer; far;

Parameters *pName* - the name of the symbol.
i - the index number of the symbol.
Returns - the number of characters in the name string

Notes When parsing through a set of symbol definitions, the symbol name is not used by many of the Corel Visual CADD procedures. When an application is interfacing with people however, it is typically required that the user know the name of a symbol. VCGetSymbolName will retrieve this information. Also whenever a symbol is loaded or created in a Corel Visual CADD session it is indexed in order of creation or loading. This may be fine if an application loads symbols in a specific order, but it may be necessary to parse through the loaded symbols to retrieve the name and compare it with a symbol name used by the application.

See Also VCGetSymName, VCGetSymbolIndex

VCGetSymbolPlacementCount

Version 1.2

Description Returns the number of placements of the symbol definition in the current drawing.

Declaration

C/C++: extern "C" short WINAPI VCGetSymbolPlacementCount(short i);

Visual Basic: Declare Function VCGetSymbolPlacementCount Lib "VCMAIN32.DLL" (ByVal i As Integer) As Integer

Delphi: function VCGetSymbolPlacementCount(i: Integer):Integer; far;

Parameters *i* - the index number of the symbol.
Returns - an integer value of number of placements, 0 if none.

Notes Bill of Materials programs quite often need to count symbol placements. This allows a simple BOM by counting for instance the number of phones placed in a floor plan or number of IC's in a circuit board.

See Also VCGetSymbolName, VCGetSymName, VCGetSymbolIndex

VCGetSYSPath
VCSetSYSPath

Version 1.2

Description The default path for loading custom files.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetSYSPath(short* iError, char* szPath);
extern "C" void WINAPI VCSetSYSPath(short* iError, char* szPath);
```

Visual Basic:
```
Declare Function VCGetSYSPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
As Integer
Declare Sub VCSetSYSPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
```

Delphi:
```
function VCGetSYSPath(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetSYSPath(var iError: Integer; szPath: PChar); far;
```

Parameters *sz* - pth for the system files

Notes Corel Visual CADD reads several custom files on startup. These files range from linetype and hatch definitions to user defined scripts and menus. The system path should point to the location of these files or default values will be implemented.

See Also VCGetDWGPath, VCGetDXFPath, VCGetGCDPath, VCGetVCDPath, VCGetVCSPath, VCGetCMPPath, VCGetVCFPath, VCGetSCRPath

VCGetSystemHatchName

Version 1.2

Description Used to retrieve the pattern name from the system hatch file at corresponding input index.

Declaration

C/C++: extern "C" short WINAPI VCGetSystemHatchName(short* iError, char* pName, short i);

Visual Basic: Declare Function VCGetSystemHatchName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pName As String, ByVal i As Integer) As Integer

Delphi: function VCGetSystemHatchName(var iError: Integer; pName: PChar; i Integer):Integer; far;

Parameters *pName* - the returned string representing the hatch pattern.
i - the index for the hatch pattern defined in hatches.hat.
Returns - integer representing the length of the string.

Notes With Corel Visual CADD, the user can modify existing or create custom hatch patterns. All the hatch patterns are contained in the text file HATCHES.HAT. VCGetSystemHatchName is used to retrieve the hatch names defined in the text file.

See Also VCGetSystemHatchName, VCGetSystemHatchNameCount, on-line Help: Customizing Hatch Patterns

VCGetSystemHatchNameCount

Version	1.2
Description	Returns the number of hatch patterns defined in the system hatch file.
Declaration	
C/C++:	extern "C" short WINAPI VCGetSystemHatchNameCount(void);
Visual Basic:	Declare Function VCGetSystemHatchNameCount Lib "VCMAIN32.DLL" () As Integer
Delphi:	function VCGetSystemHatchNameCount:Integer; far;
Parameters	*Returns* - the number of defined hatch patterns.
Notes	With Corel Visual CADD, the user can modify existing or create custom hatch patterns. All the hatch patterns are contained in the text file HATCHES.HAT. VCGetSystemHatchNameCount returns a count of the valid patterns defined in this file.
See Also	VCGetSystemHatchName, On-Line Help: Customizing Hatch Patterns

VCGetTextAspect
VCSetTextAspect

Version 1.2

Description Specifies the current text aspect ratio setting. The text aspect ratio is the proportion of the text height to the text width.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetTextAspect(short* iError);
extern "C" void WINAPI VCGetTextAspectBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetTextAspect(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetTextAspectBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetTextAspect Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetTextAspectBP(var iError: Integer; var dRet: Double); far;
procedure VCSetTextAspect(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the current text aspect ratio.

See Also VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot, VCGetTextString, VCGetTextUnderline

VCGetTextBold
VCSetTextBold

Version 1.2

Description Specifies the bold display option for TT Fonts in Corel Visual CADD.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetTextBold(short* iError);
extern "C" void WINAPI VCSetTextBold(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetTextBold Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetTextBold Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetTextBold(var iError: Integer):Integer; far;
procedure VCSetTextBold(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetTextAspect, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot, VCGetTextString, VCGetTextUnderline

VCGetTextCharSpace
VCSetTextCharSpace

Version 1.2

Description Character spacing is the amount of space that appears between characters in a text string. It determines if the characters in a word are crowded or spread out. The value is a percentage of the characters height and applies only to vector fonts.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetTextCharSpace(short* iError);
extern "C" void WINAPI VCGetTextCharSpaceBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetTextCharSpace(short* iError, double dCharSpacing);
```

Visual Basic:
```
Declare Sub VCGetTextCharSpaceBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetTextCharSpace Lib "VCMAIN32.DLL" (iError As Integer, ByVal dCharSpacing As Double)
```

Delphi:
```
procedure VCGetTextCharSpaceBP(var iError: Integer; var dRet: Double); far;
procedure VCSetTextCharSpace(var iError: Integer; dCharSpacing: Double); far;
```

Parameters *dCharSpacing* - the charcter spacing as a decimal percentage (i.e. 1.5 is 150%)

See Also VCGetTextAspect, VCGetTextBold, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot, VCGetTextString, VCGetTextUnderline

VCGetTextColor
VCSetTextColor

Version 1.2

Description Specifies the current color setting for subsequent text placements. Text and dimensions have their own color and layer settings and are not affected by VCSetColorIndex.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetTextColor(short* iError);
extern "C" void WINAPI VCSetTextColor(short* iError, short i);
```

Visual Basic:
```
Declare Sub VCSetTextColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer)
Declare Function VCGetTextColor Lib "VCMAIN32.DLL" (iError As Integer) As Integer
```

Delphi:
```
function VCGetTextColor(var iError: Integer):Integer; far;
procedure VCSetTextColor(var iError: Integer; i: Integer); far;
```

Parameters *i* - the current text color index.

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot, VCGetTextString, VCGetTextUnderline, VCGetDimItemColor, VCGetColorIndex

VCGetTextFontName
VCSetTextFontName

Version 1.2

Description The name of the font to be used for all for all subsequent text placements.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetTextFontName(short* iError, char* pS);
extern "C" void WINAPI VCSetTextFontName(short* iError, char* sz);
```

Visual Basic:
```
Declare Function VCGetTextFontName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String)
As Integer
Declare Sub VCSetTextFontName Lib "VCMAIN32.DLL" (iError As Integer, ByVal sz As String)
```

Delphi:
```
function VCGetTextFontName(var iError: Integer; pS: PChar):Integer; far;
procedure VCSetTextFontName(var iError: Integer; sz: PChar); far
```

Parameters *pS* - the name of the current font.

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot, VCGetTextString, VCGetTextUnderline

VCGetTextHeight
VCSetTextHeight

Version 1.2

Description Unlike most other Windows programs, Corel Visual CADD measures text height in real world units, specifically inches, instead of points.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetTextHeight(short* iError);
extern "C" void WINAPI VCGetTextHeightBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetTextHeight(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetTextHeightBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetTextHeight Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetTextHeightBP(var iError: Integer; var dRet: Double); far;
procedure VCSetTextHeight(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the text height.

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot, VCGetTextString, VCGetTextUnderline

VCGetTextItalic
VCSetTextItalic

Version 1.2

Description Specifies the italic display option for TT Fonts in Corel Visual CADD.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetTextItalic(short* iError);
extern "C" void WINAPI VCSetTextItalic(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetTextItalic Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetTextItalic Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetTextItalic(var iError: Integer):Integer; far;
procedure VCSetTextItalic(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetTextItalic, VCGetTextBold, VCGetTextItalic, VCGetTextBold

VCGetTextItalicValue
VCSetTextItalicValue

Version 1.2

Description Vector fonts can be slanted to emulate italics.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetTextItalicValue(short* iError, double* dI);
extern "C" void WINAPI VCGetTextItalicValueBP(short* iError, double* dI);
extern "C" void WINAPI VCSetTextItalicValue(short* iError, double dI);
```

Visual Basic:
```
Declare Sub VCGetTextItalicValueBP Lib "VCMAIN32.DLL" (iError As Integer, dI As Double)
Declare Sub VCSetTextItalicValue Lib "VCMAIN32.DLL" (iError As Integer, ByVal dI As Double)
```

Delphi:
```
procedure VCGetTextItalicValueBP(var iError: Integer; var dI: Double); far;
procedure VCSetTextItalicValue(var iError: Integer; dI: Double); far;
```

Parameters *dI* - the angle in radians for the slant

Notes The number must range between 45 and -45 degrees. As with all angle functions, the angle is specified in radians. A negative number will slant the text backwards.

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot, VCGetTextString, VCGetTextUnderline

VCGetTextJustify
VCSetTextJustify

Version 1.2

Description The text justification setting. Text can be justified left, right or centered horizontally relative to the placement point.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetTextJustify(short* iError);
extern "C" void WINAPI VCSetTextJustify(short* iError, short j);
```

Visual Basic:
```
Declare Function VCGetTextJustify Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetTextJustify Lib "VCMAIN32.DLL" (iError As Integer, ByVal j As Integer)
```

Delphi:
```
function VCGetTextJustify(var iError: Integer):Integer; far;
procedure VCSetTextJustify(var iError: Integer; j: Integer); far;
```

Parameters *j* - the ASCII equivalent for the following characters.
‘C'- center.
‘L' - left.
‘R' - right.

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot, VCGetTextString, VCGetTextUnderline

VCGetTextLayer
VCSetTextLayer

Version 1.2

Description Text can be placed on a separate layer independent of the current layer.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetTextLayer(short* iError);
extern "C" void WINAPI VCSetTextLayer(short* iError, short iTextLayer);
```

Visual Basic:
```
Declare Function VCGetTextLayer Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetTextLayer Lib "VCMAIN32.DLL" (iError As Integer, ByVal iTextLayer As Integer)
```

Delphi:
```
function VCGetTextLayer(var iError: Integer):Integer; far;
procedure VCSetTextLayer(var iError: Integer; iTextLayer: Integer); far;
```

Parameters *iTextLayer* - layer index setting from 0 to 1023

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot, VCGetTextString, VCGetTextUnderline

VCGetTextLineSpace
VCSetTextLineSpace

Version 1.2

Description The between text line VCGetDimTextLineSpace spacing as a percentage of current text height.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetTextLineSpace(short* iError);
extern "C" void WINAPI VCGetTextLineSpaceBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetTextLineSpace(short* iError, double dLineSpacing);
```

Visual Basic:
```
Declare Sub VCGetTextLineSpaceBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetTextLineSpace Lib "VCMAIN32.DLL" (iError As Integer, ByVal dLineSpacing As Double)
```

Delphi:
```
procedure VCGetTextLineSpaceBP(var iError: Integer; var dRet: Double); far;
procedure VCSetTextLineSpace(var iError: Integer; dLineSpacing: Double); far;
```

Parameters *dRet* - spacing between the lines.

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextProSpacing, VCGetTextRot, VCGetTextString, VCGetTextUnderline

VCGetTextProSpacing
VCSetTextProSpacing

Version 1.2

Description Vector text character spacing can be forced to monospace or proportional spacing. Monospace is a characteristic of typewriter output and all characters will use the same amount of space regardless of their width and height.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetTextProSpacing(short* iError);
extern "C" void WINAPI VCSetTextProSpacing(short* iError, BOOL b);
```

Visual Basic:
```
Declare Function VCGetTextProSpacing Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetTextProSpacing Lib "VCMAIN32.DLL" (iError As Integer, ByVal b As Integer)
```

Delphi:
```
function VCGetTextProSpacing(var iError: Integer):Boolean; far; external'VCMAIN';
procedure VCSetTextProSpacing(var iError: Integer; b: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextRot, VCGetTextString, VCGetTextUnderline

VCGetTextRot
VCSetTextRot

Version 1.2

Description The current text angle setting for font placement.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetTextRot(short* iError);
extern "C" void WINAPI VCGetTextRotBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetTextRot(short* iError, double d);
```

Visual Basic:
```
Declare Sub VCGetTextRotBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetTextRot Lib "VCMAIN32.DLL" (iError As Integer, ByVal d As Double)
```

Delphi:
```
procedure VCGetTextRotBP(var iError: Integer; var dRet: Double); far;
procedure VCSetTextRot(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the angle setting in radians

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextString, VCGetTextUnderline

VCGetTextString
VCSetTextString

Version 1.2

Description When placing text via the API, the user interface is not available to enter the required line of text, therefore it is necessary to set the text string before creating the placement.

Declaration

C/C++:
```
extern "C" short FAR WINAPI VCGetTextString(short* iError, char* pS);
extern "C" void FAR WINAPI VCSetTextString(short* iErroR, char* pS);
```

Visual Basic:
```
Declare Function VCGetTextString Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String) As Integer
Declare Sub VCSetTextString Lib "VCMAIN32.DLL" ( iError As Integer, ByVal pS As String )
```

Delphi:
```
function VCGetTextString(var iError: Integer; pS: PChar):Integer; far;
procedure VCSetTextString(var iError: Integer; pS: PChar); far;
```

Parameters *s* - the text string.

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot, VCGetTextUnderline

VCGetTextUnderline
VCSetTextUnderline

Version 1.2

Description Specifies the underline display option for TT Fonts in Corel Visual CADD.

Declaration

C/C++:
```
extern "C" BOOL WINAPI VCGetTextUnderline(short* iError);
extern "C" void WINAPI VCSetTextUnderline(short* iError, BOOL tf);
```

Visual Basic:
```
Declare Function VCGetTextUnderline Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetTextUnderline Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi:
```
function VCGetTextUnderline(var iError: Integer):Boolean; far;
procedure VCSetTextUnderline(var iError: Integer; tf: Boolean); far;
```

Parameters *tf* - toggle setting
0 - Off (Unchecked)
1- On(Checked)

See Also VCGetTextAspect, VCGetTextBold, VCGetTextCharSpace, VCGetTextColor, VCGetTextFontName, VCGetTextHeight, VCGetTextItalic, VCGetTextItalicValue, VCGetTextJustify, VCGetTextLayer, VCGetTextLineSpace, VCGetTextProSpacing, VCGetTextRot

VCGetToolID

Version	1.2
Description	Returns the tool id for the currently active tool or command.
Declaration	
C/C++:	extern "C" WORD WINAPI VCGetToolID();
Visual Basic:	Declare Function VCGetToolID Lib "VCTOOL32.DLL" () As Integer
Delphi:	function VCGetToolID:Integer; far;
Parameters	No additional parameters are used with this subroutine.
Notes	As it may be necessary to display the two-letter or native commands with each tool in a custom interface, VCGetToolID is provided in order to retrieve the id to pass to these functions.
See Also	Tool ID

VCGetUnitConversionFactor

Version 1.2

Description Returns the conversion factor used by Corel Visual CADD to convert from the "inch" database to the current unit setting.

Declaration

C/C++: extern "C" void WINAPI VCGetUnitConversionFactor(short* iError, double* dRet);

Visual Basic: Declare Sub VCGetUnitConversionFactor Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)

Delphi: procedure VCGetUnitConversionFactor(var iError: Integer; var dRet: Double);

Parameters *d* - the multiplier used to arrive at the current unit settings.

Notes Since all data is currently stored in the Corel Visual CADD drawing database as inches, it is necessary to format any distances or areas in the units currently set in the program. This will return a simple multiplier which will enable the conversion without having to case out each unit conversion in code.

See Also VCGetDisplayDistFormat

VCGetUseByLayerProperties
VCSetUseByLayerProperties

Version 2.0

Description Determines if a layer is using layer properties for entities drawn on that layer.

Declaration

C/C++

```
extern "C" void WINAPI VCGetUseByLayerProperties(short* iError, vbool* tfColor, vbool* tfLineType, vbool* tfLineWidth);
extern "C" void WINAPI VCSetUseByLayerProperties(short* iError, vbool tfColor, vbool tfLineType, vbool tfLineWidth);
```

Visual Basic

```
Declare Sub VCGetUseByLayerProperties Lib "VCMAIN32.DLL" (iError As Integer, tfColor As Integer, tfLineType As Integer, tfLineWidth As Integer)
Declare Sub VCSetUseByLayerProperties Lib "VCMAIN32.DLL" (iError As Integer, ByVal tfColor As Integer, ByVal tfLineType As Integer, ByVal tfLineWidth As Integer)
```

Delphi

```
procedure VCGetUseByLayerProperties(var iError: Integer; var tfColor: Boolean; var tfLineType: Boolean; var tfLineWidth: Boolean); far;
procedure VCSetUseByLayerProperties(var iError: Integer; tfColor: Boolean; tfLineType: Boolean; tfLineWidth: Boolean); far;
```

Parameters

tfColor - flag indicating if color is used in Layer Properties.
0 - color is not used.
1 - color is used.
tfLineType - flag indicating if line type is used in Layer Properties.
0 - line typeis not used.
1 - line typeis used.
tfLineWidth - flag indicating if line width is used in Layer Properties.
0 - line widthis not used.
1 - line widthis used.

Notes Layer properties were introduced into v2.0.1 allowing properties to be assigned by layer rather than by entity. For example, a layer can be set so all entities drawn on the layer will be a specific color, line type and line width. This will override the current properties settings when active. VCGetUseByLayerProperties is used to determine if the layer has active property settings while VCSetUseByLayerProperties allows an application to choose which properties to use. VCSetLayerProperties will set the values for the layer and VCClearLayerProperties turns the capability off and clears all associated values. It is important to keep track of the state of layer properties when modifying entities in the drawing. For example, if you set the color index using VCSetColorIndex but the layer properties are enabled the proper color may not get applied. Therefore when attempting to control the properties of entities as they are placed it is imperative that the application monitor the setting for by layer control as the information is being supplied by the API.

See Also VCClearLayerProperties, VCLayerHasProperties

VCGetUseFileLocking
VCSetUseFileLocking

Version 2.0

Description Locks a file for read only mode by users other than the current.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetUseFileLocking(short* iError);
extern "C" void WINAPI VCSetUseFileLocking(short* iError, vbool tf);
```

Visual Basic
```
Declare Function VCGetUseFileLocking Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetUseFileLocking Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetUseFileLocking(var iError: Integer):Boolean; far;
function VCSetUseFileLocking(var iError: Integer, tf:Boolean):Boolean; far;
```

Parameters *tf* - flag for setting file locking.
0 - do not lock files.
1 - lock files.

Notes Locked files can not be modified by another Corel Visual CADD user on a network until the drawing is saved or closed. Other users can only open, view and copy the drawing. The user name is taken form the registered user name stored in the registry for the installed machine.

See Also VCIsFileLockedByCurrentUser, VCLockFile

VCGetUseHPGL2
VCSetUseHPGL2

Version 2.0

Description Enables the use of HPGL/2 optimization for output vector devices.

Declaration

C/C++
```
extern "C" vbool WINAPI VCGetUseHPGL2(short* iError);
extern "C" void WINAPI VCSetUseHPGL2(short* iError, vbool tf);
```

Visual Basic
```
Declare Function VCGetUseHPGL2 Lib "VCDLG32.DLL" (iError As Integer) As Integer
Declare Sub VCSetUseHPGL2 Lib "VCDLG32.DLL" (iError As Integer, ByVal tf As Integer)
```

Delphi
```
function VCGetUseHPGL2(var iError: Integer):Boolean; far;
procedure VCSetUseHPGL2(var iError: Integer; tf: Boolean); far;
```

Parameters tf - toggle indicasting whether to use HPGL/2 optimization
0 - do not use the optimization
1 - use the optimization

Notes Using HPGL/2 optimization when outputting to a vector plotter will improve the quality of arcs and circles and decrease plot time if the plotter supports HPGL/2 graphics language. If this option is used then an Init String for the language must be provided to tell the plotter to recognize the HPGL/2 commands.

See Also VCApplyPlotterLanguageDefaults, VCApplyPlotterPenMapDefaults

VCGetUserDataName
VCSetUserDataName

Version 1.2

Description User Data requires an application name to define a storage segment for the attached data.

Declaration

C/C++: extern "C" short WINAPI VCGetUserDataName(short* iError, char* pS);
extern "C" void WINAPI VCSetUserDataName(short* iError, char* pS);

Visual Basic: Declare Function VCGetUserDataName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String) As Integer
Declare Sub VCSetUserDataName Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String)

Delphi: function VCGetUserDataName(var iError: Integer; pS: PChar):Integer; far;
procedure VCSetUserDataName(var iError: Integer; pS: PChar); far;

Parameters *S* - the name returned as the current registered user data name.

Notes This name must be used in order to retrieve or edit the attached data. Once a name has been set using VCSetUserDataName, the corresponding data can only be retrieved using the current User Data name. This prevent unauthorized use of another applications User Data and accidental misuse or editing of another applications User Data. VCGetUserDataName returns the currently active User Data segment.

See Also VCAddCurrentEntityUserDataByte, VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort

VCGetUserToolLBDown

Version 1.2

Description Retrieves the last point selected within the drawing area. Used with VCSetAlertApp.

Declaration

C/C++: extern "C" void WINAPI VCGetUserToolLBDown(short* iError, Point2D* dpP);

Visual Basic: Declare Sub VCGetUserToolLBDown Lib "VCMAIN32.DLL" (iError As Integer, dpP As Point2D)

Delphi: procedure VCGetUserToolLBDown(var iError: Integer; var dpP: Point2D); far;

Parameters *dpP* - set to reflect the last drawing coordinates picked by the user.

Notes To initialize Windows messaging between Corel Visual CADD and an external application, the hWnd of some control or object must be sent to Visual CADD using VCSetAlertApp. When registering the hWnd a code must also be included which specifies which messages an application will receive. These can be added together to get multiple messages. For example iCode of 12 would specify that the command line characters and abort messages would be sent. To handle these messages, an application code must have code specifically to handle a Windows message sent to the control whose hWnd is registered with VCSetAlertApp. In Visual BASIC, handle this by supplying code in the mouse down event for the control specified for each mouse down message sent by Visual CADD. Corel Visual CADD is fairly intelligent about when to send this message and only send the message when a drawing point has been selected. This means that the user can issue snaps or use tracking without invoking the application code for the mouse down event. To retrieve the point the user selected in the drawing area, use VCGetUserToolLBDown which sets a Point2D of the last point picked. When trapping the user input, register the control with an iCode of either 0 (all messages) or 8 (mouse down messages) and add code to the control for key press. When the key press code is activated by the message from Corel Visual CADD, use VCGetCmdStr to retrieve the last key press from Corel Visual CADD. Once the key press has been determined through code can act according to process the information or send it back for Corel Visual CADD to use with VCSetCmdStr. Once the application has completed with the messaging, use VCClearAlertApp to remove an application from the messaging registry.

See Also VCSetAlertApp, VCSetCmdStr, VCGetCmdStr, VCSetAlertApp

VCGetUserToolLBUp

Version 2.0.1

Description Returns a left button up message to a user tool.

Declaration

C/C++: extern "C" void WINAPI VCGetUserToolLBUp(short* iError, Point2D* dpP);

Visual Basic: Declare Sub VCGetUserToolLBUp Lib "VCMAIN32.DLL" (iError As Integer, dpP As Point2D)

Delphi: procedure VCGetUserToolLBUp(var iError: Integer; var dpP: Point2D); far;

Parameters *dpP* - set to reflect the last drawing coordinates picked by the user.

Notes To initialize Windows messaging between Corel Visual CADD and an external application, the hWnd of some control or object must be sent to Corel Visual CADD using VCSetAlertApp. When registering the hWnd a code must also be included which specifies which messages an application will receive. These can be added together to get multiple messages. For example iCode of 12 would specify that the command line characters and abort messages would be sent. To handle these messages, an application code must have code specifically to handle a Windows message sent to the control whose hWnd is registered with VCSetAlertApp. In Visual BASIC, handle this by supplying code in the mouse down event for the control specified for each mouse down message sent by Visual CADD. Corel Visual CADD is fairly intelligent about when to send this message and only send the message when a drawing point has been selected. This means that the user can issue snaps or use tracking without invoking the application code for the mouse down event. To retrieve the point the user selected in the drawing area, use VCGetUserToolLBDown which sets a Point2D of the last point picked. When trapping the user input, register the control with an iCode of either 0 (all messages) or 8 (mouse down messages) and add code to the control for key press. When the key press code is activated by the message from Corel Visual CADD, use VCGetCmdStr to retrieve the last key press from Corel Visual CADD. Once the key press has been determined through code can act according to process the information or send it back for Corel Visual CADD to use with VCSetCmdStr. Once the application has completed with the messaging, use VCClearAlertApp to remove an application from the messaging registry.

See Also VCSetAlertApp, VCSetCmdStr, VCGetCmdStr, VCSetAlertApp,VCLButtonUpTimerReset

VCGetUserToolMouseMove

Version 1.2

Description Retrieves the position the user has moved the mouse to within the drawing area. Used with VCSetAlertApp.

Declaration

C/C++: extern "C" void WINAPI VCGetUserToolMouseMove(short* iError, Point2D* dpP);

Visual Basic: Declare Sub VCGetUserToolMouseMove Lib "VCMAIN32.DLL" (iError As Integer, dpP As Point2D)

Delphi: procedure VCGetUserToolMouseMove(var iError: Integer; var dpP: Point2D); far;

Parameters *dpP* - set to reflect the last drawing coordinates picked by the user.

Notes Once mouse move messaging has been established with VCSetAlertApp, VCGetUserToolMouseMove allows each mouse movement to be retrieved from Corel Visual CADD. For example, in Visual BASIC, the hWnd for the main form can be passed to VCSetAlertApp with 0 as the iCode. Code can be added to the mouse move event of the form. Each time the mouse is moved in the Corel Visual CADD drawing area, a windows message will be sent to the form which will activate the form1_mousemove subroutine. In this subroutine, VCGetUserToolMouseMove can be used to retrieve the point that the mouse last moved over and code can be executed when the mouse passes over a certain region of the drawing. Be aware that processing additional code in an external application can require a great deal of processor overhead. Make sure that this is truly necessary before building code that uses this message.

See Also VCSetAlertApp, VCClearAlertApp, VCGetUserToolLBDown

VCGetVCDPath
VCSetVCDPath

Version 1.2

Description The default file path for opening and saving Corel Visual CADD VCD drawing files.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetVCDPath(short* iError, char* szPath);
extern "C" void WINAPI VCSetVCDPath(short* iError, char* szPath);
```

Visual Basic:
```
Declare Function VCGetVCDPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
As Integer
Declare Sub VCSetVCDPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
```

Delphi:
```
function VCGetVCDPath(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetVCDPath(var iError: Integer; szPath: PChar); far;
```

Parameters *szPath* - the file path

See Also VCGetDWGPath, VCGetDXFPath, VCGetGCDPath, VCGetSYSPath, VCGetVCSPath, VCGetCMPPath, VCGetVCFPath

VCGetVCFPath
VCSetVCFPath

Version 1.2

Description The default file path for opening and saving Corel Visual CADD font files.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetVCFPath(short* iError, char* szPath);
extern "C" void WINAPI VCSetVCFPath(short* iError, char* szPath);
```

Visual Basic:
```
Declare Function VCGetVCFPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
As Integer
Declare Sub VCSetVCFPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
```

Delphi:
```
function VCGetVCFPath(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetVCFPath(var iError: Integer; szPath: PChar); far;
```

Parameters *szPath* - the file path

See Also VCGetDWGPath, VCGetDXFPath, VCGetGCDPath, VCGetSYSPath, VCGetVCDPath, VCGetCMPPath, VCGetVCFPath

VCGetVCSPath
VCSetVCSPath

Version 1.2

Description The default file path for opening and saving Corel Visual CADD symbol files.

Declaration

C/C++:
```
extern "C" short WINAPI VCGetVCSPath(short* iError, char* szPath);
extern "C" void WINAPI VCSetVCSPath(short* iError, char* szPath);
```

Visual Basic:
```
Declare Function VCGetVCSPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
As Integer
Declare Sub VCSetVCSPath Lib "VCMAIN32.DLL" (iError As Integer, ByVal szPath As String)
```

Delphi:
```
function VCGetVCSPath(var iError: Integer; szPath: PChar):Integer; far;
procedure VCSetVCSPath(var iError: Integer; szPath: PChar); far;
```

Parameters *Path* - string returned containing the current symbol path.

See Also VCGetDWGPath, VCGetDXFPath, VCGetGCDPath, VCGetSYSPath, VCGetVCDPath, VCGetVCSPath, VCGetCMPPath, VCGetVCFPath

VCGetVidTolerance
VCSetVidTolerance

Version 1.2

Description Sets the maximum distance in on-screen inches the cursor may be from an object for Corel Visual CADD to snap or select it.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetVideoTolerance(short* iError);
extern "C" void WINAPI VCGetVideoToleranceBP(short* iError, double* dRet);
extern "C" void WINAPI VCSetVidTolerance(short* iError, double dRet);
```

Visual Basic:
```
procedure VCGetVideoToleranceBP(var iError: Integer; var dRet: Double); far
Declare Sub VCSetVidTolerance Lib "VCMAIN32.DLL" (iError As Integer, ByVal dRet As Double
```

Delphi:
```
procedure VCGetVideoToleranceBP(var iError: Integer; var dRet: Double); far;
procedure VCSetVidTolerance(var iError: Integer; dRet: Double); far
```

Parameters *dRet* - the distance to search.

See Also Tool Reference

VCGetViewCount

Version 2.0

Description Returns the number of viewports for the input drawing world.

Declaration

C/C++ extern "C" vbool WINAPI VCGetViewCount(short* iError, WORLDHANDLE hW, short* iVCnt);

Visual Basic Declare Function VCGetViewCount Lib "VCMAIN32.DLL" (iError As Integer, ByVal hW As Long, iVCnt As Integer) As Integer

Delphi function VCGetViewCount(var iError: Integer; hW: Longint; var iVCnt:

Parameters *hw* - the WORLDHANDLE for the drawing.
IVCnt - the number of viewports for the drawing.
returns - a flag indicating if the world has multiple viewports open.

Notes Corel Visual CADD allows for multiple views of a drawing. Each of these views is placed into a separate MDI Window within the Visual CADD frame. The view can be changed by moving to the Window containing the desired view.

See Also VCNewView, VCFirstView , VCNextView,VCChangeView

VCGetWallWidth1
VCSetWallWidth1

Version 1.2

Description Sets the offset, relative to the cursor movement, of the left line for the double line tool.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetWallWidth1(short* iError);
extern "C" void WINAPI VCGetWallWidth1BP(short* iError, double* dRet);
extern "C" void WINAPI VCSetWallWidth1(short* iError, double d1);
```

Visual Basic:
```
Declare Sub VCGetWallWidth1BP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetWallWidth1 Lib "VCMAIN32.DLL" (iError As Integer, ByVal d1 As Double)
```

Delphi:
```
procedure VCGetWallWidth1BP(var iError: Integer; var dRet: Double); far;
procedure VCSetWallWidth1(var iError: Integer; d1: Double); far;
```

Parameters *dRet* - the distance.

See Also VCGetWallWidth2

VCGetWallWidth2
VCSetWallWidth2

Version 1.2

Description Sets the offset, relative to the cursor movement, of the right line for the double line tool.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetWallWidth2(short* iError);
extern "C" void WINAPI VCGetWallWidth2BP(short* iError, double* dRet);
extern "C" void WINAPI VCSetWallWidth2(short* iError, double d1);
```

Visual Basic:
```
Declare Sub VCGetWallWidth2BP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetWallWidth2 Lib "VCMAIN32.DLL" (iError As Integer, ByVal d1 As Double)
```

Delphi:
```
procedure VCGetWallWidth2BP(var iError: Integer; var dRet: Double); far;
procedure VCSetWallWidth2(var iError: Integer; d1: Double); far;
```

Pameters *dRet* - the distance.

See Also VCGetWallWidth1

VCGetWorldByHWND

Version	2.0
Description	Retrieves the WORLDHANDLE for a drawing from the displaying Window.
Declaration	
C/C++	extern "C" WORLDHANDLE WINAPI VCGetWorldByHWND(short* iError, long hwnd);
Visual Basic	Declare Function VCGetWorldByHWND Lib "VCMAIN32.DLL" (iError As Integer, ByVal hWnd As Long) As Long
Delphi	function VCGetWorldByHWND(var iError: Integer; hwnd: Longint):Longint; far;
Parameters	hWnd - the HWND for the window containg the drawing. HW - the returned WORLDHANDLE.
Notes	Drawing world are referenced by an internal 0 based WORLDHANDLE index or a windows HWND for the control displaying the drawing. Typically, the API utilizes the internal WORLDHANDLE when referencing the drawing. VCGetWorldIndexByHWND is used to retrieve the internal index from a Windows HWND.
See Also	VCChangeView, VCGetCurrWorld, VCGetWorldIndexByHWND

VCGetWorldExtents

Version	2.0
Description	Retrieves the drawing extents for the active drawing.
Declaration	
C/C++	extern "C" void WINAPI VCGetWorldExtents(short* iError, Point2D* dpMin, Point2D* dpMax);
Visual Basic	Declare Sub VCGetWorldExtents Lib "VCMAIN32.DLL" (iError As Integer, dpMin As Point2D, dpMax As Point2D)
Delphi	procedure VCGetWorldExtents(var iError: Integer; var dpMin: Point2D; var
Parameters	*dpMin* - the Point2D structure containing the lower left corner of a bounding rectangle. *dpMax* - the Point2D structure containing the upper right corner of a bounding rectangle.
Notes	The drawing extents reflect a bounding box placed around the entire drawing. Corel Visual CADD displays the information to the user for reference within the file.
See Also	VCGetWorldSize, VCEntityExtents

VCGetWorldIndexByHWND

Version	2.0
Description	Retrieves the WORLDHANDLE for a drawing from the displaying Window.
Declaration	
C/C++	extern "C" void WINAPI VCGetWorldIndexByHWND(short* iError, long hWnd, WORLDHANDLE* hW);
Visual Basic	Declare Sub VCGetWorldIndexByHWND Lib "VCMAIN32.DLL" (iError As Integer, ByVal hWnd As Long, hW As Long)
Delphi	procedure VCGetWorldIndexByHWND(var iError: Integer; hWnd: Longint; var hW: Longint); far;
Parameters	hWnd - the HWND for the window containg the drawing. HW - the returned WORLDHANDLE.
Notes	Drawing world are referenced by an internal 0 based WORLDHANDLE index or a windows HWND for the control displaying the drawing. Typically, the API utilizes the internal WORLDHANDLE when referencing the drawing. VCGetWorldIndexByHWND is used to retrieve the internal index from a Windows HWND.
See Also	VCChangeView, VCGetCurrWorld, VCGetWorldByHWND

VCGetXYHandle
VCSetXYHandle

Version 2.0

Description The XY handle displays the cursor coordinates in the interface.

Declaration

C/C++
```
extern "C" long WINAPI VCGetXYHandle();
extern "C" void WINAPI VCSetXYHandle(HWND hWnd);
```

Visual Basic
```
Declare Function VCGetXYHandle Lib "VCMAIN32.DLL" () As Long
Declare Sub VCSetXYHandle Lib "VCMAIN32.DLL" (ByVal hWnd As Integer)
```

Delphi
```
function VCGetXYHandle:Longint; far;
procedure VCSetXYHandle(hWnd: Integer); far;
```

Parameters *hWnd* - the HWND handle for the object to be used as the message area.

Notes Like VCSetMessageHandle, VCSetXYHandle sets a message handle of a Windows object to display a text message. However in this case the message is x and y coordinates of the current cursor position as related to the current manual entry mode. This also reflects the current units and decimal or fractional settings. This is normally displayed in the status bar at the bottom of the Corel Visual CADD screen.

See Also VCSetAngleHandle, VCSetDistanceHandle

VCGetZoomFactor
VCSetZoomFactor

Version 1.2

Description The multiplier to used to change the drawing magnification when the Zoom In command is used. The factor for the Zoom Out command is the reciprocal of this.

Declaration

C/C++:
```
extern "C" double WINAPI VCGetZoomFactor(short* iError);
extern "C" void WINAPI VCSetZoomFactor(short* iError, double dRet);
```

Visual Basic:
```
Declare Sub VCGetZoomFactorBP Lib "VCMAIN32.DLL" (iError As Integer, dRet As Double)
Declare Sub VCSetZoomFactor Lib "VCMAIN32.DLL" (iError As Integer, ByVal dRet As Double)
```

Delphi:
```
procedure VCGetZoomFactorBP(var iError: Integer; var dRet: Double); far;
procedure VCSetZoomFactor(var iError: Integer; dRet: Double); far;
```

Parameters *dRet* - the zoom factor

See Also VCGetAskZoomCenter

VCIncrementWidthOnAllEntities

Version 1.2

Description Changes the line width for all entities in the drawing database a specified amount.

Declaration

C/C++: extern "C" void WINAPI VCIncrementWidthOnAllEntities(short* iError, short iIncrement);

Visual Basic: Declare Sub VCIncrementWidthOnAllEntities Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIncrement As Integer)

Delphi: procedure VCIncrementWidthOnAllEntities(var iError: Integer; iIncrement Integer); far;

Parameters *iIncrement* - the value to increment.

Notes Several utility routines to accomplish specifics tasks are available directly in the API. Instead of parsing the database for each entity and then resetting the line width, this routine will automatically force the line width to an input value. When outputting to certain printers it is desirable to increase the line width in order to improve the output quality. VCForceWidthOnAllEntities and VCIncrementWidthOnAllEntities facilitate this operation under a single routine.

See Also VCForceWidthOnAllEntities

VCInit

Version	1.2
Description	Initializes the Corel Visual CADD DLLs so they may be used by another application.
Declaration	
C/C++:	extern "C" void WINAPI VCInit(void);
Visual Basic:	Declare Sub VCInit Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCInitDialogs; far;
Parameters	No parameters are used for this subroutine.
Notes	Whenever the CADD drawing is to be loaded independent of the Corel Visual CADD program itself, the DLL's must be initialized in order set up a drawing database and establish all the drawing settings. This allows the program to access the internal subroutines and functions and to display the drawing in a Visual BASIC picture box, or similar drawing area. When completed with a Corel Visual CADD session be sure to end it with a VCTerminate. VCGetInitCount will return the number of instances of current Corel Visual CADD sessions.
See Also	VCGetInitCount, VCTerminate, VCTerminate, VCPaint, VCGetInitCount

VCInitDialogs

Version 1.2

Description Initializes the Corel Visual CADD dialogs so they may be used by an external application.

Declaration

C/C++: extern "C" void WINAPI VCInitDialogs(void);

Visual Basic: Declare Sub VCInitDialogs Lib "VCDLG32.DLL" ()

Delphi: procedure VCInitDialogs; far;

Parameters No parameters are used with this subroutine.

Notes When building an external application based on the Corel Visual CADD engine, it may, or may not, be desirable to display Corel Visual CADD's internal dialogs. If the external application uses it's own dialogs and passes the values or settings to Corel Visual CADD manually than it probably will not be necessary to use the internal dialogs. If however the external application requires the internal dialogs for consistency, or ease of programming, VCInitDialogs will initialize the dialogs for use while VCTerminateDialogs will terminate their use.

See Also VCTerminateDialogs

VCInitPrintMode

Version 2.0

Description Initializes the print routines for use outside the Corel Visual CADD interface.

Declaration

C/C++ extern "C" void WINAPI VCInitPrintMode(short* iError, short iPrintMode);

Visual Basic Declare Sub VCInitPrintMode Lib "VCDLG32.DLL" (iError As Integer, ByVal iPrintMode As Integer)

Delphi procedure VCInitPrintMode(var iError: Integer; iPrintMode: Integer); far;

Parameters *iPrintMode* - which mode to initialize.
0 - PRINTMODE
1 - PLOTMODE

Notes When creating a custom interface that utilizes the Corel Visual CADD print routines, an application must initialize the mode on start and terminate it on close. The API provides access to the both the print and plot dialogs in which Corel Visual CADD handles all the output as if it were part of the interface by simply displaying the built in dialogs. The second method allows the application to create all the command and bypass the Corel Visual CADD interface. When using the first dialog method simply use VCInitDialogs and VCTerminateDialogs. When using the second method the initialization is handled by VCInitPrintMode and the de-initialization is handled by VCDeInitPrintMode.

See Also VCDeInitPrintMode

VCInvalidateRect

Version	1.2
Description	Sets a flag for Corel Visual CADD that tells the system to redraw the drawing window.
Declaration	
C/C++:	extern "C" void WINAPI VCInvalidateRect();
Visual Basic:	Declare Sub VCInvalidateRect Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCInvalidateRect; far;
Parameters	No parameters are used for this subroutine.
Notes	VCInvalidateRect is analogous to the Windows API call InvalidateRect, except that the rect is the entire drawing area, hWnd is assumed to be the current drawing window, and erase background is assumed to be true. When a WM_PAINT message is processed by Windows, the rect will be redrawn.
See Also	Windows 3.1 SDK - InvalidateRect, VCPaint, VCPaintWorld

VCIsAnythingSelected

Version 1.2

Description Returns a value to determine if anything in the current drawing is selected.

Declaration

C/C++: extern "C" vbool WINAPI VCIsAnythingSelected(void);

Visual Basic: Declare Function VCIsAnythingSelected Lib "VCMAIN32.DLL" () As Integer

Delphi: function VCIsAnythingSelected:Boolean; far;

Parameters *Returns* - an integer value representing true or false.
0 is false.
1 is true.

Notes Whenever a modify command, for example, is issued it is always a good idea to check if any objects have been selected as this will be the modified set of entities. In the case of single entity modifiers such as break, it as also good practice to clear the selection set prior to issuing that tool. Corel Visual CADD will do most of this automatically but to provide the most consistent results it is best to keep the selection set monitored.

See Also VCIsCurrentErased, VCIsCurrentSelected

VCIsCurrentErased

Version 1.2

Description Determines if the current entity has been erased.

Declaration

C/C++: extern "C" vbool WINAPI VCIsCurrentErased(short* iError);

Visual Basic: Declare Function VCIsCurrentErased Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCIsCurrentErased(var iError: Integer):Boolean; far;

Parameters *Returns* - an integer value representing true or false.
0 - entity is not erased.
1 - entity is erased.

Notes When stepping through the database entities in a drawing, all entities will become current at some point even those which have been erased. To eliminate problems that may occur in database consistency when erased entities are brought back through an applications negligence, each entity should be checked to determine whether it has been previously erased. Unless the applications purpose is to bring back erased entities, erased entities should be skipped when parsing the database and making edits to the drawing.

See Also VCSetCurrentErased, VCNextEntity, VCFirstEntity, VCSetCurrentEntity

VCIsCurrentSelected

Version 1.2

Description Checks the selection state of the current entity.

Declaration

C/C++: extern "C" vbool WINAPI VCIsCurrentSelected(short* iError);

Visual Basic: Declare Function VCIsCurrentSelected Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCIsCurrentSelected(var iError: Integer):Boolean; far;

Parameters *Returns* - an integer value representing true or false.
0 - entity is not selected.
1 - entity is selected.

Notes Each entity maintains a flag relating to its current selection state. This flag can be checked to determine whether an application should ignore the entity or process a routine.

See Also VCNextEntity, VCFirstEntity, VCSetCurrentEntity, VCSetCurrentSelected

VCIsCurrentWorldValid

Version 1.2

Description Verifies whether or not the currently set world is valid for displaying Corel Visual CADD graphical information and if the world still exists.

Declaration

C/C++: extern "C" vbool WINAPI VCIsCurrentWorldValid();

Visual Basic: Declare Function VCIsCurrentWorldValid Lib "VCMAIN32.DLL" () As Integer

Delphi: function VCIsCurrentWorldValid:Boolean; far;

Parameters *Returns* - an integer value representing true or false.
0 is false.
1 is true.

Notes When a world is established it still can be closed out by the user. While Corel Visual CADD does a good job of making sure the current world is valid, it is good practice to verify the validity of a world before trying to set it as current, as it may have been closed since its creation. Most Windows objects are not suitable viewing areas for graphics. VCIsCurrentWorldValid checks to see if the previously established object is valid or not and returns a true or false.

See Also VCIsDrawingDirty

VCIsDrawingDirty

Version 1.2

Description Returns a value determining whether the drawing has been changed.

Declaration

C/C++: extern "C" vbool WINAPI VCIsDrawingDirty(void);

Visual Basic: Declare Function VCIsDrawingDirty Lib "VCMAIN32.DLL" () As Integer

Delphi: function VCIsDrawingDirty:Boolean; far;

Parameters *Returns* - value determining whether the drawing has changed.
0 - no changes made.
1 - changes have been made.

Notes When closing a drawing changes may have occurred since it was opened. In order to determine if changes have occurred, the function VCIsDrawingDirty should be called. If there are changes to the drawing the application will probably want to save the changes before closing the drawing.

See Also VCIsCurrentWorldValid

VCIsFileLocked

Version 2.0

Description Specifies if the file is locked by a user.

Declaration

C/C++ extern "C" vbool WINAPI VCIsFileLocked(char* szFilename, char* szLockedByName, char* szTimeLocked);

Visual Basic Declare Function VCIsFileLocked Lib "VCMAIN32.DLL" (ByVal szFileName As String, ByVal szLockedByName As String, ByVal szTimeLocked As String) As Integer

Delphi function VCIsFileLocked(szFilename: PChar; szLockedByName: PChar;

Parameters *Return* - whether the file is currently locked.
0 - it is not locked.
1 - it is locked.
szFileName - the file in question.
szLockedBy - if the file is locked the function returns the user name with the open file.
szTimeLocked - if the file is locked the function returns the system time the file was locked.

Notes Locked files can not be modified by another Corel Visual CADD user on a network until the drawing is saved or closed. Other users can only open, view and copy the drawing. The user name is taken form the registered user name stored in the registry for the installed machine.

See Also VCIsFileLockedByCurrentUser, VCLockFile

VCIsFileLockedByCurrentUser

Version	2.0
Description	Specifies if the active drawing is locked by the current user.
Declaration	
C/C++	extern "C" vbool WINAPI VCIsFileLockedByCurrentUser();
Visual Basic	Declare Function VCIsFileLockedByCurrentUser Lib "VCMAIN32.DLL" () As Integer
Delphi	function VCIsFileLockedByCurrentUser:Boolean; far;
Parameters	*Return* - if the current user has locked the file. 0 - the current user has not locked the file. 1 - the current user has locked the file.
Notes	Locked files can not be modified by another Corel Visual CADD user on a network until the drawing is saved or closed. Other users can only open, view and copy the drawing. The user name is taken form the registered user name stored in the registry for the installed machine.
See Also	VCIsFileLockedr, VCLockFile

VCIsFilterActive

Version 1.2

Description Specifies the state of the selection filter.

Declaration

C/C++: extern "C" vbool WINAPI VCIsFilterActive(short* iError);

Visual Basic: Declare Function VCIsFilterActive Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCIsFilterActive(var iError: Integer):Boolean; far;

Parameters *Returns* - value determining whether the selection filter is active.
0 - the filter is not active.
1 - the filter is active.

Notes The API allows a filter setting entities prior to making selections. By setting a selection criteria based on entity properties and settings, the selection routine will only "capture" those objects meeting the filter criteria. The filter criteria can be set based on entity kind, layer, color, line type and line width.

See Also VCIsCurrentWorldValid

VCIsFontNameVText

Version 1.2

Description Determines if the specified font is a Corel Visual CADD vector font.

Declaration

C/C++: extern "C" vbool WINAPI VCIsFontNameVText(short* iError, char* pS);

Visual Basic: Declare Function VCIsFontNameVText Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS As String) As Integer

Delphi: function VCIsFontNameVText(var iError: Integer; pS: PChar):Boolean; far;

Parameters *Returns* - value determining whether the font is vector.
0 - the font is not a vector font.
1 - the font is a vector font.

Notes Corel Visual CADD utilizes both TrueType Fonts and built in vector fonts. The vector fonts can be converted from other font formats such as .SHX and .FNT. When working with text entities it is important to understand the type of font being used. Certain settings such as Bold, Italic and Underline only effect TrueType Fonts while others such as Italic value are designed for vector fonts. Therefore, when altering the settings of an existing text entity it is necessary to determine the type of font in order to apply the appropriate settings.

See Also VCIsCurrentWorldValid, VCGetTextFontName,

VCIsGraphic

Version 2.0

Description Determines if the current entity is a graphic entity, only hatches, fills, line types and text are considered graphic entities.

Declaration

C/C++ extern "C" vbool WINAPI VCIsGraphic(short* iError);

Visual Basic Declare Function VCIsGraphic Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi function VCIsGraphic(var iError: Integer):Boolean; far;

Parameters i*Return* - whether the current entity is a graphic entity.

Notes Some entities are defined by several graphical objects, hatch patterns, fills, line types and fonts. For instance, a hatch pattern is defined by lines to make a useful pattern. These entities are not available for access through the standard database parsing routines provided. This is due to the fact that typically an application will not need this specific information. Most applications will need to simply parse the database and retrieve the entity information provided. In situations where a custom vector output file is being defined or to guide a CNC milling machine, the application may need to define all the vectors making up even the complex entities. The graphic handle method allow for this detailed parsing functionality.

In order to access the information an application should first create a graphics handle using VCCreateGraphicsHandle. This function creates a parsing list from the current entity if it is a graphic entity, hatch, fill, text or line type. The iError return will be > 0 if the current entity is not a graphic entity. The application can then parse the new set with VCFirstGraphic and VCNextGraphic. Any required information can be retrieved using any standard query function such as VCGetCurrentEntityPoint. The entity is considered read-only and only retrieval API routines may be utilized. The individual graphic entities can not be set with any command. After completing the parse the application should call VCDeleteGraphicHandle to destroy the created handle.

See Also VCDeleteGraphicsHandle, VCFirstGraphic, VCNextGraphic

VCIsOleWorld

Version 1.2

Description Determines if the current world is contained within an OLE container.

Declaration

C/C++: extern "C" vbool WINAPI VCIsOleWorld();

Visual Basic: Declare Function VCIsOleWorld Lib "VCMAIN32.DLL" () As Integer

Delphi: function VCIsOleWorld:Boolean; far;

Parameters *iReturn* - value determining whether the drawing is a OLE object.
0 - the world is not an OLE world.
1 - the world is an OLE world.

Notes When a Corel Visual CADD drawing is linked to other applications, the drawing world receives a flag for notification. By using this value an application can determine if the drawing is inside the Corel Visual CADD frame or if it an OLE object in another application.

See Also VCIsCurrentWorldValid

VCIsRedoable

Version	1.2
Description	Determines if the last command is redoable.
Declaration	
C/C++:	extern "C" vbool WINAPI VCIsRedoable(void);
Visual Basic:	Declare Function VCIsRedoable Lib "VCMAIN32.DLL" () As Integer
Delphi:	function VCIsRedoable:Boolean; far;
Parameters	No additional parameters are used in this subroutine.
Notes	Commands are only redoable immediately after an undo and before any modifications or drawing additions are made.
See Also	VCBeginOperation, VCEndOperation

VCIsScriptAssigned

Version 1.2

Description Verifies whether a script has been assigned to a key sequence.

Declaration

C/C++: extern "C" vbool WINAPI VCIsScriptAssigned(short iShift, short iKey);

Visual Basic: Declare Function VCIsScriptAssigned Lib "VCMAIN32.DLL" (ByVal iShift As Integer, ByVal iKey As Integer) As Integer

Delphi: function VCIsScriptAssigned(iShift: Integer; iKey: Integer):Boolean; far;

Parameters *iShift* - determines the state of the modifier keys.
0 - none.
1 - shift.
2 - ctrl.
3 - alt.
iKey - the ASCII code representing the desired key.
returns an integer representing true or false.
0 - false.
1 - true.

Notes When assigning scripts it is often necessary to determine if a script has already been assigned to a key sequence. VCIsScriptAssigned determines this, letting the application determine whether to edit the existing script or overwrite it.

See Also VCMacro

VCIsSymbolLoaded

Version 1.2

Description Determines if the specified symbol has been loaded into the Corel Visual CADD symbol pool.

Declaration

C/C++: extern "C" vbool WINAPI VCIsSymbolLoaded(char* szSymbolName);

Visual Basic: Declare Function VCIsSymbolLoaded Lib "VCMAIN32.DLL" (ByVal szSymbolName As String) As Integer

Delphi: function VCIsSymbolLoaded(szSymbolName: PChar):Boolean; far;

Parameters *zSymbolName* - is the name of the symbol.
Returns - an integer value representing true or false.
0 - symbol is not loaded.
1 - symbol is loaded.

Notes When using symbols in applications, they must first be loaded into memory. However before loading a new symbol into memory it is a good idea to check if the symbol is already loaded. This prevents any conflicts in symbol names between symbols already existing in memory and those that may be loaded from disk or those that may be created with an external application.

See Also VCGetSymbolName, VCGetSymbolIndex, VCOpenVCS

VCIsTextFontVText

Version 1.2

Description An extension of VCIsFontNameVText that permits checking an existing text entity.

Declaration

C/C++: extern "C" vbool WINAPI VCIsTextFontVText(short* iError);

Visual Basic: Declare Function VCIsTextFontVText Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi: function VCIsTextFontVText(var iError: Integer):Boolean; far;

Parameters *Returns* - value determining whether the font is vector.
0 - the font is not a vector font.
1 - the font is a vector font.

Notes Corel Visual CADD utilizes both TrueType Fonts and built in vector fonts. The vector fonts can be converted from other font formats such as .SHX and .FNT. When working with text entities it is important to understand the type of font being used. Certain settings such as Bold, Italic and Underline only effect TrueType Fonts while others such as Italic value are designed for vector fonts. Therefore, when altering the settings of an existing text entity it is necessary to determine the type of font in order to apply the appropriate settings.

See Also VCIsFontNameVText

VCIsToggle

Version 1.2

Description Returns a true or false value based on whether the command is a toggle setting.

Declaration

C/C++: extern "C" vbool WINAPI VCIsToggle(WORD id);

Visual Basic: Declare Function VCIsToggle Lib "VCMAIN32.DLL" (ByVal id As Integer) As Integer

Delphi: function VCIsToggle(id: Integer):Boolean; far;

Parameters *id* - the command id for which the inquiry is made (see Appendix A for command id's).
returns - integer representing true or false.
0 is false.
1 is true.

Notes Several settings in Corel Visual CADD are toggles, that is they are either on or off. VCIsToggle checks the command to verify whether it is a toggle or not.

See Also VCToggle

VCIsUndoable

Version 1.2

Description Determines if the last command is undoable.

Declaration

C/C++: extern "C" vbool WINAPI VCIsUndoable(void);

Visual Basic: Declare Function VCIsUndoable Lib "VCMAIN32.DLL" () As Integer

Delphi: function VCIsUndoable:Boolean; far;

Parameters *returns* - integer representing true or false.
0 is false.
1 is true

Notes Corel Visual CADD modifies entities by erasing them and then recreating them with the changes. This allows Corel Visual CADD to maintain undo capabilities by erasing the new entity and returning the original. Whenever an entity is modified or added to the drawing database, it is undoable. This function checks to see if the last command is undoable. If there has been nothing added to the drawing then there is nothing to undo. Operations that do affect the drawing database such as zooms are not undoable. In addition, after a pack data command has been performed, no modifications prior to the operation are undoable.

See Also VCBeginOperation, VCEndOperation

VCIsWorldEmpty

Version	1.2
Description	Checks to see if an existing drawing world contains any drawing entities.
Declaration	
C/C++:	extern "C" vbool WINAPI VCIsWorldEmpty(WORLDHANDLE hW);
Visual Basic:	Declare Function VCIsWorldEmpty Lib "VCMAIN32.DLL" (ByVal hW As Long) As Integer
Delphi:	function VCIsWorldEmpty(hW: Longint):Boolean; far;
Parameters	*hW* - the Corel Visual CADD worldhandle used internally to reference each open drawing world. *Returns* - an integer value representing true or false. 0 is false. 1 is true..
Notes	Often before destroying or opening a new world is it useful to know whether the current world is empty.
See Also	VCDestroyWorld, VCNewWorld, VCSetCurrWorld, VCIsCurrentWorldValid, VCGetCurrWorld

VCIsWorldValid

Version 1.2

Description Verifies whether or not the currently set world is valid for displaying Corel Visual CADD graphical information.

Declaration

C/C++: extern "C" vbool WINAPI VCIsWorldValid(WORLDHANDLE hW);

Visual Basic: Declare Function VCIsWorldValid Lib "VCMAIN32.DLL" (ByVal hW As Long) As Integer

Delphi: function VCIsWorldValid(hW: Longint):Boolean; far;

Parameters *hW* is the Corel Visual CADD world handle used to reference drawing areas.
Returns - an integer value representing true or false.
0 is false.
1 is true.

Notes While and Windows object has a hWnd and thus can be established as the current world. Most Windows objects are not suitable viewing areas for graphics. VCIsCurrentWorldValid checks to see if the previously established object is valid and returns a true or false.

See Also VCIsCurrentWorldValid, VCIsWorldEmpty

VCLastEntity

Version 1.2

Description Makes the last entity in the drawing database the current entity.

Declaration

C/C++: extern "C" void WINAPI VCLastEntity(short* iError, ENTITYHANDLE* lH);

Visual Basic: Declare Sub VCLastEntity Lib "VCMAIN32.DLL" (iError As Integer, lH As Long)

Delphi: procedure VCLastEntity(var iError: Integer; var lH: Longint); far;

Parameters *lH* - entity handle for the last entity in the database.

Notes After creating new objects in the Corel Visual CADD database it must be drawn before it appears on the screen. This can be done by the user doing a zoom all or through code using VCDrawCurrentEntity. However the entity must first be current VCLastEntity will make the last entity current so it can be displayed. All entities added to the drawing database are added to the end and will thus be the last entity.

See Also VCNextEntity, VCFirstEntity, VCFirstSelected, VCNextSelected, VCFirstSelected, VCNextSelected

VCLayerHasProperties

Version 2.0.1

Description Determines if the given layer has layer properties assigned.

Declaration

C/C++ extern "C" vbool WINAPI VCLayerHasProperties(short* iError, short iLayer);

Visual Basic Declare Function VCLayerHasProperties Lib "VCMAIN32.DLL" (iError As Integer, ByVal iLayer As Integer) As Integer

Delphi function VCLayerHasProperties(var iError: Integer; iLayer: Integer):Boolean; far;

Parameters *iLayer* - the layer index in question.
returns - value indicating layer property status.
0 - does not have layer properties.
1 - has layer properties.

Notes Layer properties were introduced into v2.0.1 allowing properties to be assigned by layer rather than by entity. For example, a layer can be set so all entities drawn on the layer will be a specific color, line type and line width. This will override the current properties settings when active. VCGetUseByLayerProperties is used to determine if the layer has active property settings while VCSetUseByLayerProperties allows an application to choose which properties to use. VCSetLayerProperties will set the values for the layer and VCClearLayerProperties turns the capability off and clears all associated values. It is important to keep track of the state of layer properties when modifying entities in the drawing. For example, if you set the color index using VCSetColorIndex but the layer properties are enabled the proper color may not get applied. Therefore when attempting to control the properties of entities as they are placed it is imperative that the application monitor the setting for by layer control as the information is being supplied by the API.

See Also VCGetLayerProperties

VCLButtonDblClk

Version 1.2

Description Issues a left button double click in the drawing area. Ends a continuous entity placement by placing a point at the specified location.

Declaration

C/C++: extern "C" void WINAPI VCLButtonDblClk(long lParam, WORD wParam);

Visual Basic: Declare Sub VCLButtonDblClk Lib "VCMAIN32.DLL" (ByVal lParam As Long, ByVal wParam As Integer)

Delphi: procedure VCLButtonDblClk(lParam: Longint; wParam: Integer); far;

Parameters *lParam* - a packed coordinate pair as used by Windows.
wParam - passed by Windows functions to represent the identifier of the mouse message.

Notes All mouse subroutines as passed by Windows function pass the coordinate values through a lParam structure which contains the x and y coordinates and other pertinent information not used by Corel Visual CADD. Also included is a wParam which contains any modifiers to the mouse movement such as the state of the shift, ctrl, and alt keys. These may or may not be used by Corel Visual CADD to modify the results of the mouse movements depending on the current context in the application. Whenever an external application receives a left button double click message in the drawing area, the application should send the VCLButtonDblClk message to Corel Visual CADD in order to invoke the expected response. This makes if behave as if it were in the Corel Visual CADD drawing area.

The MFC Class Library in MS Visual C++ references mouse movements and points through a CPoint class structure. The macro MAKELPARAM can be used to convert the given CPoint structure to a LPARAM compatible with the Corel Visual CADD API.

See Also VCLButtonDown, VCLButtonDown2

VCLButtonDown

Version 1.2

Description Sends a left button down message to Corel Visual CADD effectively selecting a coordinate for a tool, or selecting an entity.

Declaration

C/C++: extern "C" void WINAPI VCLButtonDown(long lParam, WORD wParam);

Visual Basic: Declare Sub VCLButtonDown Lib "VCMAIN32.DLL" (ByVal lParam As Long, ByVal wParam As Integer)

Delphi: procedure VCLButtonDown(lParam: Longint; wParam: Integer); far;

Parameters *lParam* - a packed coordinate pair as used by Windows.
wParam - passed by Windows functions to represent the identifier of the mouse message.

Notes All mouse subroutines as passed by Windows function pass the coordinate values through a lParam structure which contains the x and y coordinates and other pertinent information not used by Corel Visual CADD. Also included is a wParam which contains any modifiers to the mouse movement such as the state of the shift, ctrl, and alt keys. These may or may not be used by Corel Visual CADD to modify the results of the mouse movements depending on the current context in the application. Whenever an external application receives a left button down message in the drawing area, the application should send the VCLButtonDown message to Corel Visual CADD in order to invoke the expected response. This makes if behave as if it were in the Corel Visual CADD drawing area.

The MFC Class Library in MS Visual C++ references mouse movements and points through a CPoint class structure. The macro MAKELPARAM can be used to convert the given CPoint structure to a LPARAM compatible with the Corel Visual CADD API.

See Also VCLButtonDown2, VCRButtondown, VCMButtonDown

VCLButtonDown2

Version 1.2

Description Invokes a left button down at the specified screen coordinates.

Declaration

C/C++: extern "C" void WINAPI VCLButtonDown2(short cx, short cy);

Visual Basic: Declare Sub VCLButtonDown2 Lib "VCMAIN32.DLL" (ByVal cx As Integer, ByVal cy As Integer)

Delphi: procedure VCLButtonDown2(cx: Integer; cy: Integer); far;

Parameters *cx* - the screen coordinate from 0 to the current number or horizontal screen pixels.
cy - the screen coordinate from 0 to the current number of vertical screen pixels.

Notes Similar to the VCLButtonDownWorldPoint sub routine except that the coordinates specified are screen coordinates and is completely unrelated to drawing size. Various zooms and views will affect the location of the clicks. Corel Visual CADD will convert these click points into drawing coordinates when used to locate points for a drawing or editing tool. This command can be used to select drawing entities or to locate points of a drawing or editing tool in the drawing area. This behaves exactly as the user clicking in the drawing area to select drawing coordinates.

See Also VCLButtonDown, VCRButtondown, VCMButtonDown

VCLButtonDownWorldPoint

Version	1.2
Description	Invokes a left button down message at the specified "real world" drawing coordinates.
Declaration	
C/C++:	extern "C" void WINAPI VCLButtonDownWorldPoint(Point2D* dpW);
Visual Basic:	Declare Sub VCLButtonDownWorldPoint Lib "VCMAIN32.DLL" (dpW As Point2D)
Delphi:	procedure VCLButtonDownWorldPoint(var dpW: Point2D); far;
Parameters	*dpW* is the Point2D structure specifying where in the drawing area the left click is to take place.
Notes	VCLButtonDownWorldPoint issues a click at the real world position in the drawing. This is related to the drawing area and size. If a click is needed in a specific screen area use VCLButtonDown2 instead. This command can be used to select drawing entities or to locate points of a drawing or editing tool in the drawing area. This behaves exactly as the user clicking in the drawing area to select drawing coordinates.
See Also	VCLButtonDown, VCLButtonDown2, VCRButtondown, VCMButtonDown

VCLButtonUp

Version 1.2

Description Issues a left mouse button up command to Corel Visual CADD.

Declaration

C/C++: extern "C" void WINAPI VCLButtonUp(long lParam, WORD wParam);

Visual Basic: Declare Sub VCLButtonUp Lib "VCMAIN32.DLL" (ByVal lParam As Long, ByVal wParam As Integer)

Delphi: procedure VCLButtonUp(lParam: Longint; wParam: Integer); far;

Parameters *lParam* is a packed coordinate pair as used by Windows.
wParam is passed by Windows functions to represent the identifier of the mouse message.

Notes All mouse subroutines as passed by Windows function pass the coordinate values through a lParam structure which contains the x and y coordinates and other pertinent information not used by Corel Visual CADD. Also included is a wParam which contains any modifiers to the mouse movement such as the state of the shift, ctrl, and alt keys. These may or may not be used by Corel Visual CADD to modify the results of the mouse movements depending on the current context in the application. Whenever an external application receives a left button up message in the drawing area, the application should send the VCLButtonUp message to Corel Visual CADD in order to invoke the expected response. This makes if behave as if it were in the Corel Visual CADD drawing area.

The MFC Class Library in MS Visual C++ references mouse movements and points through a CPoint class structure. The macro MAKELPARAM can be used to convert the given CPoint structure to a LPARAM compatible with the Corel Visual CADD API.

See Also VCLButtonDown, VCLButtonDown2, VCRButtondown, VCMButtonDown

VCLButtonUpTimerReset

Version	2.0.1
Description	Enables a user tools to simulate a button up event for "Drag-n-Drop".
Declaration	
C/C++	extern "C" void WINAPI VCLButtonUpTimerReset(short* iError);
Visual Basic	Declare Sub VCLButtonUpTimerReset Lib "VCMAIN32.DLL" (iError As Integer)
Delphi	procedure VCLButtonUpTimerReset(var iError: Integer); far;
Parameters	No additional parameters are used in this subroutine.
Notes	This function resets the "Drag-n-Drop" timer so the next LButtonUp will send a LButtonDown message to the current tool. This is how VCADD works internally, tools never handle LButtonUP messages. For example, the Symbol Manager calls VCLButtonUpTimerReset when a symbol is dragged off the listbox then creates a SymbolPlace tool. SymbolPlace is sent a LButtonDown message from Corel Visual CADD when the button is let up allowing a user tool to simulate Drag-n-Drop.
See Also	VCGetUserToolLBUp

VCLineAngle

Version 1.2

Description Returns the angle between the line defined by the included points and the horizontal.

Declaration

C/C++: extern "C" void WINAPI VCLineAngle(short* iError, double* dAngle, Point2D* dpP0, Point2D* dpP1);

Visual Basic: Declare Sub VCLineAngle Lib "VCMAIN32.DLL" (iError As Integer, dAngle As Double, dpP0 As Point2D, dpP1 As Point2D)

Delphi: procedure VCLineAngle(var iError: Integer; var dAngle: Double; var dpP0 Point2D; var dpP1: Point2D); far;

Parameters *dAngle* - the resultant angle of the line.
dpP0 - the coordinates of the first end of the line.
dpP1 - the coordinates of the second end of the line.

Notes Provides basic ability to determine the angle of a line defined by two provided points to the horizontal.

See Also VCLineLength, VCLinePerpPoint, VCLineAngle

VCLineLength

Version 1.2

Description Returns the length of the line defined by the two points.

Declaration

C/C++: extern "C" void WINAPI VCLineLength(short* iError, double* dAngle, Point2D* dpP0, Point2D* dpP1);

Visual Basic: Declare Sub VCLineLength Lib "VCMAIN32.DLL" (iError As Integer, dAngle As Double, dpP0 As Point2D, dpP1 As Point2D)

Delphi: procedure VCLineLength(var iError: Integer; var dAngle: Double; var dpP0 Point2D; var dpP1: Point2D); far;

Parameters *dAngle* - the result 2pt angle of the line.
dpP0 - the coordinates of the first end of the line.
dpP1 - the coordinates of the second end of the line.

Notes Provides basic ability to determine distance between two points.

See Also VCLineLength, VCLinePerpPoint, VCLineAngle

VCLinePerpPoint

Version 1.2

Description Calculates the perpendicular projection from a specified point to a the defined line.

Declaration

C/C++: extern "C" void WINAPI VCLinePerpPoint(short* iError, Point2D* dpC, Point2D* dpP0, Point2D* dpP1, Point2D* dpOff);

Visual Basic: Declare Sub VCLinePerpPoint Lib "VCMAIN32.DLL" (iError As Integer, dpC As Point2D, dpP0 As Point2D, dpP1 As Point2D, dpOff As Point2D)

Delphi: procedure VCLinePerpPoint(var iError: Integer; var dpC: Point2D; var dpP0 Point2D; var dpP1: Point2D; var dpOff: Point2D); far;

Parameters
dpC - returned by Corel Visual CADD as the calculated point on the line.
dpP0 - the first point defining the line.
dpP1 - the second point defining the line.
pOff - the point to calculate the perpendicular projection from.

Notes Snap perpendicular typically supplies the functionality to snap geometry perpendicular to existing linear geometry, however through the API this is not necessarily convenient. This function provides that functionality to the API and thus allows perpendicular constructions to existing or even simply defined lines.

See Also VCLineLength, VCLinePerpPoint, VCLineAngle

VCLinePointCompute

Version 1.2

Description Calculates the coordinates of a point a specified angle and distance from the given line.

Declaration

C/C++: extern "C" void WINAPI VCLinePointCompute(short* iError, Point2D* dpC, Point2D* dpP0, Point2D* dpP1, double dDist, double dAngle);

Visual Basic: Declare Sub VCLinePointCompute Lib "VCMAIN32.DLL" (iError As Integer, dpC As Point2D, dpP0 As Point2D, dpP1 As Point2D, ByVal dDist As Double, ByVal dAngle As Double)

Delphi: procedure VCLinePointCompute(var iError: Integer; var dpC: Point2D; var dpP0 Point2D; var dpP1: Point2D; dDist: Double; dAngle: Double); far;

Parameters
dpC - returned by Corel Visual CADD as the calculated point on the line.
dpP0 - the first point defining the line.
dpP1 - the second point defining the line.
Dist - the distance out the projection.
dAngle - the angle from the defined line at the dpP0 point.

Notes Using existing lines or line definitions, this function allows a point to be located a specified distance and angle from the first vertex of the line. Imagine standing at point dpP0 looking at dpP1, if a user were to turn dAngle radians in the clockwise direction and look Dist distance out at that angle dpC is the coordinates of the location now being viewed.

See Also VCLinePointCompute

VCLoadAlias

Version	1.2
Description	Loads a custom file containing two-letter command structures.
Declaration	
C/C++:	extern "C" void WINAPI VCLoadAlias(char* szFile, short* iError);
Visual Basic:	Declare Sub VCLoadAlias Lib "VCDLG32.DLL" (ByVal szFile As String, iError As Integer)
Delphi:	procedure VCLoadAlias(szFile: PChar; var iError: Integer); far;
Parameters	*szFile* - the path and file name for the new commands.
Notes	The Corel Visual CADD interface can be customized to fit an applications specific task. This customization includes loading new menus, tool palettes, speedbars and two-letter commands. All the custom files are contained in text files located in the system path. These files can be edited to create the interface desired and then loaded directly through the API.
See Also	VCLoadCmdExt,VCLoadMainSpeedbar, VCLoadToolPalette

VCLoadAscii

Version	2.0
Description	Loads an ASCII text file and initiate a placement tool for the file.
Declaration	
C/C++	extern "C" void WINAPI VCLoadAscii(short* iError, char* szAscii);
Visual Basic	Declare Sub VCLoadAscii Lib "VCTOOL32.DLL" (iError As Integer, ByVal szAscii As String)
Delphi	procedure VCLoadAscii(var iError: Integer; szAscii: PChar); far;
Parameters	*szAscii* - the file to load.
Notes	An ASCII text file can be directly loaded and placed into a drawing. The command will load the text file using the current text settings.
See Also	VCLoadDrawing

VCLoadCmdExt

Version 1.2

Description Loads a file containing custom commands.

Declaration

C/C++: extern "C" void WINAPI VCLoadCmdExt(char* szFile, short* iError);

Visual Basic: Declare Sub VCLoadCmdExt Lib "VCDLG32.DLL" (ByVal szFile As String, iError As Integer)

Delphi: procedure VCLoadCmdExt(szFile: PChar; var iError: Integer); far;

Parameters *szFile* - the path and file name for the new commands.

Notes Custom commands are user-defined commands that can optimize the work environment. They are base on scripts. More details are available in the User Manual. The Corel Visual CADD interface can be customized to fit an applications specific task. This customization includes loading new menus, tool palettes, speedbars and two-letter commands. All the custom files are contained in text files located in the system path. These files can be edited to create the interface desired and then loaded directly through the API.

See Also VCLoadAlias ,VCLoadMainSpeedbar, VCLoadToolPalette

VCLoadDrawing

Version 1.2

Description Loads a drawing and converts it if necessary.

Declaration

C/C++: extern "C" void WINAPI VCLoadDrawing(short* iError, char* pName, short iFileType);

Visual Basic: Declare Sub VCLoadDrawing Lib "VCTRAN32.DLL" (iError As Integer, ByVal pName As String, ByVal iFileType As Integer)

Delphi: procedure VCLoadDrawing(var iError: Integer; pName: PChar; iFileType Integer); far;

Parameters *pName* - the name and path of the file to be loaded
iFileType - represents the type of drawing file that is to be loaded.
-1 - Determine By Extension
0 - FILE_VCD
3 - FILE_GCD
5 - FILE_DWG
6 - FILE_DXF

Notes Corel Visual CADD will load all the for-mentioned file types. However, be aware that Visual CADD has limited 3D support and may not be able to convert all file information from AutoCAD drawings. Also note that if using the iFileType of -1, the extension is the sole classification for the file type to be converted. This can be a problem with pre-Generic CADD 6.0 drawings as these were *.DWG files but not the same *.DWG files as AutoCAD. This can be a problem and may cause the program to crash.

See Also VCACADReadWith3D, VCOpenVCS, VCOpenCMP

VCLoadMainSpeedbar

Version	1.2
Description	Loads a file containing a custom speedbar.
Declaration	
C/C++:	extern "C" void WINAPI VCLoadMainSpeedbar(char* szFile, short* iError);
Visual Basic:	Declare Sub VCLoadMainSpeedbar Lib "VCDLG32.DLL" (ByVal szFile As String, iError As Integer)
Delphi:	procedure VCLoadMainSpeedbar(szFile: PChar; var iError: Integer); far;
Parameters	*szFile* - the path and file name for the new commands.
Notes	The Corel Visual CADD interface can be customized to fit an applications specific task. This customization includes loading new menus, tool palettes, speedbars and two-letter commands. All the custom files are contained in text files located in the system path. These files can be edited to create the interface desired and then loaded directly through the API.
See Also	VCLoadAlias ,VCLoadCmdExt, VCLoadToolPalette

VCLoadPlotterDriver

Version 2.0

Description Loads a plotter driver for the direct plot routine.

Declaration

C/C++ extern "C" void WINAPI VCLoadPlotterDriver(short* iError, char* szName);

Visual Basic Declare Sub VCLoadPlotterDriver Lib "VCDLG32.DLL" (iError As Integer, ByVal szName As String)

Delphi procedure VCLoadPlotterDriver(var iError: Integer; szName: PChar); far;

Parameters *szName* - the name of the plotter driver language.

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

See Also VCLoadPlotterLanguage, VCLoadPlotterPenMap

VCLoadPlotterLanguage

Version 2.0

Description Loads a plotter language for the direct plot routine.

Declaration

C/C++ extern "C" void WINAPI VCLoadPlotterLanguage(short* iError, char* szName);

Visual Basic Declare Sub VCLoadPlotterLanguage Lib "VCDLG32.DLL" (iError As Integer, ByVal szName As String)

Delphi procedure VCLoadPlotterLanguage(var iError: Integer; szName: PChar); far;

Parameters *szName* - the name of the plotter language

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs to be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type.

See Also VCLoadPlotterDriver, VCLoadPlotterPenMap

VCLoadPlotterPenMap

Version 2.0

Description Loads a pen map for the direct plot routine.

Declaration

C/C++ extern "C" void WINAPI VCLoadPlotterPenMap(short* iError, char* szName);

Visual Basic Declare Sub VCLoadPlotterPenMap Lib "VCDLG32.DLL" (iError As Integer, ByVal szName As String)

Delphi procedure VCLoadPlotterPenMap(var iError: Integer; szName: PChar); far;

Parameters *szName* - the name of the pen map to load

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

See Also VCLoadPlotterDriver, VCLoadPlotterLanguage

VCLoadToolPalette

Version	1.2
Description	Loads a file containing a custom speedbar.
Declaration	
C/C++:	extern "C" void WINAPI VCLoadToolPalette(char* szFile, short* iError);
Visual Basic:	Declare Sub VCLoadToolPalette Lib "VCDLG32.DLL" (ByVal szFile As String, iError As Integer)
Delphi:	procedure VCLoadToolPalette(szFile: PChar; var iError: Integer); far;
Parameters	*szFile* - the path and file name for the new commands.
Notes	The Corel Visual CADD interface can be customized to fit an applications specific task. This customization includes loading new menus, tool palettes, speedbars and two-letter commands. All the custom files are contained in text files located in the system path. These files can be edited to create the interface desired and then loaded directly through the API.
See Also	VCLoadAlias ,VCLoadCmdExt,VCLoadMainSpeedbar

VCLoadVCDFromFile

Version	2.0
Description	Loads a Corel Visual CADD native file.
Declaration	
C/C++	extern "C" void WINAPI VCLoadVCDFromFile(short* iError, char* pS_);
Visual Basic	Declare Sub VCLoadVCDFromFile Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS_ As String)
Delphi	procedure VCLoadVCDFromFile(var iError: Integer; pS_: PChar); far;
Parameters	*pS*- the path and file name for saving the drawing.
Notes	VCLoadVCDFromFile is a specific load routine to work with Corel Visual CADD native files. An error will occur if attempting to load files other than *.VCD files. In situations where other vector drawing formats such *.DWG, *.GCD or *.DXF will also be used the routine VCLoadDrawing should be implemented which will load all these vector file types.
See Also	VCSaveVCDToStream, VCSaveVCDToFile, VCLoadVCDFromStream

VCLoadVCDFromStream

Version	2.0
Description	Loads a Corel Visual CADD drawing from an OLE stream.
Declaration	
C/C++	extern "C" void WINAPI VCLoadVCDFromStream(short* iError, void* pS_);
Visual Basic	Declare Sub VCLoadVCDFromStream Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS_ As String)
Delphi	procedure VCLoadVCDFromStream(var iError: Integer; var pS_: Pointer); far;
Parameters	*pS* - a pointer to the data stream for storing the information.
Notes	All of the Corel Visual CADD OLE handler routines are available to create an OLE server. This routine will write the file to stream for the application to handle in its OLE event. Please see the documentation for creating an OLE application in your compiler help for details on a stream.
See Also	VCSaveVCDToStream, VCSaveVCDToFile, VCLoadVCDFromFile

VCLockFile

Version 2.0

Description Locks the given file name for read only capabilities.

Declaration

C/C++ extern "C" vbool WINAPI VCLockFile(char* szFileName, vbool tfFileReadOnly, vbool tfLoading);

Visual Basic Declare Function VCLockFile Lib "VCMAIN32.DLL" (ByVal szFileName As String, ByVal tfFileReadOnly As Integer, ByVal tfLoading As Integer) As Integer

Delphi function VCLockFile(szFileName: PChar; tfFileReadOnly: Boolean; tfLoading:

Parameters *tfFileReadOnly* - flag for setting the file read0only.
0 - do not set as read-only.
1 - set as read-only.
tfLoading - flag for allowing other users to load the file.
0 - do not let others open the file.
1 - let others open the file.

Notes Locked files can not be modified by another Corel Visual CADD user on a network until the drawing is saved or closed. Other users can only open, view and copy the drawing. The user name is taken form the registered user name stored in the registry for the installed machine.

See Also VCIsFileLocked, VCIsFileLockedByCurrentUser

VCLockMessage

Version 2.0

Description Locks the status bar message to display the given message.

Declaration

C/C++ extern "C" void WINAPI VCLockMessage(short* iError, char* szMess, vbool tfLock);

Visual Basic Declare Sub VCLockMessage Lib "VCMAIN32.DLL" (iError As Integer, ByVal szMess As String, ByVal tfLock As Integer)

Delphi procedure VCLockMessage(var iError: Integer; szMess: PChar; tfLock: Boolean);

Parameters *szMess* - the message to display, should be set to NULL when unlocking.
tfLock - locks or unlocks the status display.
0 - unlocks the display.
1 - locks the display.

Notes The status bar is used to display the current tool position and guide a user through the tool operation. An application can set and change this displayed message with VCSetPrompt. Once the applications tool is complete however the default Corel Visual CADD messages will show. VCLockMessage allows an application to lock all messages from the display and provide a single instruction for a user. For example, an application may never utilize prompts but instead only requires a user to make settings in an application dialog. The application can lock the message and provide a prompt such as "Enter the settings". This prompt will not be overwritten by the Corel Visual CADD messaging system. The application then calls the VCLockMessage routine again to reset the prompt and allow Corel Visual CADD to show the prompts.

See Also VCGetCmdStr

VCLParamToPoint2D

Version 1.2

Description Converts a lParam as passed from a Windows function to a Point2D.

Declaration

C/C++:
```
extern "C" Point2D WINAPI VCLParamToPoint2D(long lParam);
extern "C" void WINAPI VCLParamToPoint2DBP(long lParam, Point2D* pRet);
```

Visual Basic:
```
Declare Sub VCLParamToPoint2DBP Lib "VCMAIN32.DLL" (ByVal lParam As Long, pRet As Point2D)
```

Delphi:
```
procedure VCLParamToPoint2DBP(lParam: Longint; var pRet: Point2D); far;
```

Parameters *lParam* - a packed coordinate pair as used by Windows.
Returns - Point2D structure composed of a double x and double y coordinate pair.

Notes Windows functions for mouse movement in particular passes the coordinate values as lParams. While these may be passed directly to many of the Corel Visual CADD functions, it may be necessary to convert these to Point2D's. This function will make that conversion. This function uses a Point2D structure which must be previously defined as a structure of x and y coordinate values both defined as doubles.

See Also VCLButtonDown, VCLButtonDown2, VCLButtonDownWorldPoint

VCMacro

Version	1.2
Description	Issues a command to Corel Visual CADD to execute the included macro string.
Declaration	
C/C++:	extern "C" void WINAPI VCMacro(char* sz);
Visual Basic:	Declare Sub VCMacro Lib "VCMAIN32.DLL" (ByVal sz As String)
Delphi:	procedure VCMacro(sz: PChar); far;
Parameters	*sz* - the string of Corel Visual CADD native or two letter commands.
Notes	Any two letter command or native Corel Visual CADD can be used to issue a macro or script command. Sequences of commands can also be sent but must be separated by semicolons. Each macro must also be concluded with a semicolon.
See Also	VCIsScriptAssigned

VCMakeValidDosFilenameForSave

Version 1.2

Description Modifies the proposed filename and checks to see if valid.

Declaration

C/C++: extern "C" vbool WINAPI VCMakeValidDosFilenameForSave(char* pFilename);

Visual Basic: Declare Function VCMakeValidDosFilenameForSave Lib "VCMAIN32.DLL" (ByVal pFilename As String) As Integer

Delphi: function VCMakeValidDosFilenameForSave(pFilename: PChar):Boolean; far;

Parameters *pFilename* - the proposed filename to change.
Returns - a 0 if successful.

Notes While this function will modify pFilename to eliminate spaces and invalid characters, it also attempts to open the file to see if it exists or is read only. If there is a problem, it will prompt the user and return a 1 if the user cancels.

See Also VCSaveDrawing, VCSaveVCS, VCSaveStyle, VCSaveVCA

VCManualEntryMode

Version 1.2

Description Sets how Corel Visual CADD interprets coordinates entered by the user.

Declaration

C/C++: extern "C" void WINAPI VCManualEntryMode(short ID);

Visual Basic: Declare Sub VCManualEntryMode Lib "VCMAIN32.DLL" (ByVal ID As Integer)

Delphi: procedure VCManualEntryMode(ID: Integer); far;

Parameters *ID* - an index representing the mode for Manual Entry. This mode is passed through the appropriate command sequence for Manual Entry Mode.

Notes Use the Manual Entry Relative command to set the operating mode to the relative manual entry mode. The manual entry mode determines how Corel Visual CADD interprets coordinates (whether Cartesian or polar) that a user types. In the relative mode, each point placed or referenced through a snap or other command becomes a temporary origin for the next operation. This mode is particularly useful when distances are measured in sequence, with the end of one measurement being the beginning of the next. In the absolute mode, coordinates are interpreted as relative to the drawing origin. This mode is particularly useful when locations are calculated or imported through external programs or macros. In basepoint mode, specify a temporary origin that remains in effect until a user changes its location or change modes. This mode is particularly useful when locations are known in relation to one specific point.

See Also VCSetMBMode, VCSetMOMOde, VCSetMRMode

VCMatchCurrentEntity

Version 1.2

Description Sets all appropriate settings to the same as the current entity.

Declaration

C/C++: extern "C" void WINAPI VCMatchCurrentEntity(short* iError);

Visual Basic: Declare Sub VCMatchCurrentEntity Lib "VCMAIN32.DLL" (iError As Integer)

Delphi: procedure VCMatchCurrentEntity(var iError: Integer); far;

Parameters No additional parameters are used with this subroutine..

Notes When retrieving settings associated with entities such as dimensions, not all properties are available through direct queries to the entity. VCMatchCurrentEntity allows an external application to set all the drawing settings that relate to the entity the same as the current entity. This allows the API to get the specific properties of the entity from the settings rather than from each entity. While this does upset the desired settings set by the user or application, VCSaveSettings will temporarily save them all prior to matching the current entity and VCRestoreSettings will bring the desired settings back again. The current entity is set with VCSetCurrentEntity, VCFirstEntity, or VCNextEntity.

See Also VCRestoreSettings, VCNextEntity, VCEndOperation, VCSetCurrentErased, VCSetCurrentEntity, VCDrawCurrentEntity, VCLastEntity, VCDuplicate, VCBeginOperation, VCGetCurrentEntityHandle, VCIsCurrentSelected, VCFirstEntity, VCSaveSettings, VCNextEntity, VCFirstEntity, VCSetCurrentEntity, VCRestoreSettings, VCSaveSettings

VCMButtonDown

Version 1.2

Description Sends a middle button down message to Corel Visual CADD effectively selecting a coordinate for a tool, or selecting an entity.

Declaration

C/C++: extern "C" void WINAPI VCMButtonDown(long lParam, WORD wParam);

Visual Basic: Declare Sub VCMButtonDown Lib "VCTOOL32.DLL" (ByVal lParam As Long, ByVal wParam As Integer)

Delphi: procedure VCMButtonDown(lParam: Longint; wParam: Integer); far;

Parameters *lParam* - a packed coordinate pair as used by Windows.
wParam - passed by Windows functions to represent the identifier of the mouse message.

Notes All mouse subroutines as passed by Windows function pass the coordinate values through a lParam structure which contains the x and y coordinates and other pertinent information not used by Corel Visual CADD. Also included is a wParam which contains any modifiers to the mouse movement such as the state of the shift, ctrl, and alt keys. These may or may not be used by Corel Visual CADD to modify the results of the mouse movements depending on the current context in the application. Whenever an external application receives a middle button down message in the drawing area, the application should send the VCMButtonDown message to Corel Visual CADD in order to invoke the expected response. This makes if behave as if it were in the Visual CADD drawing area.

See Also VCLButtonDown2, VCLButtondown, VCRButtonDown

VCMerge

Version 1.2

Description Loads a copy of an existing drawing into the current drawing without renaming or erasing the current drawing contents or environment. Does not delete or modify the file being merged into the current drawing.

Declaration

C/C++: extern "C" void WINAPI VCMerge(char* pName);

Visual Basic: Declare Sub VCMerge Lib "VCTOOL32.DLL" (ByVal pName As String)

Delphi: procedure VCMerge(pName: PChar); far;

Parameters *pName* - the drawing path and file name.

Notes Use the Merge command to combine the contents of two drawings. The name and drawing environment of the first drawing loaded are preserved, although symbols and attributes of the second drawing are added to those of the first (conflicts such as duplicate symbol names are resolved in favor of the first drawing).

See Also VCLoadDrawing, VCCopy, VCMergeDrawing, VCMergeVCDNoPaint

VCMergeDrawing

Version 1.2

Description Loads and merges the specified file into the current drawing.

Declaration

C/C++: extern "C" void WINAPI VCMergeDrawing(short* iError, char* pName, short iFileType);

Visual Basic: Declare Sub VCMergeDrawing Lib "VCTRAN32.DLL" (iError As Integer, ByVal pName As String, ByVal iFileType As Integer)

Delphi: procedure VCMergeDrawing(var iError: Integer; pName: PChar; iFileType Integer); far;

Parameters *pName* - the path and name of the file to be merged.

iFileType - represents the type of drawing file that is to be loaded.
-1 - Determine By Extension
0 - FILE_VCD
3 - FILE_GCD
5 - FILE_DWG
6 - FILE_DXF

Notes Merging a drawing will load the new drawing into the drawing already active. If there are any named layer conflicts the first drawing will retain the layer names and the merged drawing will lose the layer name where the conflict exists. The same is true of loaded symbols. If there is a conflict between names, the previous drawing will retain the symbol definitions and the merged drawing will be forced to adopt the new symbols. This is the case for any conflicts: whichever drawing is first wins.

See Also VCLoadDrawing, VCMergeVCDNoPaint

VCMergeVCDNoPaint

Version 1.2

Description Loads and merges the specified Corel Visual CADD drawing into the current drawing, but does not repaint the screen upon completion.

Declaration

C/C++: extern "C" void WINAPI VCMergeVCDNoPaint(short* iError, char* pInputName);

Visual Basic: Declare Sub VCMergeVCDNoPaint Lib "VCTRAN32.DLL" (iError As Integer, ByVal pInputName As String)

Delphi: procedure VCMergeVCDNoPaint(var iError: Integer; pInputName: PChar); far;

Parameters *pInputName* - the path and filename of the drawing to be merged.

Notes Merging a drawing will load the new drawing into the drawing already active. If there are any named layer conflicts the first drawing will retain the layer names and the merged drawing will lose the layer name where the conflict exists. The same is true of loaded symbols. If there is a conflict between names, the previous drawing will retain the symbol definitions and the merged drawing will be forced to adopt the new symbols. This is the case for any conflicts: whichever drawing is first wins. In this case, the drawing is not painted, or redrawn, after the merge is completed. When a repaint is initiated by another event the merged drawing will then appear.

See Also VCLoadDrawing, VCMergeDrawing

VCModalDlg

Version	2.0
Description	Initiates the dialog for the input command as modal dialog.
Declaration	
C/C++	extern "C" void WINAPI VCModalDlg(short* iError, long cmd_id);
Visual Basic	Declare Sub VCModalDlg Lib "VCDLG32.DLL" (iError As Integer, ByVal cmd_id As Long)
Delphi	procedure VCModalDlg(var iError: Integer; cmd_id: Longint); far;
Parameters	*cmd_id* - the command ID for the dialog.
Notes	Most dialogs can be launched directly through the API with a dialog routine. These dialogs can also be launched with VCModalDlg by passing the command id.
See Also	Dialog Reference

VCMouseMove

Version 1.2

Description Used to send mouse movements to Corel Visual CADD.

Declaration

C/C++: extern "C" void WINAPI VCMouseMove(long lParam, WORD wParam);

Visual Basic: Declare Sub VCMouseMove Lib "VCMAIN32.DLL" (ByVal lParam As Long, ByVal wParam As Integer)

Delphi: procedure VCMouseMove(lParam: Longint; wParam: Integer); far;

Parameters *lParam* - a packed coordinate pair as used by Windows.
wParam - passed by Windows functions to represent the identifier of the mouse message.

Notes All mouse subroutines as passed by Windows function pass the coordinate values through a lParam structure which contains the x and y coordinates and other pertinent information not used by Corel Visual CADD. Also included is a wParam which contains any modifiers to the mouse movement such as the state of the shift, ctrl, and alt keys. These may or may not be used by Corel Visual CADD to modify the results of the mouse movements depending on the current context in the application. For example, holding the ctrl key while dragging objects toggles the state of the ortho mode.

The MFC Class Library in MS Visual C++ references mouse movements and points through a CPoint class structure. The macro MAKELPARAM can be used to convert the given CPoint structure to a LPARAM compatible with the Corel Visual CADD API.

See Also VCMouseMoveWorldPoint, VCMouseMove2, VCGetUserToolMouseMove

VCMouseMove2

Version	1.2
Description	Moves the Corel Visual CADD cursor to the specified position in screen coordinates.
Declaration	
C/C++:	extern "C" void WINAPI VCMouseMove2(short cx, short cy);
Visual Basic:	Declare Sub VCMouseMove2 Lib "VCMAIN32.DLL" (ByVal cx As Integer, ByVal cy As Integer)
Delphi:	procedure VCMouseMove2(cx: Integer; cy: Integer); far;
Parameters	*cx* - the screen coordinate from 0 to the current number or horizontal screen pixels. *cy* - the screen coordinate from 0 to the current number of vertical screen pixels.
Notes	This subroutine is similar to the VCMouseMoveWorldPoint except it uses screen coordinates.
See Also	VCMouseMoveWorldPoint, VCMouseMoveWorldPoint, VCMouseMove

VCMouseMoveWorldPoint

Version 1.2

Description Moves the Corel Visual CADD cursor to the specified position in the "real world" drawing coordinates.

Declaration

C/C++: extern "C" void WINAPI VCMouseMoveWorldPoint(Point2D dpW);

Visual Basic: Declare Sub VCMouseMoveWorldPoint Lib "VCMAIN32.DLL" (dpW As Point2d)

Delphi: procedure VCMouseMoveWorldPoint(dpPW : Point2D); far;

Parameters *dpW* - the coordinate which the cursor is to be moved.

Notes This function uses a Point2D structure which must be previously defined as a structure of x and y coordinate values both defined as doubles.

See Also VCMouseMove, VCMouseMove2

VCMoveCursor

Version 1.2

Description Moves the cursor as if the user had pressed an arrow key.

Declaration

C/C++: extern "C" void WINAPI VCMoveCursor(short nVKey);

Visual Basic: Declare Sub VCMoveCursor Lib "VCMAIN32.DLL" (ByVal nVKey As Integer)

Delphi: procedure VCMoveCursor(nVKey: Integer); far;

Parameters *nVKey* - the Windows constant for the arrow keys as follows.
VK_DOWN moves the cursor down.
VK_RIGHT moves the cursor right.
VK_UP moves the cursor up.
VK_LEFT moves the cursor left.
VK_HOME moves the cursor up and left.
VK_END moves the cursor down and left.
VK_PRIOR moves the cursor left up and right.
VK_NEXT moves the cursor down and right.

Notes All arrow key movement is determined by the setting in the system tab regarding screen or world scale increments and the number assigned therein.

See Also VCGetCursorSize, VCGetCursorColor

VCNameView

Version	1.2
Description	Names the current view for later display.
Declaration	
C/C++:	extern "C" void WINAPI VCNameView(char* szView);
Visual Basic:	Declare Sub VCNameView Lib "VCMAIN32.DLL" (ByVal szView As String)
Delphi:	procedure VCNameView(szView: PChar); far;
Parameters	*szView* - the name for the view.
Notes	Named views are useful whenever a specific screen view needs to be accessed repeatedly for drawing or editing. The view is returned with VCZoomView.
See Also	VCZoomView

VCNewView

Version	2.0
Description	Creates a new view for the input drawing handle.
Declaration	
C/C++	extern "C" void WINAPI VCNewView(short* iError, long hWnd_);
Visual Basic	Declare Sub VCNewView Lib "VCMAIN32.DLL" (iError As Integer, ByVal hWnd_ As Long)
Delphi	procedure VCNewView(var iError: Integer; hWnd_: Longint); far;
Parameters	*hWnd* - the windows handle for the control to display the viewport.
Notes	Corel Visual CADD supports multiple viewports for drawings and displays the views in separate Window frames. These views are created through the API with VCNewView. When working with drawings utilizing multiple viewports, an application can parse through the views to update specific views as needed. The viewports are treated as separate MDI windows are managed by the Corel Visual CADD frame.
See Also	VCFirstView, VCNextView, VCZoomAllViews, VCZoomRegenAllViews

VCNewWorld

Version 1.2

Description Creates another instance of a "world" for Corel Visual CADD to create or modify a drawing.

Declaration

C/C++: extern "C" WORLDHANDLE WINAPI VCNewWorld(HWND hWnd_);

Visual Basic: Declare Function VCNewWorld Lib "VCMAIN32.DLL" (ByVal hWnd_ As Integer) As Long

Delphi: function VCNewWorld(hWnd_: Integer):Longint; far;

Parameters *hWnd* - the hWnd handle for the object to be used as the new world.
Returns - a long representing a Corel Visual CADD worldhandle.

Notes Whenever the application needs to create a new drawing area, such as in an MDI window, Corel Visual CADD needs to know the hWnd value for the object which is to contain the new drawing space. This creates a new environment for the drawing and exists entirely outside any other current drawings or worlds. The returned long is used by other Corel Visual CADD API calls to reference which world the call be affecting.

See Also VCDestroyWorld, VCIsWorldEmpty, VCIsWorldValid, VCGetCurrWorld, VCClearDrawing, VCClearDrawingNoPrompt, VCPackDataVCDrawToDC

VCNextEntity

Version 1.2

Description Positions a pointer for entity operations to the next entity in the database after the current one.

Declaration

C/C++: extern "C" vbool WINAPI VCNextEntity(short* iError, short* bKind);

Visual Basic: Declare Function VCNextEntity Lib "VCMAIN32.DLL" (iError As Integer, bKind As Integer) As Integer

Delphi: function VCNextEntity(var iError: Integer; var bKind: Integer):Boolean; far;

Parameters *bKind* - set by the function to what type of entity is now current.
Returns - 0 if not successful and 1 otherwise.

Notes Whenever querying entities for their particular properties, it is necessary to have a method to step through the drawing database and select which entity a given query will focus on. The API offer several utility parsing methods for flexibility in locating entities in the database. Each offers advantages in certain situations. VCFirst/NextEntity move through each entity in the drawing database. VCFirst/NextEntityExpand parses the database as if the drawing file had been exploded. Every entity, including those in symbol definition and hatch patterns are included in the search. VCFirst/NextOnScreen clip the drawing and allow for quick entity access to only those entities found in the current zoom. VCFirst/NextSelected parse only through the selection set. This method combined with a selection filter allow access to specific entities meeting a set of criteria quickly in the drawing database. If no entities exist for the method, the return value will be a 0 and the code can handle this case accordingly.

See Also VCFirstEntity, VCGetCurrentEntityHandle, VCLastEntity, VCFirstEntityExpand, VCNextEntityExpand, VCFirstEntity, VCNextOnScreen, VCFirstSelected, VCNextSelected, VCSetCurrentEntity

VCNextEntityExpand

Version 1.2

Description Positions a pointer for entity operations to the next entity in the database after the current one.

Declaration

C/C++: extern "C" vbool WINAPI VCNextEntityExpand(short* iError, short* bKind);

Visual Basic: Declare Function VCNextEntityExpand Lib "VCMAIN32.DLL" (iError As Integer, bKind As Integer) As Integer

Delphi: function VCNextEntityExpand(var iError: Integer; var bKind: Integer):Boolean;

Parameters *bKind* - set by the function to what type of entity is now current.
Returns - 0 if not successful and 1 otherwise.

Notes Whenever querying entities for their particular properties, it is necessary to have a method to step through the drawing database and select which entity a given query will focus on. The API offer several utility parsing methods for flexibility in locating entities in the database. Each offers advantages in certain situations. VCFirst/NextEntity move through each entity in the drawing database. VCFirst/NextEntityExpand parses the database as if the drawing file had been exploded. Every entity, including those in symbol definition and hatch patterns are included in the search. VCFirst/NextOnScreen clip the drawing and allow for quick entity access to only those entities found in the current zoom. VCFirst/NextSelected parse only through the selection set. This method combined with a selection filter allow access to specific entities meeting a set of criteria quickly in the drawing database. If no entities exist for the method, the return value will be a 0 and the code can handle this case accordingly.

See Also VCFirstEntity, VCLastEntity, VCFirstEntityExpand, VCFirstEntity, VCNextOnScreen, VCFirstSelected, VCNextSelected, VCSetCurrentEntity

VCNextGraphic

Version 2.0

Description Positions a pointer for entity operations to the next graphic in the entity.

Declaration

C/C++ extern "C" vbool WINAPI VCNextGraphic(short* iError, GRAPHICHANDLE hG);

Visual Basic Declare Function VCNextGraphic Lib "VCMAIN32.DLL" (iError As Integer, ByVal hG As Long) As Integer

Delphi function VCNextGraphic(var iError: Integer; hG: Longint):Boolean; far;

Parameters *hG* - the returned GRAPHICHANDLE for the current entity
Returns - 0 if not successful and 1 otherwise.

Notes Some entities are defined by several graphical objects, hatch patterns, fills, line types and fonts. For instance, a hatch pattern is defined by lines to make a useful pattern. These entities are not available for access through the standard database parsing routines provided. This is due to the fact that typically an application will not need this specific information. Most applications will need to simply parse the database and retrieve the entity information provided. In situations where a custom vector output file is being defined or to guide a CNC milling machine, the application may need to define all the vectors making up even the complex entities. The graphic handle method allow for this detailed parsing functionality.

In order to access the information an application should first create a graphics handle using VCCreateGraphicsHandle. This function creates a parsing list from the current entity if it is a graphic entity, hatch, fill, text or line type. The iError return will be > 0 if the current entity is not a graphic entity. The application can then parse the new set with VCFirstGraphic and VCNextGraphic. Any required information can be retrieved using any standard query function such as VCGetCurrentEntityPoint. The entity is considered read-only and only retrieval API routines may be utilized. The individual graphic entities can not be set with any command. After completing the parse the application should call VCDeleteGraphicHandle to destroy the created handle.

See Also

VCNextOnScreen

Version 1.2

Description Determines the next entity on screen and makes it current.

Declaration

C/C++: extern "C" vbool WINAPI VCNextOnScreen(short* iError, short* bKind);

Visual Basic: Declare Function VCNextOnScreen Lib "VCMAIN32.DLL" (iError As Integer, bKind As Integer) As Integer

Delphi: function VCNextOnScreen(var iError: Integer; var bKind: Integer):Boolean;

Parameters *bKind* - set by the function to what type of entity is now current.
Returns - 0 if not successful and 1 otherwise.

Notes Whenever querying entities for their particular properties, it is necessary to have a method to step through the drawing database and select which entity a given query will focus on. The API offer several utility parsing methods for flexibility in locating entities in the database. Each offers advantages in certain situations. VCFirst/NextEntity move through each entity in the drawing database. VCFirst/NextEntityExpand parses the database as if the drawing file had been exploded. Every entity, including those in symbol definition and hatch patterns are included in the search. VCFirst/NextOnScreen clip the drawing and allow for quick entity access to only those entities found in the current zoom. VCFirst/NextSelected parse only through the selection set. This method combined with a selection filter allow access to specific entities meeting a set of criteria quickly in the drawing database. If no entities exist for the method, the return value will be a 0 and the code can handle this case accordingly.

See Also VCFirstEntity, VCLastEntity, VCFirstEntityExpand, VCNextEntityExpand, VCFirstEntity, VCFirstOnScreen, VCFirstSelected, VCNextSelected, VCSetCurrentEntity

VCNextSelected

Version 1.2

Description Determines the next selected entity and makes it current.

Declaration

C/C++: extern "C" vbool WINAPI VCNextSelected(short* iError, short* bKind);

Visual Basic: Declare Function VCNextSelected Lib "VCMAIN32.DLL" (iError As Integer, bKind As Integer) As Integer

Delphi: function VCNextSelected(var iError: Integer; var bKind: Integer):Boolean;

Parameters *bKind* - set by the function to what type of entity is now current.
Returns - 0 if not successful and 1 otherwise.

Notes Whenever querying entities for their particular properties, it is necessary to have a method to step through the drawing database and select which entity a given query will focus on. The API offer several utility parsing methods for flexibility in locating entities in the database. Each offers advantages in certain situations. VCFirst/NextEntity move through each entity in the drawing database. VCFirst/NextEntityExpand parses the database as if the drawing file had been exploded. Every entity, including those in symbol definition and hatch patterns are included in the search. VCFirst/NextOnScreen clip the drawing and allow for quick entity access to only those entities found in the current zoom. VCFirst/NextSelected parse only through the selection set. This method combined with a selection filter allow access to specific entities meeting a set of criteria quickly in the drawing database. If no entities exist for the method, the return value will be a 0 and the code can handle this case accordingly.

See Also VCGetCurrentEntityHandle, VCNextEntity, VCLastEntity, VCFirstEntityExpand, VCNextEntityExpand, VCFirstEntity, VCNextOnScreen, VCFirstSelected, VCFirstEntity, VCSetCurrentEntity

VCNextSelectedRF

Version 2.0

Description Positions a pointer to the next entity in the given reference frame.

Declaration

C/C++ extern "C" vbool WINAPI VCFirstSelectedRF(short* iError, long* hE);

Visual Basic Declare Function VCFirstSelectedRF Lib "VCMAIN32.DLL" (iError As Integer, hE As Long) As Integer

Delphi function VCFirstSelectedRF(var iError: Integer; var hE: Longint):Boolean;

Parameters *hE* - the entity handle for the reference frame to parse.
Returns - 0 if not successful and 1 otherwise.

Notes Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. In order to add a reference frame entity an application must first set the drawing name to add as a reference entity with VCSetRefFrameName.

VCFirstSelectedRF and VCNextSelectedRF allow an application to parse the entities inside the reference frame. Any values returned for coordinates, using routines such as VCGetCurrentEntityPoint, are returned in values corresponding to the active drawing not the frame entity. For example if a real world drawing is referenced into a paper space drawing, the values returned will represent the coordinates for the entity in the paper space drawing not the absolute coordinates from the real world drawing. When the absolute coordinates are desired the referenced file must be opened and parsed with other standard database routines.

See Also VCFirstSelectedRF, VCGetCurrentEntityPoint

VCNextView

Version	2.0
Description	Moves to the next viewport for the active drawing.
Declaration	
C/C++	extern "C" vbool WINAPI VCNextView(short* iError);
Visual Basic	Declare Function VCNextView Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Delphi	function VCNextView(var iError: Integer):Boolean; far;
Parameters	*hWnd* - the windows handle for the control to display the viewport.
Notes	Corel Visual CADD supports multiple viewports for drawings and displays the views in separate Window frames. These views are created through the API with VCNewView. When working with drawings utilizing multiple viewports, an application can parse through the views to update specific views as needed. The viewports are treated as separate MDI windows are managed by the Corel Visual CADD frame.
See Also	VCFirstView, VCNextView, VCZoomAllViews, VCZoomRegenAllViews

VCNoDrawingSpeedbar

Version 2.0

Description Turns the display of the speedbar off.

Declaration

C/C++ extern "C" void WINAPI VCNoDrawingSpeedbar(short* iError);

Visual Basic Declare Sub VCNoDrawingSpeedbar Lib "VCDLG32.DLL" (iError As Integer)

Delphi procedure VCNoDrawingSpeedbar(var iError: Integer); far;

Parameters No additional parameters are used with this subroutine.

Notes The Corel Visual CADD interface is enhanced with context sensitive ribalogs. These ribalogs display setting information and prompt for input during a tool operation. An application can override the display of the ribalogs allowing for no user input during the tool operation. This allows the application to control the complete operation of the tool. For example, a copy tool may always need to make three (3) copies of the entity. Instead of ,launching the tool and hoping the user inputs the correct number, the application can set the number of copies through the API and turn off the display so the user can not override the values.

See Also

VCObjectSelect

Version	1.2
Description	Selects the object located at the designated point.
Declaration	
C/C++:	extern "C" void WINAPI VCObjectSelect(Point2D* dpP0);
Visual Basic:	Declare Sub VCObjectSelect Lib "VCMAIN32.DLL" (dpP0 As Point2D)
Delphi:	procedure VCObjectSelect(var dpP0: Point2D); far;
Parameters	*pP0* - the coordinates of the point from which to select an object.
Notes	Behaves exactly as the object select tool, however it can be called from the API.
See Also	VCSetCurrentSelected

VCOpenCMP

Version 1.2

Description Loads a Generic CADD component definition into the drawing session.

Declaration

C/C++: extern "C" void WINAPI VCOpenCMP(char* szFile);

Visual Basic: Declare Sub VCOpenCMP Lib "VCMAIN32.DLL" (ByVal szFile As String)

Delphi: procedure VCOpenCMP(szFile: PChar); far;

Parameters *szFile* - a string representing the path and name of the symbol to load.

Notes This subroutine only loads symbols with the .CMP file extension. This offers advantages over VCOpenSymbol since the file type does not have to be passed as a parameter. This call only loads the symbol into memory, the symbol is not placed until a place symbol command is executed. Loaded symbols are available to all drawings created in that session.

See Also VCOpenVCD, VCOpenVCA, VCOpenVCS, VCSaveVCS

VCOpenGCD

Version	1.2
Description	Opens a Generic CADD drawing into the current drawing session.
Declaration	
C/C++:	extern "C" void WINAPI VCOpenGCD(char* szFile);
Visual Basic:	Declare Sub VCOpenGCD Lib "VCMAIN32.DLL" (ByVal szFile As String)
Delphi:	procedure VCOpenGCD(szFile: PChar); far;
Parameters	*fname* - a string representing the path and name of the file to open.
Notes	This subroutine only loads files with the GCD extension into the current drawing session. This offers advantages over VCLoadDrawing since the file type does not have to be passed as a parameter.
See Also	VCLoadDrawing, VCOpenVCD, VCOpenVCS, VCOpenVCA, VCLoadDrawing

VCOpenStyle

Version	1.2
Description	Loads a style file into the current drawing session.
Declaration	
C/C++:	extern "C" void WINAPI VCOpenStyle(char* fname);
Visual Basic:	Declare Sub VCOpenStyle Lib "VCMAIN32.DLL" (ByVal fname As String)
Delphi:	procedure VCOpenStyle(fname: PChar); far;
Parameters	*fname* - a string representing the path and name for the style.
Notes	Styles are groups of settings stored in files for quick access. All the format settings included in this file are applied to any subsequent draw commands. This differs from VCSaveEnvironment in that the settings are not restored until a VCOpenStyle call.
See Also	VCSaveEnvironment, VCSaveStyle,

VCOpenVCA

Version	1.2
Description	Loads an attribute definition into Corel Visual CADD.
Declaration	
C/C++:	extern "C" void WINAPI VCOpenVCA(char* szFile);
Visual Basic:	Declare Sub VCOpenVCA Lib "VCMAIN32.DLL" (ByVal szFile As String)
Delphi:	procedure VCOpenVCA(szFile: PChar); far;
Parameters	*fname* - a string representing the path and name of the attribute to load.
Notes	This subroutine only loads attributes with the VCA file extension The attributes are not placed until an Attach or Embed Attribute command. This offers advantages over VCOpenAtb since the file type does not have to be passed as a parameter.
See Also	VCOpenVCA, VCOpenVCD, VCOpenVCS, VCSaveVCA

VCOpenVCD

Version	1.2
Description	Loads the specified Corel Visual CADD drawing into the current drawing session.
Declaration	
C/C++:	extern "C" void WINAPI VCOpenVCD(char* szFile);
Visual Basic:	Declare Sub VCOpenVCD Lib "VCMAIN32.DLL" (ByVal szFile As String)
Delphi:	procedure VCOpenVCD(szFile: PChar); far;
Parameters	*fname* - a string representing the path and name of the file to open.
Notes	This subroutine only loads files with the .VCD extension. This offers advantages over VCLoadDrawing since the file type does not have to be passed as a parameter.
See Also	VCLoadDrawing, VCOpenVCD, VCOpenGCD, VCOpenVCS, VCOpenVCA

VCOpenVCS

Version	1.2
Description	Loads a symbol definition into Corel Visual CADD.
Declaration	
C/C++:	extern "C" void WINAPI VCOpenVCS(char* szFile);
Visual Basic:	Declare Sub VCOpenVCS Lib "VCMAIN32.DLL" (ByVal szFile As String)
Delphi:	procedure VCOpenVCS(szFile: PChar); far;
Parameters	*fname* - a string representing the path and name of the symbol to load.
Notes	This subroutine only loads symbols with the VCS file extension. This offers advantages over VCOpenSymbol since the file type does not have to be passed as a parameter. This call only loads the symbol into memory, the symbol is not placed until a place symbol command is executed. Loaded symbols are available to all drawings created in that session.
See Also	VCOpenVCS, VCOpenVCS, VCOpenVCD, VCOpenCMP, VCOpenVCA, VCSaveVCS

VCOleClassMethodInvoke

Version 2.0

Description Registers and invokes a function from an OLE DLL.

Declaration

C/C++ extern "C" void WINAPI VCOleClassMethodInvoke(short* iError, char* DllName, char* ClassName, char* MethodName, char* CmdLine);

Visual Basic Declare Sub VCOleClassMethodInvoke Lib "VCMAIN32.DLL" (iError As Integer, ByVal DllName As String, ByVal ClassName As String, ByVal MethodName As String, ByVal CmdLine As String)

Delphi procedure VCOleClassMethodInvoke(var iError: Integer; DllName: PChar; ClassName: PChar; MethodName: PChar; CmdLine: PChar); far;

Parameters *DllName* - the name of the DLL containing the OLE class.
ClassName - the name of the class contained in the DLL.
MethodName - the member function name contained in the DLL.
CmdLine - a command line string for any input arguments.

Notes An application can be created as an EXE, a Windows DLL or an OLE DLL. Each has advantages in functionality and interaction with the CAD engine. In addition, each is accessed through the Corel Visual CADD interface in different methods. An OLE DLL is a specialized link library containing methods and classes for controlling various operations. These DLL are specifically related to Visual Basic programmers. The OLE class allows a developer to create a class member function that can be directly run from the Corel Visual CADD interface allowing an application to take advantage of the performance increase associated with a DLL. In order to access this functionality the DLL and the class must be registered. VCCreateOLEClass registers the DLL and class. VCInvokeMethod will invoke the DLL method and VCDeleteOleClass will delete the registered DLL and class.

See Also VCCreateOleClass, VCDeleteOleClass

VCPackData

Version 1.2

Description Removes all erased entities or unused definitions from the drawing database.

Declaration

C/C++: extern "C" void WINAPI VCPackData(WORLDHANDLE hW);

Visual Basic: Declare Sub VCPackData Lib "VCTOOL32.DLL" (ByVal hW As Long)

Delphi: procedure VCPackData(hW: Longint); far;

Parameters *hW* - the Corel Visual CADD worldhandle of the drawing to remove erased entities.

Notes Simply executes the command as if the user had selected it from the menu. Any entities erased or marked as erased by a modify command remains in the drawing to enable the undo and redo commands. While maintaining these can get quite cumbersome with limited memory resources, using pack data will remove these erased entities from the database and free up the memory taken by them. Immediately after a pack data, undo will have nothing to undo.

See Also VCClearDrawingNoPrompt, VCPurgeErasedEntities

VCPaint

Version 1.2

Description Repaints the area enclosed by the specified rectangle.

Declaration

C/C++: extern "C" void WINAPI VCPaint(HWND hWnd, RECT rc);

Visual Basic: Declare Sub VCPaintWorld Lib "VCMAIN32.DLL" ()

Delphi: procedure VCPaintWorld; far;

Parameters *hWnd* - the hWnd handle for the object to be used as the new world.
Rc - a Windows RECT structure used to define the boundary area.

Notes Similar to VCPaintWorld except that it only paints a specified rectangle. This is particularly helpful when a portion of the screen has been covered by a dialog box and now needs to be repainted.

See Also VCPaintWorld, VCPaintWorld, VCInvalidateRect

VCPaintWorld

Version	1.2
Description	Forces Corel Visual CADD to repaint the current world immediately.
Declaration	
C/C++:	extern "C" void WINAPI VCPaintWorld(void);
Visual Basic:	Declare Sub VCPaintWorld Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCPaintWorld; far;
Parameters	No parameters are used for this subroutine.
Notes	Unlike VCInvalidateRect, VCPaintWorld forces a repaint of the drawing area immediately instead of waiting for the next WM_PAINT message.
See Also	VCInvalidateRect, VCPaint, VCInvalidateRect

VCPastable

Version 1.2

Description Determines if the current selection in the Windows clipboard is suitable for pasting into Corel Visual CADD.

Declaration

C/C++: extern "C" vbool WINAPI VCPastable();

Visual Basic: Declare Function VCPastable Lib "VCTOOL32.DLL" () As Integer

Delphi: function VCPastable:Boolean; far;

Parameters *Returns* -a true or false representing whether the clipboard contents can be pasted into Corel Visual CADD.
0 - false.
1 - true.

See Also VCCopy, VCPaste

VCPopupButton

Version 1.2

Description Pops the context sensitive mouse menu at a specified location on screen.

Declaration

C/C++: extern "C" void WINAPI VCPopupButton(long lParam, WORD wParam);

Visual Basic: Declare Sub VCPopupButton Lib "VCTOOL32.DLL" (ByVal lParam As Long, ByVal wParam As Integer)

Delphi: procedure VCPopupButton(lParam: Longint; wParam: Integer); far;

Parameters *lParam* - a Windows parameter containing the coordinates where the menu should appear.
wParam - the corresponding message.

Notes In cases where the mouse menu is required but not available via the second mouse button, VCPopupButton will display the menu without having to use the mouse buttons.

See Also

VCPostMessage

Version 1.2

Description The PostMessage function posts (places) a message in a Windows message queue and then returns without waiting for Corel Visual CADD to process the message.

Declaration

C/C++: extern "C" vbool WINAPI VCPostMessage(WORD iMessage, WORD wParam, LPARAM lParam);

Visual Basic: Declare Function VCPostMessage Lib "VCMAIN32.DLL" (ByVal iMessage As Integer, ByVal wParam As Integer, ByVal lParam As Long) As Integer

Delphi: function VCPostMessage(iMessage: Integer; wParam: Integer; lParam Longint):Boolean; far;

Parameters *iMessage* - the message to post.
wParam - Specifies 16 bits of additional message-dependent information.
lParam - Specifies 32 bits of additional message-dependent information.
Returns - value is nonzero if the function is successful. Otherwise, it is zero.

Notes Windows frequently sends or posts messages to any running application depending on current focus, windows positioning, or current system activity. When programming an external application, these messages will be received from the system and need to be passed on to the Corel Visual CADD engine. In many cases there is a specific API call to do this, such as VCTimer, but there are many more that may need to be sent or relayed to Corel Visual CADD. This is the function used to do this. The wParam and lParam context will come from the message received from the system and typically be the message received. If the message is being posted to another application, and the wParam or lParam parameters are used to pass a handle or pointer to a global memory object, the memory should be allocated by the GlobalAlloc function, using the GMEM_SHARE flag. The PostMessage function fails if the message queue for the receiving application is full. This is especially likely if an application posts several messages without allowing the receiving task to run.

See Also VCTimer

VCPurgeErasedEntities

Version	1.2
Description	Removes all erased entities from the drawing.
Declaration	
C/C++:	extern "C" void WINAPI VCPurgeErasedEntities(WORLDHANDLE hW);
Visual Basic:	Declare Sub VCPurgeErasedEntities Lib "VCMAIN32.DLL" (ByVal hW As Long)
Delphi:	procedure VCPurgeErasedEntities(hW: Longint); far;
Parameters	*hW* - the Corel Visual CADD worldhandle used internally to reference each open drawing world.
Notes	Any draw or modify command changes the Corel Visual CADD drawing database. Visual CADD keeps track of these changes by "marking" the items that have been changed but does not remove them from the database. This allows the Undo and Redo operations to restore the drawing. Maintaining the copies of the entities however takes up memory in the database. VCPurgeErasedEntities should be used to "clean" out these copies and free up memory resources.
See Also	VCPackData

VCRButtonDown

Version 1.2

Description Sends a right button down message to Corel Visual CADD effectively selecting a coordinate for a tool, or selecting an entity.

Declaration

C/C++: extern "C" void WINAPI VCRButtonDown(long lParam, WORD wParam);

Visual Basic: Declare Sub VCRButtonDown Lib "VCTOOL32.DLL" (ByVal lParam As Long, ByVal wParam As Integer)

Delphi: procedure VCRButtonDown(lParam: Longint; wParam: Integer); far;

Parameters *lParam* - a packed coordinate pair as used by Windows.
wParam - passed by Windows functions to represent the identifier of the mouse message.

Notes All mouse subroutines as passed by Windows function pass the coordinate values through a lParam structure which contains the x and y coordinates and other pertinent information not used by Corel Visual CADD. Also included is a wParam which contains any modifiers to the mouse movement such as the state of the shift, ctrl, and alt keys These may or may not be used by Visual CADD to modify the results of the mouse movements depending on the current context in the application. Whenever an external application receives a right button down message in the drawing area, the application should send the VCRButtonDown message to Corel Visual CADD in order to invoke the expected response. This makes if behave as if it were in the Corel Visual CADD drawing area.

The MFC Class Library in MS Visual C++ references mouse movements and points through a CPoint class structure. The macro MAKELPARAM can be used to convert the given CPoint structure to a LPARAM compatible with the Corel Visual CADD API.

See Also VCButton, VCLButtonDblClk, VCLButtonDown, VCLButtonDown2, VCLButtonDownWorldPoint, VCLButtonUpVCRButtonUp

VCRButtonUp

Version 1.2

Description Issues a right mouse button up message to Corel Visual CADD.

Declaration

C/C++: extern "C" void WINAPI VCRButtonUp(long lParam, WORD wParam);

Visual Basic: Declare Sub VCRButtonUp Lib "VCMAIN32.DLL" (ByVal lParam As Long, ByVal wParam As Integer)

Delphi: procedure VCRButtonUp(lParam: Longint; wParam: Integer); far;

Parameters *lParam* - a packed coordinate pair as used by Windows.
wParam - passed by Windows functions to represent the identifier of the mouse message.

Notes All mouse subroutines as passed by Windows function pass the coordinate values through a lParam structure which contains the x and y coordinates and other pertinent information not used by Corel Visual CADD. Also included is a wParam which contains any modifiers to the mouse movement such as the state of the shift, ctrl, and alt keys These may or may not be used by Visual CADD to modify the results of the mouse movements depending on the current context in the application. Whenever an external application receives a right button up message in the drawing area, the application should send the VCRButtonUp message to Corel Visual CADD in order to invoke the expected response. This makes if behave as if it were in the Corel Visual CADD drawing area.

The MFC Class Library in MS Visual C++ references mouse movements and points through a CPoint class structure. The macro MAKELPARAM can be used to convert the given CPoint structure to a LPARAM compatible with the Corel Visual CADD API.

See Also VCButton, VCLButtonDblClk, VCLButtonDown, VCLButtonDown2, VCLButtonDownWorldPoint, VCLButtonUp

VCRelativePath

Version	2.0
Description	Returns the relative path defining the location of a reference frame entity file.
Declaration	
C/C++	extern "C" void WINAPI VCRelativePath(short* iError, char* ReturnPath, char* CurrPath, char* AbsPath);
Visual Basic	Declare Sub VCRelativePath Lib "VCMAIN32.DLL" (iError As Integer, ByVal ReturnPath As String, ByVal CurrPath As String, ByVal AbsPath As String)
Delphi	procedure VCRelativePath(var iError: Integer; ReturnPath: PChar; CurrPath:
Parameters	*ReturnPath* - the returned relative path. *CurrPath* - the path for the current drawing that is to have the reference frame. *AbsPath* - the absolute path to the file being referenced.
Notes	Reference frames allow external files to be linked into an existing drawing. When linked, the files are represented by a relative path between the current file location and the absolute path to the file. For example, if the current active drawing for an open VCD files is "C:\VCADD\SAMPLES\THISFILE.VCD" and a file is referenced into this drawing located at an absolute location of "C:\VCADD\LINKEDFILE.VCD" this routine will return the difference of the paths. In this case it will return " ..\" or indication that the linked file is located back one subdirectory. The routine can be used to retrieve the relative path for any given directory. Simply pass in a current directory(where the active drawing is) and the absolute path the linked file(file that is being referenced) and the routine will return the relative path for the directories.
See Also	VCAddRefFrameEntityVCChangeRefFrameName, VCGetRefFrameName, VCGetRefFrame, VCGetRefFrameIsDataBound, VCGetRefFrameIsDynamic

VCRemoveAllViews

Version	2.0
Description	Removes all multiple viewports for the given drawing.
Declaration	
C/C++	extern "C" vbool WINAPI VCRemoveAllViews(short* iError, WORLDHANDLE hW);
Visual Basic	Declare Function VCRemoveAllViews Lib "VCMAIN32.DLL" (iError As Integer, ByVal hW As Long) As Integer
Delphi	function VCRemoveAllViews(var iError: Integer; hW: Longint):Boolean; far;
Parameters	*hW* - the WORLDHANDLE for the drawing with multiple viewports.
Notes	Drawing worlds are referenced by a WORLDHANDLE which is an internal 0 based index for managing the drawing worlds. When working with any command to affect a drawing world you will pass this index to the routine. The active drawing index can be determined with VCGetCurrWorld and set with VCSetCurrWorld.
See Also	VCGetCurrWorld, VCRemoveView, VCNewView

VCRemoveFileLock

Version	2.0
Description	Removes all file locking for the given file.
Declaration	
C/C++	extern "C" void WINAPI VCRemoveFileLock(char* szFile);
Visual Basic	Declare Sub VCRemoveFileLock Lib "VCMAIN32.DLL" (ByVal szFile As String)
Delphi	procedure VCRemoveFileLock(szFile: PChar); far;
Parameters	*szFile* - the path and file name for the file to unlock.
Notes	Locked files can not be modified by another Corel Visual CADD user on a network until the drawing is saved or closed. Other users can only open, view and copy the drawing. The user name is taken form the registered user name stored in the registry for the installed machine.
See Also	VCIsFileLocked, VCIsFileLockedByCurrentUser, VCLockFile

VCRemovePlotterPageSize

Version	2.0
Description	Removes a plotter page size from the direct plot options.
Declaration	
C/C++	extern "C" void WINAPI VCRemovePlotterPageSize(short* iError, short iIndex);
Visual Basic	Declare Sub VCRemovePlotterPageSize Lib "VCDLG32.DLL" (iError As Integer, ByVal iIndex As Integer)
Delphi	procedure VCRemovePlotterPageSize(var iError: Integer; iIndex: Integer); far;
Parameters	*iIndex* - the zero based index for the page size item to remove.
Notes	Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device. The direct plot routine allows for custom page sizes to be defined with the VCAddPlotterPageSizeRoutine and by the user through the Corel Visual CADD interface. These can be removed from the interface by the user or through the API with VCRemovePlotterPageSize and added with VCAddPlotterPageSize. Custom page sizes enhance the users control over vector output devices and allows the user or an application to set page parameters suited to a desired output.
See Also	VCAddPlotterPageSize, VCAddPlotterPenMapName, VCAddPlotterLanguageName, VCAddPlotter

VCRemoveSymbols

Version 1.2

Description Removes the list of symbols from the drawing database, including all placements.

Declaration

C/C++: extern "C" void WINAPI VCRemoveSymbols(short* iError, char* pNames, short iCnt);

Visual Basic: Declare Sub VCRemoveSymbols Lib "VCMAIN32.DLL" (iError As Integer, ByVal pNames As String, ByVal iCnt As Integer)

Delphi: procedure VCRemoveSymbols(var iError: Integer; pNames: PChar; iCnt: Integer);

Parameters *pNames* - a pointer to a delimited string of names of symbols names to be removed. The delimeter is the "|" character.
iCnt - the count of symbols to be removed and the size of the array.

Notes As a user or external application loads several symbols into Corel Visual CADD, it may become increasingly low on memory. To alleviate this problem it may be necessary to remove unused symbol definitions from memory. It is also possible to remove all placements of a symbol by using this subroutine. A symbol definition can have two unique naming conventions. An on disk name used when saved to file(limited to the characters defined by the operating system) and an internal name used to store the name in a Corel Visual CADD drawing session. VCRemoveSymbols require the internal name not the on disk name. The internal name can be determined from the saved name with VCGetSymbolInternalName.

See Also VCReplaceSymbol VCUnloadUnusedSymDefs

VCRemoveView

Version	2.0
Description	Removes a specific viewport from a drawing world.
Declaration	
C/C++	extern "C" vbool WINAPI VCRemoveView(short* iError, WORLDHANDLE hW, long hWnd);
Visual Basic	Declare Function VCRemoveView Lib "VCMAIN32.DLL" (iError As Integer, ByVal hW As Long, ByVal hWnd As Long) As Integer
Delphi	function VCRemoveView(var iError: Integer; hW: Longint; hWnd:
Parameters	*hW* - the WORLDHANDLE for the drawing. *HWnd* - the Windows handle for the viewport to remove.
Notes	Drawing world are referenced by a WORLDHANDLE which is an internal 0 based index for managing the drawing worlds. When working with any command to affect a drawing world you will pass this index to the routine. The active drawing index can be determined with VCGetCurrWorld and set with VCSetCurrWorld.
See Also	VCNewView, VCFirstView, VCNextView

VCReplaceSymbol

Version	1.2
Description	Replaces all placements of a symbol with another loaded symbol.
Declaration	
C/C++:	extern "C" void WINAPI VCReplaceSymbol(char* szFrom, char* szTo, vbool tfSelectedOnly);
Visual Basic:	Declare Sub VCReplaceSymbol Lib "VCTOOL32.DLL" (ByVal szFrom As String, ByVal szTo As String, ByVal tfSelectedOnly As Integer)
Delphi:	procedure VCReplaceSymbol(szFrom: PChar; szTo: PChar; tfSelectedOnly Boolean); far;
Parameters	*szFrom* - the name of the symbol to be replaced. *szTo* - the name of the symbol to replace with. *tfSelectedOnly* - states whether to replace all placements or just those that are selected. 0 - all placements. 1 - selected placements.
Notes	VCReplaceSymbol conveniently replaces all symbols with another symbol definition while leaving rotation, scale factors and layer information all intact from the original symbol placement. A symbol definition can have two unique naming conventions. An on disk name used when saved to file(limited to the characters defined by the operating system) and an internal name used to store the name in a Corel Visual CADD drawing session. VCReplaceSymbol require the internal name not the on disk name. The internal name can be determined from the saved name with VCGetSymbolInternalName.
See Also	VCRemoveSymbolsVCDuplicate

VCResetCmdExt

Version	2.0
Description	Reloads the CMDEXT.DEF containing Corel Visual CADD custom commands.
Declaration	
C/C++	extern "C" void WINAPI VCResetCmdExt(short* iError);
Visual Basic	Declare Sub VCResetCmdExt Lib "VCDLG32.DLL" (iError As Integer)
Delphi	procedure VCResetCmdExt(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	Corel Visual CADD utilizes a custom command file CMDEXT.DEF to load custom commands not directly available through the Corel Visual CADD interface. This file is described in detail in the Customizing Corel Visual CADD section. The file is read on application startup and must b "forced" to update by an external application using this command.
See Also	VCLoadAlias, VCLoadCmdExt

VCResetOnScreenList

Version	1.2
Description	Resets the parsing list for the VCFirstOnScreen and VCNextOnScreen routines.
Declaration	
C/C++:	extern "C" void WINAPI VCResetOnScreenList(short* iError);
Visual Basic:	Declare Sub VCResetOnScreenList Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCResetOnScreenList(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine..
Notes	When using the VCFirstOnScreen and VCNextOnScreen commands to parse the database, it is necessary to update the list when the view changes during an operation. This resets the list and the parsing can continue based on the new entities.
See Also	VCNextEntity, VCLastEntity, VCFirstEntityExpand, VCNextEntityExpand, VCFirstOnScreen, VCNextOnScreen, VCFirstSelected, VCNextSelected, VCSetCurrentEntity

VCResetPrintMargins

Version	2.0
Description	Resets the print margins to the default values read from the Windows driver.
Declaration	
C/C++	extern "C" void WINAPI VCResetPrintMargins(short* iError);
Visual Basic	Declare Sub VCResetPrintMargins Lib "VCDLG32.DLL" (iError As Integer)
Delphi	procedure VCResetPrintMargins(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	The Corel Visual CADD print routine utilizes the standard Windows print/plot drivers. These drivers are set with default values based on specifications by the plotter manufacturer. A user or an application can modify these margins in order to achieve a desired output. These can then be reset to the default values with VCResetPrintMargins.
See Also	VCGetPrintSettings

VCResizeChildWindow

Version	1.2
Description	Resize the current child (MDI) window within Corel Visual CADD. Used when the application receives a WM_SIZE message.
Declaration	
C/C++:	extern "C" void WINAPI VCResizeChildWindow();
Visual Basic:	Declare Sub VCResizeChildWindow Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCResizeChildWindow; far;
Parameters	No Parameters are used in this subroutine.
Notes	Whenever an applications window is resized, the system passes a WM_SIZE message to the application. In order for Corel Visual CADD to resize and calculate the drawing view an application needs to use Corel Visual CADD VCResizeChildWindow subroutine.
See Also	Windows SDK

VCRestoreSettings

Version	1.2
Description	Restores all of the current settings from a temporary memory buffer.
Declaration	
C/C++:	extern "C" void WINAPI VCRestoreSettings();
Visual Basic:	Declare Sub VCRestoreSettings Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCRestoreSettings; far;
Parameters	No additional parameters are used with this subroutine.
Notes	Before using the VCMatchCurrentEntity command, save the current settings with VCSaveSettings and restore them with VCRestoreSettings.
See Also	VCSaveSettings, VCSaveSysSettings, VCRestoreSysSettings

VCRestoreSysSettings

Version	1.2.1
Description	Restores all of the system settings (everything but the entity-specific data) from a temporary memory buffer.
Declaration	
C/C++:	extern "C" void WINAPI VCRestoreSysSettings(void);
Visual Basic:	Declare Sub VCRestoreSysSettings Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCRestoreSysSettings; far;
Parameters	No parameters are used with this subroutine.
Notes	After retrieving all relevant information from the system settings following a VCMatchCurrentEntity, it is necessary to retrieve all the valid current settings and re-establish all the user settings as current.
See Also	VCSaveSettings, VCSaveSysSettings, VCRestoreSettings

VCRFGetDrawBoundary
VCRefFrameSetDrawBoundary

Version 2.0

Description Reference frames can be identified with a bounding rectangle.

Declaration

C/C++
```
extern "C" vbool WINAPI VCRFGetDrawBoundary(short* iError);
extern "C" void WINAPI VCSetRefFrameDrawBoundary(short* iError, vbool vb);
```

Visual Basic
```
Declare Function VCRFGetDrawBoundary Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Declare Sub VCSetRefFrameDrawBoundary Lib "VCMAIN32.DLL" (iError As Integer, ByVal vb As Integer)
```

Delphi
```
function VCRFGetDrawBoundary(var iError: Integer):Boolean; far;
procedure VCSetRefFrameDrawBoundary(var iError: Integer; vb: Boolean); far;
```

Parameters *vb* - determines if the reference frame boundary is shown.
0 - the boundary is not displayed.
1 - the boundary is displayed.

Notes Reference frame entities enable a drawing file to be referenced or linked into another drawing. The frames can be used to layout drawings for printing or to create overlay patterns. The reference frame can be bound, data is not dynamic and is stored in the parent drawing, or dynamic in which the referenced file is updated as changes are made to the original.

When linked, the files are represented by a relative path between the current file location and the absolute path to the file. For example, if the current active drawing for an open VCD files is "C:\VCADD\SAMPLES\THISFILE.VCD" and a file is referenced into this drawing located at an absolute location of "C:\VCADD\LINKEDFILE.VCD" VCRelativePath will return the difference of the paths. In this case it will return " ..\" or indication that the linked file is located back one subdirectory.

The reference frame, the actual border around the linked file, behaves as a primitive entity with color, rotation, scale and other properties. All these can be used to manipulate the frame for displaying the desired data.

To add a reference frame, the application should first set a pointer to the file being referenced with VCSetRefFrameName. VCAddRefFrameEntity will then reference this file in at the current position.

See Also VCRFGetTransparent, VCAddRefFrameEntityVCGetRefFrameDrawBoundary, VCGetRefFrameColor, VCGetRefFrame

VCRFGetTransparent
VCRFSetTransparent

Version 2.0

Description Transparent reference frames allow grids and other entities to be viewed through the reference frame.

Declaration

C/C++ extern "C" vbool WINAPI VCRFGetTransparent(short* iError);

Visual Basic Declare Function VCRFGetTransparent Lib "VCMAIN32.DLL" (iError As Integer) As Integer

Delphi function VCRFGetTransparent(var iError: Integer):Boolean; far;

Parameters *vb* - determines if the reference frame is transparent.
0 - the boundary is not displayed.
1 - the boundary is displayed.

Notes Reference frame entities enable a drawing file to be referenced or linked into another drawing. The frames can be used to layout drawings for printing or to create overlay patterns. The reference frame can be bound, data is not dynamic and is stored in the parent drawing, or dynamic in which the referenced file is updated as changes are made to the original.

When linked, the files are represented by a relative path between the current file location and the absolute path to the file. For example, if the current active drawing for an open VCD files is "C:\VCADD\SAMPLES\THISFILE.VCD" and a file is referenced into this drawing located at an absolute location of "C:\VCADD\LINKEDFILE.VCD" VCRelativePath will return the difference of the paths. In this case it will return " ..\" or indication that the linked file is located back one subdirectory.

The reference frame, the actual border around the linked file, behaves as a primitive entity with color, rotation, scale and other properties. All these can be used to manipulate the frame for displaying the desired data.

To add a reference frame, the application should first set a pointer to the file being referenced with VCSetRefFrameName. VCAddRefFrameEntity will then reference this file in at the current position.

See Also VCRFGetDrawBoundary, VCAddRefFrameEntityVCGetRefFrameDrawBoundary, VCGetRefFrameColor, VCGetRefFrame

VCRFUpdateFileLink

Version 2.0

Description Forces a reference frame entity to update file information.

Declaration

C/C++ extern "C" void WINAPI VCRFUpdateFileLink(short* iError);

Visual Basic Declare Sub VCRFUpdateFileLink Lib "VCMAIN32.DLL" (iError As Integer)

Delphi procedure VCRFUpdateFileLink(var iError: Integer); far;

Parameters No additional parameters are used with this subroutine.

Notes Reference frame entities enable a drawing file to be referenced or linked into another drawing. The frames can be used to layout drawings for printing or to create overlay patterns. The reference frame can be bound, data is not dynamic and is stored in the parent drawing, or dynamic in which the referenced file is updated as changes are made to the original.

When linked, the files are represented by a relative path between the current file location and the absolute path to the file. For example, if the current active drawing for an open VCD files is "C:\VCADD\SAMPLES\THISFILE.VCD" and a file is referenced into this drawing located at an absolute location of "C:\VCADD\LINKEDFILE.VCD" VCRelativePath will return the difference of the paths. In this case it will return " ..\" or indication that the linked file is located back one subdirectory.

The reference frame, the actual border around the linked file, behaves as a primitive entity with color, rotation, scale and other properties. All these can be used to manipulate the frame for displaying the desired data.

To add a reference frame, the application should first set a pointer to the file being referenced with VCSetRefFrameName. VCAddRefFrameEntity will then reference this file in at the current position.

Binding a reference frame entity into a drawing inserts the contents directly into the file. The referenced data may not be updated to reflect changes. Linking a reference frame into a drawing keeps the file size to a minimum and updates the contents of the frame based on changes to the contents of the origin. Corel Visual CADD automatically updates these links when the file is loaded and saved. An application may however need to force the reference file to update to reflect changes immediately.

See Also VCAddRefFrameEntity, VCGetRefFrame, VCGetRefFrameName

VCRIsButtonDown

Version	2.0
Description	Used to determine the toggle state for a control.
Declaration	
C/C++	extern "C" vbool WINAPI VCRIsButtonDown(short* iError, char* szNative, WORD id);
Visual Basic	Declare Function VCRIsButtonDown Lib "VCDLG32.DLL" (iError As Integer, ByVal szNative As String, ByVal id As Integer) As Integer
Delphi	function VCRIsButtonDown(var iError: Integer; szNative: PChar; id: Integer):Integer;far;
Parameters	*szNative* - native command name for the tool - currently ignored. *Id* - the Tool ID for the toggle control.
Notes	Returns a value indicating if the control is on. Generally there are specific routines for retrieving the state of a toggle setting.
See Also	VCToggle

VCRun

Version	1.2
Description	Runs the specified application.
Declaration	
C/C++:	extern "C" void WINAPI VCRun(char* szName);
Visual Basic:	Declare Sub VCRun Lib "VCTOOL32.DLL" (ByVal szName As String)
Delphi:	procedure VCRun(szName: PChar); far;
Parameters	*szname* - name of the file to run
Notes	VCRun is used to run an external application from Corel Visual CADD, whether a Visual CADD specific program or a unrelated program. The program to be run is specified from within Corel Visual CADD with the EXENAME parameter within a script or thorough the API with VCSetExeName. This can also be retrieved using VCGetExeName. VCRun will abort any current tool.
See Also	VCDllRun, VCGetExeName, VCRunNested

VCRunNested

Version	1.2
Description	Runs the specified application as a nested tool.
Declaration	
C/C++:	extern "C" void WINAPI VCRunNested(char* szName);
Visual Basic:	Declare Sub VCRunNested Lib "VCTOOL32.DLL" (ByVal szName As String)
Delphi:	procedure VCRunNested(szName: PChar); far;
Parameters	*szName* - the application name and path.
Notes	VCRunNested. unlike VCRun, executes the specified application as a nested tool instead of a new tool. Nested tools do not interfere with any currently running tools, that is the current tool remains active until the nested tool is complete and then continues operation from where the nesting occurred.
See Also	VCDllRun, VCGetExeName,VCRun

VCSaveCurrent3DViewto2D

Version	2.0.1
Description	Saves the current 3D view to a projected 2D plane.
Declaration	
C/C++	extern "C" void WINAPI VCSaveCurrent3DViewTo2D(short* iError, WORLDHANDLE TargetWorld);
Visual Basic	Declare Sub VCSaveCurrent3DViewTo2D Lib "VCMAIN32.DLL" (iError As Integer, ByVal TargetWorld As Long)
Delphi	procedure VCSaveCurrent3DViewTo2D(var iError: Integer; TargetWorld: Longint);far;
Parameters	*TargetWorld* - the drawing world to display the 2D projection.
Notes	When creating 3D views of a drawing, three parameters are required: view type, eye location, and viewed position. VCSetProjection3D determines the view type and thus how the lines will be viewed in relation to each other, that is flat, parallel or perspective. VCSetView3D establishes the absolute 3D coordinate of the viewers eye and thus the level of perspective exaggeration used or the relative size of the view. VCChangeView3D can allow the users view point to be moved incrementally in certain directions and thus creates a limited "walk-through" functionality. 3D views can be viewed in wireframe or with Corel Visual CADD built in quick shading. VCSet3DDisplay provides the ability to view the drawing as a quick shade and VCSet3DQShadeOptions determines the level of quick shade when the drawing is shaded. After setting a 3D view through the target and eye position method, an application can project the view to a 2D drawing. VCSaveCurrent3DViewTo2D takes the current 3D view and display it to a specified drawing world as a 2D projection.
See Also	VCSaveDrawing, VCSetProjection3D,

VCSaveDrawing

Version 1.2

Description Saves the current drawing and converts if necessary.

Declaration

C/C++: extern "C" void WINAPI VCSaveDrawing(short* iError, char* pName, short iFileType, vbool tfSaveSelected);

Visual Basic: Declare Sub VCSaveDrawing Lib "VCTRAN32.DLL" (iError As Integer, ByVal pName As String, ByVal iFileType As Integer, ByVal tfSaveSelected As Integer)

Delphi:

Parameters *pName* - the name and path of the file to be loaded
iFileType - the type of drawing file that is to be loaded.
-1 - Determine By Extension
0 - FILE_VCD
3 - FILE_GCD
5 - FILE_DWG
6 - FILE_DXF

Notes If a drawing was originally from AutoCAD, there may be some changes to the final drawing if particular entities were used while editing in Corel Visual CADD. Some of these entities include ellipses, fills, and hatches due to the fact that AutoCAD doesn't support these entities directly or their handling of them is different. The same is also true Generic CADD files although on a smaller scale. Generic CADD does not support continuous lines so all entities made of continuous lines will be converted to single lines. Although these entities may be supported differently in other packages, they will not be lost but instead converted to the closest possible entity type available in the other file format.

See Also VCLoadDrawing, VCMergeDrawing VCLoadVCDFromFile, VCAcadRead, VCAcadReadWith3D

VCSaveEnvironment

Version	1.2
Description	Saves the Corel Visual CADD drawing environment.
Declaration	
C/C++:	extern "C" void WINAPI VCSaveEnvironment(short* iError);
Visual Basic:	Declare Sub VCSaveEnvironment Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSaveDrawing(var iError: Integer; pName: PChar; iFileType: Integer;
Parameters	No parameters are used with this subroutine.
Notes	Saves the current drawing settings in the default environment into a style file. These settings are then available to the user during the next drawing session or from drawing to drawing. This differs from VCSaveStyle in that it is automatically loaded with each new session or drawing.
See Also	VCSaveStyle, VCOpenStyle, VCSaveSettings, VCRestoreSettings, VCSaveSysSettings, VCRestoreSysSettings

VCSavePlotterDriver

Version	2.0
Description	Saves the current plotter driver settings to file.
Declaration	
C/C++	extern "C" void WINAPI VCSavePlotterDriver(short* iError, char* szName);
Visual Basic	Declare Sub VCSavePlotterDriver Lib "VCDLG32.DLL" (iError As Integer, ByVal szName As String)
Delphi	procedure VCSavePlotterDriver(var iError: Integer; szName: PChar); far;
Parameters	szName - the name to save the current driver settings under
Notes	Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device. The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.
See Also	VCAddPlotter

VCSavePlotterLanguage

Version	2.0
Description	Saves the current plotter language settings to file.
Declaration	
C/C++	extern "C" void WINAPI VCSavePlotterLanguage(short* iError, char* szName);
Visual Basic	Declare Sub VCSavePlotterLanguage Lib "VCDLG32.DLL" (iError As Integer, ByVal szName As String)
Delphi	procedure VCSavePlotterLanguage(var iError: Integer; szName: PChar); far;
Parameters	*szName* - the name to save the current plotter langage settings under
Notes	Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device. The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter. Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs to be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type
See Also	VCAddPlotterLanguageName

VCSavePlotterPenMap

Version	2.0
Description	Saves the current pen map settings to file.
Declaration	
C/C++	extern "C" void WINAPI VCSavePlotterPenMap(short* iError, char* szName);
Visual Basic	Declare Sub VCSavePlotterPenMap Lib "VCDLG32.DLL" (iError As Integer, ByVal szName As String)
Delphi	procedure VCSavePlotterPenMap(var iError: Integer; szName: PChar); far;
Parameters	*szName* - the name for the pen map.
Notes	Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device. The pen map controls the color, speed and width setting for each pen used by the plotter.
See Also	VCAddPlotterPenMapName, VCGetPlotterPenMapCount, VCGetPlotterPenMapName, VCGetPlotterPenMapping

VCSaveSettings

Version	1.2
Description	Saves all of the current entity and environment settings to a temporary memory buffer.
Declaration	
C/C++:	extern "C" void WINAPI VCSaveSettings();
Visual Basic:	Declare Sub VCSaveSettings Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCSaveSettings; far;
Parameters	No parameters are used with this subroutine.
Notes	Used to temporarily save the current settings so VCMatchCurrentEntity can extract all information form the current entity and its settings extracted from the system settings. The users default settings can then be restored with VCRestoreSettings.
See Also	VCRestoreSettings, VCSaveSysSettings, VCRestoreSysSettings, VCSaveEnvironment

VCSaveSysSettings

Version	1.2.1
Description	Saves all of the current environment settings (except for entity specific values) to a temporary memory buffer.
Declaration	
C/C++:	extern "C" void WINAPI VCSaveSysSettings(void);
Visual Basic:	Declare Sub VCSaveSysSettings Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCSaveSysSettings; far;
Parameters	No parameters are used with this subroutine.
Notes	Used to temporarily save the current settings so VCMatchCurrentEntity can extract all information form the current entity and its settings extracted from the system settings. The users default settings can then be restored with VCRestoreSettings.
See Also	VCSaveSettings, VCRestoreSettings, VCRestoreSysSettings

VCSaveStyle

Version 1.2

Description Saves part of the current drawing environment to a style file. This style file can subsequently be loaded to recreate the drawing environment.

Declaration

C/C++: extern "C" void WINAPI VCSaveStyle(char* fname);

Visual Basic: Declare Sub VCSaveStyle Lib "VCMAIN32.DLL" (ByVal fname As String)

Delphi: procedure VCSaveStyle(fname: PChar); far;

Parameters *fname* - a string representing the path and name for the style.

Notes Use the Save Style command to save related groups of current settings to disk. The settings can then be restored with the VCLoadStyle command. Styles are predefined collections of Corel Visual CADD settings, similar in concept to style sheets or templates used in most word processors (for more information about styles, see Load Styles). A style file can include anything from a single set of entity properties (layer, color, line type and line width) to virtually the entire Corel Visual CADD drawing environment. Style files allow users to quickly configure all relevant settings necessary for a particular task. By sharing style files users can easily create and follow office drafting standards.

See Also VCOpenStyle, VCSaveEnvironment

VCSaveVCA

Version	1.2
Description	Saves an attribute definition to disk.
Declaration	
C/C++:	extern "C" void WINAPI VCSaveVCA(char* szAttribName, char* szAttribFile);
Visual Basic:	Declare Sub VCSaveVCA Lib "VCMAIN32.DLL" (ByVal szAttribName As String, ByVal szAttribFile As String)
Delphi:	procedure VCSaveVCA(szAttribName: PChar; szAttribFile: PChar); far;
Parameters	*szAttribName* - a string representing the name of the attribute within the drawing. *szAttribFile* - a string representing the path and name for the file to save.
Notes	Attributes created in one drawing can not be used in other drawings until they have been saved to a VCA file on disk. A attribute definition can have two unique naming conventions. An on disk name used when saved to file(limited to the characters defined by the operating system) and an internal name used to store the name in a Corel Visual CADD drawing session. Most attribute commands require the internal name not the on disk name. The internal name can be determined from the saved name with VCGetAttributeInternalName.
See Also	VCOpenVCA, VCOpenVCS VCSaveVCS

VCSaveVCDToFile

Version	2.0
Description	Saves a VCD file.
Declaration	
C/C++	extern "C" void WINAPI VCSaveVCDToFile(short* iError, char* pS_);
Visual Basic	Declare Sub VCSaveVCDToFile Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS_ As String)
Delphi	procedure VCSaveVCDToFile(var iError: Integer; pS_: PChar); far;
Parameters	*pS* - the path and file name to siave the file.
Notes	There are several file formats Corel Visual CADD can save to. These include VCD, Generic CAD, AutoCAD DXF and AutoCAD DWG. VCSaveVCDFile is a direct routine to save the drawing into Corel Visual CADD native format. If your application is to provide support for the other file formats use VCSaveDrawing..
See Also	VCSaveDrawing, VCAcadWriteDWG, VCAcadWriteDXF

VCSaveVCDToStream

Version	2.0
Description	Saves a VCD to an OLE stream.
Declaration	
C/C++	extern "C" void WINAPI VCSaveVCDToStream(short* iError, void* pS_);
Visual Basic	Declare Sub VCSaveVCDToStream Lib "VCMAIN32.DLL" (iError As Integer, ByVal pS_ As String)
Delphi	procedure VCSaveVCDToStream(var iError: Integer; var pS_: Pointer); far;
Parameters	*pS* - name to save the stream.
Notes	All of the Corel Visual CADD OLE handler routines are available to create an OLE server. This routine will write the file to stream for the application to handle in its OLE event. Please see the documentation for creating an OLE application in your compiler help for details on a stream.
See Also	VCLoadVCDFromStream

VCSaveVCS

Version 1.2

Description Saves a symbol definition to disk for use in other drawing.

Declaration

C/C++: extern "C" void WINAPI VCSaveVCS(char* szSymbolName, char* szSymbolFile);

Visual Basic: Declare Sub VCSaveVCS Lib "VCMAIN32.DLL" (ByVal szSymbolName As String, ByVal szSymbolFile As String)

Delphi: procedure VCSaveVCS(szSymbolName: PChar; szSymbolFile: PChar); far;

Parameters *szSymbolName* - a string representing the name of the symbol within the drawing.
szSymbolFile - a string representing the path and name for the file to save.

Notes A symbol definition must exist prior to placement in a drawing. The symbol definition can be loaded into the drawing session from disk or created from existing entities. Corel Visual CADD utilizes several symbol formats in addition to the native VCS files. These include AutoCAD block (DWG, DXF) and Generic CADD components (CMP). These are loaded with VCOpenDWG and VCOpenCMP commands. An internal symbol definition can also be created within a drawing sessions using VCCreateSymbolDef and any of the VCAdd*Entity commands. The symbol can be added to the drawing database with VCAddSymbolEntity in which the programmer is responsible for handling placement and rubberbanding methods or with VCSymbolPlace in which Corel Visual CADD handles these internally. A symbol definition can have two unique naming conventions. An on disk name used when saved to file(limited to the characters defined by the operating system) and an internal name used to store the name in a Corel Visual CADD drawing session. VCAddSymbolEntity and VCPlaceSymbol both require the internal name not the on disk name. The internal name can be determined from the saved name with VCGetSymbolInternalName.

See Also VCOpenVCS, VCOpenVCA VCSaveVCA

VCSelectionRibalog

Version	1.2
Description	Displays the selection speedbar.
Declaration	
C/C++:	extern "C" void WINAPI VCSelectionRibalog(WORD id);
Visual Basic:	Declare Sub VCSelectionRibalog Lib "VCTOOL32.DLL" (ByVal id As Integer)
Delphi:	procedure VCSelectionRibalog(id: Integer); far;
Parameters	*id* - the command id of the tool initiating the selection speedbar.
Notes	Typically when a user uses an edit tool to modify existing entities, the entities to edit should be pre-selected following the Windows noun-verb paradigm. However in Corel Visual CADD, in order to assist those making the transition from DOS based packages, if nothing is selected when a edit tool is initiated, the selection bar will appear so the user can make a selection and then go on with tool as usual. When constructing external user tools, it is helpful to check the number of entities selected and then use VCSelectionRibalog to display the selection speedbar if the count is 0. The command id is used to place a text line on the speedbar while selecting entities, to serve as a reminder to the user of what tool they are using.
See Also	Appendix A

VCSendMessage

Version 1.2

Description The VCSendMessage function sends the specified message to Corel Visual CADD. The function calls the callback procedure for the window and does not return until that window procedure has processed the message. This is in contrast to the VCPostMessage function, which places (posts) the message in the window's message queue and returns immediately.

Declaration

C/C++: extern "C" LRESULT WINAPI VCSendMessage(WORD iMessage, WORD wParam, LPARAM lParam);

Visual Basic: Declare Function VCSendMessage Lib "VCMAIN32.DLL" (ByVal iMessage As Integer, ByVal wParam As Integer, ByVal lParam As Long) As Long

Delphi: function VCSendMessage(iMessage: Integer; wParam: Integer; lParam Longint):Longint; far;

Parameters *iMessage* - Specifies the message to be sent.
wParam - Specifies 16 bits of additional message-dependent information.
lParam - Specifies 32 bits of additional message-dependent information.
Returns - the result of the message processing and depends on the message sent.

Notes Windows frequently sends or posts messages to any running application depending on current focus, windows positioning, or current system activity. When programming an external application, these messages will be received from the system and need to be passed on to the Corel Visual CADD engine. In many cases there is a specific API call to do this, such as VCTimer, but there are many more that may need to be sent or relayed to Corel Visual CADD. This is the function used to do this. The wParam and lParam context will come from the message received from the system and typically be the message received. If the message is being posted to another application, and the wParam or lParam parameters are used to pass a handle or pointer to a global memory object, the memory should be allocated by the GlobalAlloc function, using the GMEM_SHARE flag.

See Also VCPostMessage

VCSet3DDisplay

Version	1.2
Description	Determines whether the 3D view is viewed as a wireframe or shaded view.
Declaration	
C/C++:	extern "C" void WINAPI VCSet3DDisplay(short* iError, short iCode);
Visual Basic:	Declare Sub VCSet3DDisplay Lib "VCMAIN32.DLL" (iError As Integer, ByVal iCode As Integer)
Delphi:	procedure VCSet3DDisplay(var iError: Integer; iCode: Integer); far;
Parameters	*iCode* - sets the 3D view type. 0 - VIEW3D_WIREFRAME 1 - VIEW3D_QSHADE
Notes	When creating 3D views of a drawing three parameters are required. They are view type, eye location, and viewed position. VCSetProjection3Ddetermines the view type and thus how the lines will be viewed in relation to each other, that is flat, parallel or perspective. VCSetView3D established the distance of the viewer from the viewed location as 3D coordinates and thus the level of perspective exaggeration used or the relative size of the view. VCChangeView3Dcan allow the users view point to be moved incrementally in certain directions and thus creates a limited "walk-through" functionality. 3D views can be viewed in wireframe or with Corel Visual CADD's built in quick shading. VCSet3DDisplay provides the ability to view the drawing as a quick shade and VCSet3DQShadeOptions determines the level of quick shade when the drawing is shaded.
See Also	VCSet3DQShadeOptions, VCChangeView3D, VCSetView3D, VCSetProjection3D

VCSet3DQShadeOptions

Version	1.2
Description	Determines the level of shading to be applied when a 3D view is shaded.
Declaration	
C/C++:	extern "C" void WINAPI VCSet3DQShadeOptions(short* iError, short iCode);
Visual Basic:	Declare Sub VCSet3DQShadeOptions Lib "VCMAIN32.DLL" (iError As Integer, ByVal iCode As Integer)
Delphi:	procedure VCSet3DQShadeOptions(var iError: Integer; iCode: Integer); far;
Parameters	*iCode* - sets the 3D view type. 0 - SHADE3D_ROUGH. 1 - SHADE3D_EXACT. 2 - SHADE3D_EXACT_SPLIT.
Notes	When creating 3D views of a drawing three parameters are required. They are view type, eye location, and viewed position. VCSetProjection3D determines the view type and thus how the lines will be viewed in relation to each other, that is flat, parallel or perspective. VCSetView3Destablished the distance of the viewer from the viewed location as 3D coordinates and thus the level of perspective exaggeration used or the relative size of the view. VCChangeView3Dcan allow the users view point to be moved incrementally in certain directions and thus creates a limited "walk-through" functionality. 3D views can be viewed in wireframe or with Corel Visual CADD's built in quick shading. VCSet3DDisplay provides the ability to view the drawing as a quick shade and VCSet3DQShadeOptions determines the level of quick shade when the drawing is shaded. When using quick shade keep in mind that as each level of accuracy is used the shading time may increase dramatically depending on the complexity of the drawing. Rough shading provides a quick but inexact shade and is best to get an idea of how the shade will look without taking a great deal of time. Exact will accurately shade all polygons but will not determine which polygons lie in front of others. For the best shading use exact and split which will calculate intersections of polygons and place forward surfaces in front of others.
See Also	VCSet3DDisplay, VCChangeView3D, VCSetView3D, VCSetProjection3D

VCSetAlertApp

Version 1.2

Description Registers an external hWnd with Corel Visual CADD, thus enabling messages to be sent back to the application when the specified events have occurred.

Declaration

C/C++: extern "C" void WINAPI VCSetAlertApp(short* iError, HWND hWnd, short iCode);

Visual Basic: Declare Sub VCSetAlertApp Lib "VCMAIN32.DLL" (iError As Integer, ByVal hWnd As Integer, ByVal iCode As Integer)

Delphi: procedure VCSetAlertApp(var iError: Integer; hWnd: Integer; iCode: Integer); far;

Parameters *hWnd* - the hWnd value of the object which will receive the messages from Corel Visual CADD.
iCode - a code representing the messages to be sent to the external application.
0 - ALERT_APP_ALL
1 - ALERT_APP_UTOOL_MOUSEDOWN
2 - ALERT_APP_UTOOL_MOUSEMOVE
4 - ALERT_APP_UTOOL_ABORT
8 - ALERT_APP_CMDLINE_CHAR
16 - ALERT_APP_CLOSE
32 - ALERT_APP_UTOOL_PENUP
64 - ALERT_APP_WORLD_CLOSE
128 - ALERT_APP_UTOOL_ERASERUBBER
256 - ALERT_APP_TOOL_COMPLETE
512 - ALERT_APP_UTOOL_INIT
1024 - ALERT_APP_UTOOL_TERMINATE
2048 - ALERT_APP_FRAME_CLOSE
4096 - ALERT_APP_FRAME_RESIZE
8192 - ALERT_APP_ENTITY_ERASED
16384 - ALERT_APP_ENTITY_SELECT_CHANGE
32768 - ALERT_APP_ACTIVATE
65536 - ALERT_APP_DEACTIVATE

Notes To initialize Windows messaging between Corel Visual CADD and an external application, the hWnd of some control or object must be sent to Visual CADD using VCSetAlertApp. When registering the hWnd a code must also be included which specifies which messages an application will receive. These can be added together to get multiple messages. For example iCode of 12 would specify that the command line characters and abort messages would be sent. To handle these messages, an application code must have code specifically to handle a Windows message sent to the control whose hWnd is registered with VCSetAlertApp. In Visual BASIC, handle this by supplying code in the mousedown event for the control specified for each mouse down message sent by Corel Visual CADD. Visual CADD is fairly intelligent about when to send this message and only send the message when a drawing point has been selected. This means that the user can issue snaps or use tracking without invoking the application code for the mousedown event. To retrieve the point the user selected in the drawing area, use VCGetUserToolLBDown which sets a Point2D of the last point picked. When trapping the user input, register the control with an iCode of either 0 (all messages) or 8 and add code to the control for keypress. When the keypress code is activated by the message from Corel Visual CADD, use VCGetCmdStr to retrieve the last keypress from Visual CADD. Once the keypress has been determined through code can act according to process the information or send it back for Corel Visual CADD to use with VCSetCmdStr. Once the application has completed with the messaging, use VCClearAlertApp to remove an application from the messaging registry.

See Also VCClearAlertApp, VCSetAlertAppDll, VCClearAlertAppDll

VCSetAlertAppDll

Version 2.0

Description Registers an external DLL with Corel Visual CADD, enabling messages to be sent back to the application when the specified events have occurred.

Declaration

C/C++ extern "C" void WINAPI VCSetAlertAppDll(short* iError, char* DllName, char* NativeCmd, long iCode);

Visual Basic Declare Sub VCSetAlertAppDll Lib "VCMAIN32.DLL" (iError As Integer, ByVal DllName As String, ByVal NativeCmd As String, ByVal iCode As Long)

Delphi procedure VCSetAlertAppDll(var iError: Integer; DllName: PChar; NativeCmd:

Parameters *DllName* - the name of the DLL containing the user tool.
NativeCmd - the native command name given to the tool.
iCode - a code representing the messages to be sent to the external DLL.
0 - ALERT_APP_ALL
1 - ALERT_APP_UTOOL_MOUSEDOWN
2 - ALERT_APP_UTOOL_MOUSEMOVE
4 - ALERT_APP_UTOOL_ABORT
8 - ALERT_APP_CMDLINE_CHAR
16 - ALERT_APP_CLOSE
32 - ALERT_APP_UTOOL_PENUP
64 - ALERT_APP_WORLD_CLOSE
128 - ALERT_APP_UTOOL_ERASERUBBER
256 - ALERT_APP_TOOL_COMPLETE
512 - ALERT_APP_UTOOL_INIT
1024 - ALERT_APP_UTOOL_TERMINATE
2048 - ALERT_APP_FRAME_CLOSE
4096 - ALERT_APP_FRAME_RESIZE
8192 - ALERT_APP_ENTITY_ERASED
16384 - ALERT_APP_ENTITY_SELECT_CHANGE
32768 - ALERT_APP_ACTIVATE
65536 - ALERT_APP_DEACTIVATE

Notes A new option available to Corel Visual CADD is to make tools and interfaces in dynamic link libraries (DLL's). This interface to Corel Visual CADD provides all the functionality of the message based EXE's tools that were used with version 1.x. Some advantages to DLL's over EXE's are: a DLL shares the same memory space as Corel Visual CADD, once loaded into memory, a DLL will stay in memory until Corel Visual CADD closes, code can be run on load and different code can be run each time a function is called, no interface or hWnd's are required, no checking is required to see if Corel Visual CADD is running since it is the one calling the DLL, and several tools can be in one DLL without command line options necessary for EXE's to achieve the same functionality.

Any tool is made up of several functions that handle each of the events passed by Corel Visual CADD. The old way was to use VCSetAlertApp to register a list of messages your user tool needed in order to function properly. This was limiting in many development languages like Visual BASIC because only certain controls could receive the needed messages and even those controls were limited by the number of messages they could handle. Even if all the needed messages were available they could accidentally be triggered if the interface was displayed on screen. Now, VCSetAlertAppDLL registers a group of exported functions in a DLL to be used instead relying on message handlers.

See Also VCSetAlertApp, VCClearAlertApp, VCClearAlertAppDll

VCSetAllDimPartsColor

Version 1.2

Description Sets all components of dimensions to the specified color.

Declaration

C/C++: extern "C" void WINAPI VCSetAllDimPartsColor(short* iError, short iColor);

Visual Basic: Declare Sub VCSetAllDimPartsColor Lib "VCMAIN32.DLL" (iError As Integer, ByVal iColor As Integer)

Delphi: procedure VCSetAllDimPartsColor(var iError: Integer; iColor: Integer); far;

Parameters *iColor* - sets the color index number to be used for all dimension parts.

Notes The elements that make up a dimension include the dimension line, left and right extension lines, left and right arrow and the dimension text. The API give developers complete control over the visual properties of each of the dimension elements independent of each other. Changing the properties of dimension elements will not effect previously drawn dimensions. Each component of a dimension can be individually set to a color with VCSetDimItemColor. VCSetAllDimPartsColor instead, sets the display color of all parts of the dimension settings. This forces all parts to be displayed in that color for all subsequent dimension placements.

See Also VCSetAllDimPartsOn, VCGetDimItemLineWidth, VCGetDimItemColor, VCGetDimItemLineType, VCGetDimItemShow

VCSetAllDimPartsOn

Version 1.2

Description Toggles the display of all dimension components to on.

Declaration

C/C++: extern "C" void WINAPI VCSetAllDimPartsOn(short* iError);

Visual Basic: Declare Sub VCSetAllDimPartsOn Lib "VCMAIN" (iError As Integer)

Delphi: procedure VCSetAllDimPartsOn(var iError: Integer); far;

Parameters *tf* - determines the display state for the dimension parts.
0 - turn all dimension part display off.
1- turn all dimension part display on.

Notes The elements that make up a dimension include the dimension line, left and right extension lines, left and right arrow and the dimension text. The API give developers complete control over the visual properties of each of the dimension elements independent of each other. Changing the properties of dimension elements will not effect previously drawn dimensions. Each component of a dimension can be individually set to display or not either by the user or by code with VCSetDimItemShow. VCSetAllDimPartsOn instead, toggles the display of all parts of dimension settings to on. This forces all parts to be displayed for all subsequent dimension placements.

See Also VCSetAllDimPartsColor, VCGetDimItemLineWidth, VCGetDimItemColor, VCGetDimItemLineType, VCGetDimItemShow

VCSetAngleHandle

Version	1.2
Description	Specifies to Corel Visual CADD which object is to display the angle reading from Corel Visual CADD.
Declaration	
C/C++:	extern "C" void WINAPI VCSetAngleHandle(HWND hWnd_);
Visual Basic:	Declare Sub VCSetAngleHandle Lib "VCMAIN32.DLL" (ByVal hWnd_ As Integer)
Delphi:	procedure VCSetAngleHandle(hWnd_: Integer); far;
Parameters	*hWnd* - the hWnd handle for the object to be used as the message area.
Notes	Like VCSetMessageHandle, VCSetAngleHandle sets a message handle of a Windows object to display a text message. However in this case the message is the 2pt angle formed by line segment between the last point placed and the current cursor location and the horizontal. This will reflect the current angular format settings. This is normally displayed in the status bar at the bottom of the Corel Visual CADD screen.
See Also	VCSetMessageHandle, VCSetDistanceHandle, VCSetXYHandle

VCSetAtbDefLabelValue

Version 1.2

Description Sets the value for a field label in the specified attribute.

Declaration

C/C++: extern "C" void WINAPI VCSetAtbDefLabelValue(short* iError, char* szName, char* szLabel, char* Value, short iRec);

Visual Basic: Declare Sub VCSetAtbDefLabelValue Lib "VCMAIN32.DLL" (iError As Integer, ByVal szName As String, ByVal szLabel As String, ByVal Value As String, ByVal iRec As Integer)

Delphi: procedure VCSetAtbDefLabelValue(var iError: Integer; szName: PChar; szLabel PChar; Value: PChar; iRec: Integer); far;

Parameters

szName - the internal name for the attribute defintion
szLabel - the label name to set the value
Value - the value to attach to at the label
iRec - the record number for the attribute

Notes Attributes are non-graphical data that can be attached to a symbol. The attribute are made up of fields represented by a label and a value. The label is a name for the attribute field and is designated when creating the attribute. The value is the value of the attribute field and can be edited after creating the attribute.

See Also VCGetAtbDefLabel, VCGetAtbDefRecordCount, VCGetAtbDefValue, VCGetAtbFont, VCGetAtbInternalName, VCGetCurEntAtbCount, VCGetCurEntAtbRecCount, VCGetCurEntAtbRecValue

VCSetCurrentDeSelected

Version	1.2
Description	Deselects the current entity.
Declaration	
C/C++:	extern "C" void WINAPI VCSetCurrentDeSelected(short* iError);
Visual Basic:	Declare Sub VCSetCurrentDeSelected Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSetCurrentDeSelected(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	In order for the Corel Visual CADD user to make edits or changes to any entities within the current drawing it is necessary for the entities to first be selected. If an application is to parse through the drawing for specific entities and allow the user to edit or change these entities, they must be selected. VCSetCurrentSelected does this without affecting the current selection set. This allows the application to step through the database and select any entities that match the selection criteria. VCSetCurrentDeSelected allows control over which entities are included in the selection set.
See Also	VCObjectSelect , VCClearSelection, VCCrossingSelect, VCDeSelectAll, VCSelectAll, VCSelectInvert, VCSelectLast, VCWindowSelect, VCSetCurrentSelected

VCSetCurrentEntity

Version	1.2
Description	Marks the specified entity as the current entity in order to retrieve information about that entity.
Declaration	
C/C++:	extern "C" void WINAPI VCSetCurrentEntity(short* iError, ENTITYHANDLE lH);
Visual Basic:	Declare Sub VCSetCurrentEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal lH As Long)
Delphi:	procedure VCSetCurrentEntity(var iError: Integer; lH: Longint); far;
Parameters	*lH* - the Corel Visual CADD entityhandle for the desired entity.
Notes	In order to modify or retrieve settings of drawing entities, it is first necessary to select a current entity. This can be done by either parsing through one at a time using VCFirstEntity and VCNextEntity or by directly selecting the current entity with VCSetCurrentEntity. This does not visually select the entity but simply set a pointer to the entity for later operations.
See Also	VCNextEntity, VCFirstEntity, VCFirstSelected, VCNextSelected,VCGetEntityCurrentHandle

VCSetCurrentEntityPoint

Version 1.2

Description Used to add the coordinates for a VCAddContinuousLineEntity or VCAddContinuousBezierEntity entity at the specified index point.

Declaration

C/C++: extern "C" void WINAPI VCSetCurrentEntityPoint(short* iError, short i, Point2D* dpP);

Visual Basic: Declare Sub VCSetCurrentEntityPoint Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer, dpP As Point2D)

Delphi: procedure VCSetCurrentEntityPoint(var iError: Integer; i: Integer; var dpP Point2D); far;

Parameters *i* - the index to the list of points to the line or curve, with 0 being the first point.
dpP - the Point2D structure containing the coordinates to place the entity.

Notes VCAddContinuousLineEntity and VCAddContinuousBezierEntity allow for a] number of points to be placed with the VCSetCurrentEntityPoint command instead of through a parameter.

See Also VCSetCurrentEntityPoints, VCAddContinuousBezierEntity, VCAddContinuousLineEntity, VCGetCurrentEntityPoint

VCSetCurrentEntityPoint3D

Version	1.2
Description	Used to add the coordinates for a VCAddPolygon3D entity at the specified index point.
Declaration	
C/C++:	extern "C" void WINAPI VCSetCurrentEntityPoint3D(short* iError, short i, Point3D* dpP);
Visual Basic:	Declare Sub VCSetCurrentEntityPoint3D Lib "VCMAIN32.DLL" (iError As Integer, ByVal i As Integer, dpP As Point3D)
Delphi:	procedure VCSetCurrentEntityPoint3D(var iError: Integer; i: Integer; var dpP Point3D); far;
Parameters	*i* - the index to the list of points to the line or curve, with 0 being the first point. *dpP* - the Point3D structure containing the coordinates to place the entity.
Notes	Any entity added to the Corel Visual CADD drawing database or to a symbol definition will take on the current properties for line type, color, layer, and width. These all need to be set before creating these entities or may be changed after creation with the change commands. All point locations including those within a symbol definition are relative to the drawing origin. Each entity added will be appended to the end of the database and take on the entity handle of 1 higher than the last entity in the drawing before the addition. Once a polygon3D is added to the drawing it contains no points and must have points added using VCSetCurrentEntityPoint3D.
See Also	VCAddPolygon3D,VCGetCurrentEntityPoint3D,VCGetCurrentEntityPoint

VCSetCurrentEntityPoints

Version	2.0.1
Description	Operates identical to VCSetCurrentEntityPoint except allows a complete array to be passed instead of individual points.
Declaration	
C/C++	extern "C" void WINAPI VCSetCurrentEntityPoints(short* iError, Point2D* p2dArray, short iPointCount);
Visual Basic	Declare Sub VCSetCurrentEntityPoints Lib "VCMAIN32.DLL" (iError As Integer, p2dArray As Point2D, ByVal iPointCount As Integer)
Delphi	procedure VCSetCurrentEntityPoints(var iError: Integer; var p2dArray: Point2D; iPointCount: Integer); far;
Parameters	*p2dArray* - the array of Point2D for placement points *iPointCount* - the number of points contained in the array.
Notes	By passing an array of points, an application can speed the generation of large continuous entities. For example, a COGO contour line which may require hundreds of points to define the contour. VCSetCurrentEntityPoint can be used to set individual points one at a time or VCSetCurrentEntityPoints can be used to pass the entire list of points directly into the routine. This will provide a significant performance increase for large continuous entities.
See Also	VCSetCurrentEntityPoints, VCAddContinuousBezierEntity, VCAddContinuousLineEntity, VCGetCurrentEntityPoint

VCSetCurrentEntitySubEntity

Version	2.0
Description	Sets the sub entity for parsing the boundary of a contour.
Declaration	
C/C++	extern "C" void WINAPI VCSetCurrentEntitySubEntity(short* iError, ENTITYHANDLE lH, short iContour, short iEntity);
Visual Basic	Declare Sub VCSetCurrentEntitySubEntity Lib "VCMAIN32.DLL" (iError As Integer, ByVal lH As Long, ByVal iContour As Integer, ByVal iEntity As Integer)
Delphi	procedure VCSetCurrentEntitySubEntity(var iError: Integer; lH: Longint; iContour: Integer; iEntity: Integer); far;
Parameters	*lH* - the entity handle for the hatch or fill. *IContour* - an 0 based index to the contour to parse. *IEntity* - the 0 based index to the subentity.
Notes	VCGetEntityContourCount provides a method to determine the number of contours that define the boundary for a hatch or fill. VCGetEntitySubEntityCount gives you the number of entities that are in each contour. For example, say you have an exploded rectangle with a hatch inside it. The VCGetEntityContourCount would return 1(the number of contours that define the hatch boundary), while VCGetEntitySubEntityCount will return a value of 4 (the number of entities that make up the contour boundary). After determining the number of contour and entities in the boundary an application can then parse the subentities be setting them based on an index, effectively walking through the hatch or fill boundary. Any property retrieval function such as VCGetCurrentEntityColorIndex can be used to get the desired information.
See Also	VCGetEntityContourCount, VCGetEntitySubEntityCount

VCSetCurrentErased

Version	1.2
Description	Sets the erased flag for the current entity to true.
Declaration	
C/C++:	extern "C" void WINAPI VCSetCurrentErased(short* iError);
Visual Basic:	Declare Sub VCSetCurrentErased Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSetCurrentErased(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	When modifying entities in the Corel Visual CADD database the program must create a new entity and erase the old one. This effectively appears to be a change and still allows undo and redo to work. VCSetCurrentErased is used to remove the previous entity.
See Also	VCSetCurrentSelected, VCSetCurrentDeselected, VCSetCurrentEntity, VCGetCurrentEntityHandle, VCSetCurrentUnErased

VCSetCurrentSelected

Version	1.2
Description	Selects the current entity and makes it available for modify commands.
Declaration	
C/C++:	extern "C" void WINAPI VCSetCurrentSelected(short* iError);
Visual Basic:	Declare Sub VCSetCurrentSelected Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSetCurrentSelected(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	In order for the Corel Visual CADD user to make edits or changes to any entities within the current drawing it is necessary for the entities to somehow be selected. If an application is to parse through the drawing for specific entities and allow the user to edit or change these entities, they must be selected. VCSetCurrentSelected does this without affecting the current selection set. This allows the application to step through the database and select any entities that match the selection criteria. VCSetCurrentDeSelected allows control over which entities are included in the selection set.
See Also	VCObjectSelect , VCClearSelection, VCCrossingSelect, VCDeSelectAll, VCSelectAll, VCSelectInvert, VCSelectLast, VCWindowSelect, VCSetCurrentDeSelected

VCSetCurrentUnErased

Version	1.2
Description	Sets the erased flag for the current entity to false.
Declaration	
C/C++:	extern "C" void WINAPI VCSetCurrentUnErased(short* iError);
Visual Basic:	Declare Sub VCSetCurrentUnErased Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSetCurrentUnErased(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	When modifying entities in the Corel Visual CADD database the program must create a new entity and erase the old one. This effectively appears to be a change and still allows undo and redo to work. VCSetCurrentErased is used to remove the previous entity.
See Also	VCSetCurrentSelected, VCSetCurrentDeSelected, VCSetCurrentEntity, VCGetCurrentEntityHandle, VCSetCurrentErased

VCSetDialogFrameHwnd

Version	2.0
Description	Sets the Windows handle for the frame to display Corel Visual CADD dialogs.
Declaration	
C/C++	extern "C" void WINAPI VCSetDialogFrameHwnd(HWND hWndFrame);
Visual Basic	Declare Sub VCSetDialogFrameHwnd Lib "VCDLG32.DLL" (ByVal hWndFrame As Integer)
Delphi	procedure VCSetDialogFrameHwnd(hWndFrame: Integer); far;
Parameters	*hWndFrame* - Windows handle for the application frame
Notes	Many features of the Corel Visual CADD interface can be utilized directly in a custom application created with a separate interface. The interface features include dialogs, toolbars, menus and child drawing windows. In order to access these features in the custom interface, a Windows hWnd needs to be provided to display each of the Corel Visual CADD features. Once the dialog frame is set, all the Visual CADD dialogs and ribalogs will function the same in a custom interface.
See Also	VCSetDialogToolFrameHwnd, VCSethMenu, VCSethWndMdiClient

VCSetDialogToolFrameHwnd

Version	2.0
Description	Sets the HWND for displaying the Corel Visual CADD toolbars.
Declaration	
C/C++	extern "C" void WINAPI VCSetDialogToolFrameHwnd(HWND hWndFrame);
Visual Basic	Declare Sub VCSetDialogToolFrameHwnd Lib "VCDLG32.DLL" (ByVal hWndFrame As Integer)
Delphi	procedure VCSetDialogToolFrameHwnd(hWndFrame: Integer); far;
Parameters	*hWndFrame* - the HWND for the window.
Notes	Many features of the Corel Visual CADD interface can be utilized directly in a custom application created with a separate interface. The interface features include dialogs, toolbars, menus and child drawing windows. In order to access these features in the custom interface, a Windows hWnd needs to be provided to display each of the Corel Visual CADD features. Once the dialog frame is set, all the Corel Visual CADD dialogs and ribalogs will function the same in a custom interface.
See Also	VCSetDialogFrameHwndVCSethMenu, VCSethWndMdiClient

VCSetEatNextLButtonDown

Version	2.0
Description	Causes the next mouse down in the interface to be ignored.
Declaration	
C/C++	extern "C" vbool WINAPI VCSetEatNextLButtonDown(short* iError, vbool tfSet);
Visual Basic	Declare Function VCSetEatNextLButtonDown Lib "VCMAIN32.DLL" (iError As Integer, ByVal tfSet As Integer) As Integer
Delphi	function VCSetEatNextLButtonDown(var iError: Integer; tfSet: Boolean):Boolean;
Parameters	*tfSet* - flag whether to ignore the next left button down. 0 - do not ignore the next left button down. 1 - ignore the next left button down.
Notes	Usually only a utility function, this routine is used to ignore a mouse down button in the Corel Visual CADD interface.
See Also	VCGetUserToolLBDown, VCGetUserToolLBUp, VCGetUserToolMouseMove

VCSetEntitySection

Version	1.2
Description	Specifies that when parsing the drawing database, to only step through the drawing area instead of the symbol area.
Declaration	
C/C++:	extern "C" void WINAPI VCSetEntitySection(short* iError);
Visual Basic:	Declare Sub VCSetEntitySection Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSetEntitySection(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	When setting and getting current entity properties, the current entity is always set using VCFirstEntity, VCNextEntity, or VCSetCurrentEntity. By default these only traverse the drawing entities and not the symbol entities. VCSetEntitySection tells Corel Visual CADD that each current entity selection to only select drawing entities and not entities in symbols. VCSetSymbolSection conversely only steps through entities in symbol definitions and not drawing entities.
See Also	VCSetSymbolSection, VCSetCurrentEntity, VCNextEntity, VCFirstEntity

VCSetFilterActive

Version 1.2

Description When the filter is on, selection operations capture only those objects meeting all of the filter criteria.

Declaration

C/C++: extern "C" void WINAPI VCSetFilterActive(short* iError, vbool tf);

Visual Basic: Declare Sub VCSetFilterActive Lib "VCMAIN32.DLL" (iError As Integer, ByVal tf As Integer)

Delphi: procedure VCSetFilterActive(var iError: Integer; tf: Boolean); far;

Parameters *tf* - determines the state of the selection filter
0 - the filter is inactive.
1 - the filter is active.

Notes The API allows an application to filter entities prior to making selections. By setting a selection criteria based on entity properties and settings, the selection routine will only "capture" those objects meeting the filter criteria. The filter criteria can be set based on entity kind, layer, color, line type and line width. Can be used to create a fast parsing method for specific entity types when combined with the specialized parsing calls VCFirstSelected and VCNextSelected.

See Also VCGetFilterKind, VCGetFilterKind2, VCGetFilterLayer, VCGetFilterLineType, VCGetFilterName, VCGetFilterWidth, VCGetFilterColor, VCFirstSelected, VCNextSelected

VCSetFilterMatch

Version	1.2
Description	Sets the selection filter properties based on the input entity.
Declaration	
C/C++:	extern "C" void WINAPI VCSetFilterMatch(short* iError, ENTITYHANDLE EH);
Visual Basic:	Declare Sub VCSetFilterMatch Lib "VCMAIN32.DLL" (iError As Integer, ByVal EH As Long)
Delphi:	procedure VCSetFilterMatch(var iError: Integer; EH: Longint); far;
Parameters	*iError* is set depending on the success or failure of the function. 0 - The function succeeded. 1 - The function failed due to an invalid drawing world. *EH* - entity handle for the entity to match for the filter properties
Notes	The API allows an application to filter entities prior to making selections. By setting a selection criteria based on entity properties and settings, the selection routine will only "capture" those objects meeting the filter criteria. The filter criteria can be set based on entity kind, layer, color, line type and line width. VCSetFilterMatch allows a filter properties to be set based on a specific entity without matching and then setting the property values. Can be used to create a fast parsing method for specific entity types when combined with the specialized parsing calls VCFirstSelected and VCNextSelected.
See Also	VCGetFilterKind, VCGetFilterKind2, VCGetFilterLayer, VCGetFilterLineType, VCGetFilterName, VCGetFilterWidth, VCGetFilterColor, VCFirstSelected, VCNextSelected

VCSethWndMdiClient

Version	2.0
Description	Sets the HWND for displaying the Corel Visual CADD MDI Windows.
Declaration	
C/C++	extern "C" void WINAPI VCSethWndMdiClient(long hWnd);
Visual Basic	Declare Sub VCSethWndMdiClient Lib "VCMAIN32.DLL" (ByVal hWnd As Long)
Delphi	procedure VCSethWndMdiClient(hWnd: Longint); far;
Parameters	*hWnd* - the HWND for the window.
Notes	Many features of the Corel Visual CADD interface can be utilized directly in a custom application created with a separate interface. The interface features include dialogs, toolbars, menus and child drawing windows. In order to access these features in the custom interface, a Windows hWnd needs to be provided to display each of the Corel Visual CADD features. Once the dialog frame is set, all the Corel Visual CADD dialogs and ribalogs will function the same in a custom interface.
See Also	VCSetDialogFrameHwndVCSetDialogToolFrameHwndVCSethMenu, VCSethWndMdiClient

VCSetGraphicPenWidth

Version	2.0
Description	Sets the pen width used for displaying fill patterns.
Declaration	
C/C++	extern "C" void WINAPI VCSetGraphicPenWidth(short* iError, double dPen);
Visual Basic	Declare Sub VCSetGraphicPenWidth Lib "VCMAIN32.DLL" (iError As Integer, ByVal dPen As Double)
Delphi	procedure VCSetGraphicPenWidth(var iError: Integer; dPen: Double); far;
Parameters	*dPen* - the real world width for the pen.
Notes	Some entities defined by several graphical objects, hatch patterns, fills, line types and fonts. For instance, a hatch pattern is defined by lines to make a useful pattern. These entities are not available for access through the standard database parsing routines provided. This is due to the fact that typically an application will not need this specific information. Most applications will need to simply parse the database and retrieve the entity information provided. In situations where a custom vector output file is being defined or to guide a CNC milling machine, the application may need to define all the vectors making up even the complex entities. The graphic handle method allow for this detailed parsing functionality. In order to access the information an application should first create a graphics handle using VCCreateGraphicsHandle. This function creates a parsing list from the current entity if it is a graphic entity, hatch, fill, text or line type. The iError return will be > 0 if the current entity is not a graphic entity. The application can then parse the new set with VCFirstGraphic and VCNextGraphic. Any required information can be retrieved using any standard query function such as VCGetCurrentEntityPoint. The entity is considered read-only and only retrieval API routines may be utilized. The individual graphic entities can not be set with any command. After completing the parse the application should call VCDeleteGraphicHandle to destroy the created handle.
See Also	VCCreateGraphicsHandle, VCFirstGraphic, VCNextGraphic VCIsGraphic

VCSetHeaderUserData

Version 1.2

Description Sets the user data section to the drawing header for attaching user data.

Declaration

C/C++: extern "C" void WINAPI VCSetHeaderUserData(short* iError);

Visual Basic: Declare Sub VCSetHeaderUserData Lib "VCMAIN32.DLL" (iError As Integer)

Delphi: procedure VCSetHeaderUserData(var iError: Integer); far;

Parameters No additional parameters are used with this subroutine..

Notes User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data may be of the C variable types double, float, long, or short. In addition to these types, a user defined type of "chunk" may also be stored. A chunk may be any size and is simply a pointer to a memory location. The size of the chunk is also passed so Corel Visual CADD can retrieve the appropriate amount of data from the specified memory location. Whenever using user data, an application must set a user data name in order to protect private data and to ensure that different applications do not interfere with the others data. VCSetUserDataName is provided for this purpose, while VCGetUserDataName checks the currently set user data name. The name must only be set one time before adding any user data. By registering as a Corel Visual CADD 3rd Party developer, Corel will provide a user data "name" which should be used for this purpose. The VCAddCurrentEntityUserData* calls always append the new variable as the last user data variable. The VCSetCurrentEntityUserData* calls add the user data variable at the index specified in the call, provided that there are indeed that many indices already attached, and overwrite any existing user data at that index. User data is always attached to the current entity which is set using VCFirstEntity, VCNextEntity, VCFirstSelected, VCNextSelected or VCSetCurrentEntity. As previously mentioned, user data may be attached to the drawing header. This is achieved by using VCSetHeaderUserData and then attaching the appropriate user data. Once VCNextEntity or any other current entity selections are used, the user data calls will again be used on the current entity.

See Also VCAddCurrentEntityUserDataChunk, VCAddCurrentEntityUserDataDouble, VCAddCurrentEntityUserDataFloat, VCAddCurrentEntityUserDataLong, VCAddCurrentEntityUserDataShort

VCSetLastCommandId

Version 1.2

Description Sets a command id as the last command issued in Corel Visual CADD. Useful for establishing what the spacebar will repeat or execute.

Declaration

C/C++: extern "C" void WINAPI VCSetLastCommandId(WORD CmdId);

Visual Basic: Declare Sub VCSetLastCommandId Lib "VCMAIN32.DLL" (ByVal CmdId As Integer)

Delphi: procedure VCSetLastCommandId(CmdId: Integer); far;

Parameters *id* - the command id of the command to be set as the last command.

Notes In Corel Visual CADD, after a command has been completed, the spacebar can be used to repeat the last command. Using this subroutine, an application can establish any command as the last command and effectively assign what the spacebar will execute.

See Also Appendix A

VCSetMaxUID

Version	2.0
Description	Sets the maximum UID that may be used.
Declaration	
C/C++	extern "C" void WINAPI VCSetMaxUID(short* iError, UID uidMax);
Visual Basic	Declare Sub VCSetMaxUID Lib "VCMAIN32.DLL" (iError As Integer, ByVal uidMax As Long)
Delphi	procedure VCSetMaxUID(var iError: Integer; uidMax: Longint); far;
Parameters	*uidMax* - the maximum UID that can be used in the drawing.
Notes	Each entity in Corel Visual CADD maintains a unique entity identifier in order to track the entity. This is in addition to the dynamic entity handle which changes as entities are deleted and modified in the database. As entities are added to the drawing both an entity handle and a UID are assigned to the entity. The entity handle will change as items are deleted and modified on the database while the UID will remain constant. Whenever linking entities to external databases or arrays, the application should utilize the UID due to its unchanging value with each entity. The entity handle is used when parsing the database or setting specific entities within the drawing session. The UID can should be audited prior to any external storage in order to ensure uniqueness in the ID.
See Also	VCAuditUIDS

VCSetMBMode

Version	1.2
Description	Sets an operating mode where Corel Visual CADD interprets manually-entered coordinates relative to a user-defined point.
Declaration	
C/C++:	extern "C" void WINAPI VCSetMBMode(short* iError);
Visual Basic:	Declare Sub VCSetMBMode Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSetMBMode(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	Corel Visual CADD allows three modes of coordinate input. These are referred to as manual entry modes and include the following: manual entry origin, manual entry relative, and manual entry basepoint. Use the Manual Entry Basepoint command to set the operating mode to the basepoint manual entry mode. In the basepoint mode, specify a temporary origin that remains in effect until a user changes its location or change modes. This mode is particularly useful when locations are known in relation to one specific point. The entry mode is a three way toggle and is set using VCSetMOMode, VCSetMRMode, and VCSetMBMode which toggle to origin, relative and base point respectively. All coordinate data input through the API is always relative to the origin regardless of the manual entry setting.
See Also	VCSetMRMode, VCSetMOMode

VCSetMOMode

Version	1.2
Description	Sets an operating mode where Corel Visual CADD interprets manually-entered coordinates relative to the drawing origin.
Declaration	
C/C++:	extern "C" void WINAPI VCSetMOMode(short* iError);
Visual Basic:	Declare Sub VCSetMOMode Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSetMOMode(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	Corel Visual CADD allows three modes of coordinate input. These are referred to as manual entry modes and include the following: manual entry origin, manual entry relative, and manual entry basepoint. Use the Manual Entry Absolute command to set the operating mode to the absolute manual entry mode. In the absolute mode, coordinates are interpreted as relative to the drawing origin. This mode is particularly useful when locations are calculated or imported through external programs or macros. The entry mode is a three way toggle and is set using VCSetMOMode, VCSetMRMode, and VCSetMBMode which toggle to origin, relative and base point respectively. All coordinate data input through the API is always relative to the origin regardless of the manual entry setting.
See Also	VCSetMBMode, VCSetMRMode

VCSetMRMode

Version	1.2
Description	Sets an operating mode where Corel Visual CADD interprets manually-entered coordinates relative to the last point referenced.
Declaration	
C/C++:	extern "C" void WINAPI VCSetMRMode(short* iError);
Visual Basic:	Declare Sub VCSetMRMode Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSetMRMode(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	Corel Visual CADD allows three modes of coordinate input. These are referred to as manual entry modes and include the following: manual entry origin, manual entry relative, and manual entry basepoint. Use the Manual Entry Relative command to set the operating mode to the relative manual entry mode. In the relative mode, each point placed or referenced through a snap or other command becomes a temporary origin for the next operation. This mode is particularly useful when distances are measured in sequence, with the end of one measurement being the beginning of the next. The entry mode is a three way toggle and is set using VCSetMOMode, VCSetMRMode, and VCSetMBMode which toggle to origin, relative and base point respectively. All coordinate data input through the API is always relative to the origin regardless of the manual entry setting.
See Also	VCSetMBMode, VCSetMOMode

VCSetNamedLayer

Version	2.0
Description	Names a layer at the given index.
Declaration	
C/C++	extern "C" void WINAPI VCSetNamedLayer(short* iError, short iIndex, char* pName);
Visual Basic	Declare Sub VCSetNamedLayer Lib "VCMAIN32.DLL" (iError As Integer, ByVal iIndex As Integer, ByVal pName As String)
Delphi	procedure VCSetNamedLayer(var iError: Integer; iIndex: Integer; pName: PChar); far;
Parameters	*iIndex* - the layer index to name from 0 to 1023. *pName* - the name to apply.
Notes	The API provides two methods for naming layers in the active drawing. The first utilizes VCAddNamedLayer and simply names the first layer in the list that has not already been named. The function begins a parse on a 0 based layer index until the first non-named layer. It then names the layer the given value and returns the index for the layer. This routine is generally used when building a setup routine where the entire layer naming scheme is known up front. The second method allows the application to apply a name to a specific layer. VCSetNamedLayer takes a layer index as a parameter for naming. This operates more in hand with the Corel Visual CADD interface since a user or application can pick the layer to name prior to the operation.
See Also	VCAddNamedLayer

VCSetPlotSettings

Version	2.0
Description	Specifies the settings for use in the direct plot routine.
Declaration	
C/C++	extern "C" void WINAPI VCSetPlotSettings(short* iError, PlotStruct* pSettings);
Visual Basic	Declare Sub VCSetPlotSettings Lib "VCDLG32.DLL" (iError As Integer, pSettings As PlotStruct)
Delphi	procedure VCSetPlotSettings(var iError: Integer; var pSettings: PlotStruct);
Parameters	*pSettings* - the PlotStruct containg the settings for the plot routine
Notes	Corel Visual CADD contains both a Print and Plot command. The print command utilizes the standard Windows drivers for output to the device. The plot command is an internal routine allowing more control over vector output devices by bypassing the Windows drivers. Each of these commands maintain separate default settings for the print output such as scale, orientation and page size. These settings are maintained in a structure defined for Corel Visual CADD.
See Also	VCGetPrintSettings

VCSetPlotterCurrentLanguageIndex

Version 2.0

Description Specifies the current plotter language.

Declaration

C/C++ extern "C" void WINAPI VCSetPlotterCurrentLanguageName(short* iError, char* szLanguageName);

Visual Basic Declare Sub VCSetPlotterCurrentLanguageIndex Lib "VCDLG32.DLL" (iError As Integer, ByVal iIndex As Integer)

Delphi procedure VCSetPlotterCurrentLanguageIndex(var iError: Integer; iIndex:Integer); far;

Parameters *iIndex* - the index for the current plotter language

Notes Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device.

The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.

Corel Visual CADD ships with support for many common plotter languages. However, if the desired language is not available, an application can create a language directly through the API. A plotter language consists of a delimiter, initialization string, de-initialization string, pen up, pen move, pen draw, pen speed and pen change commands. Each of these needs to be specified when creating a language. The required control codes are generally listed in the output devices documentation and set to a specific plotter type

See Also VCSetPlotterCurrentPenMapIndex, VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName

VCSetPlotterCurrentPenMapIndex

Version	2.0
Description	Specifies the current pen map used by the direct plot routine.
Declaration	
C/C++	extern "C" void WINAPI VCSetPlotterCurrentPenMapIndex(short* iError, short iIndex);
Visual Basic	Declare Sub VCSetPlotterCurrentPenMapIndex Lib "VCDLG32.DLL" (iError As Integer, ByVal iIndex As Integer)
Delphi	procedure VCSetPlotterCurrentPenMapIndex(var iError: Integer; iIndex:
Parameters	*iIndex* - the index specifying the current pen map
Notes	Corel Visual CADD ships with a direct plot routine in order to enhance the control over vector output devices. By using the direct plot method, an application can bypass the Windows drivers and send information directly to the plotter. This leads to enhanced control of the pen mappings for the device. The direct plot routine utilizes a driver, language and pen map to control the output. The driver determines the device settings such as communication port, Baud Rate, Parity and Data Bits. The language controls the character codes used by the plotter to control the pen movements. These are defined by Pen Up, Pen Down and Pen Move and other commands. The pen map controls the color, speed and width setting for each pen used by the plotter.
See Also	VCGetPlotterCurrentLanguageName, VCGetPlotterCurrentPageSize, VCGetPlotterCurrentPenMapName

VCSetProjection3D

Version	1.2
Description	Determines the type of projection to be used when using 3D views.
Declaration	
C/C++:	extern "C" void WINAPI VCSetProjection3D(short* iError, short iCode);
Visual Basic:	Declare Sub VCSetProjection3D Lib "VCMAIN32.DLL" (iError As Integer, ByVal iCode As Integer)
Delphi:	procedure VCSetProjection3D(var iError: Integer; iCode: Integer); far;
Parameters	*iCode* - sets the 3D view type. 0 - VIEW3D_FLAT 1 - VIEW3D_PARALLEL 2 - VIEW3D_PERSPECTIVE
Notes	When creating 3D views of a drawing three parameters are required. They are view type, eye location, and viewed position. VCSetProjection3D determines the view type and thus how the lines will be viewed in relation to each other, that is flat, parallel or perspective. VCSetView3D established the distance of the viewer from the viewed location as 3D coordinates and thus the level of perspective exaggeration used or the relative size of the view. VCChangeView3Dcan allow the users view point to be moved incrementally in certain directions and thus creates a limited "walk-through" functionality. 3D views can be viewed in wireframe or with Corel Visual CADD's built in quick shading. VCSet3DDisplay provides the ability to view the drawing as a quick shade and VCSet3DQShadeOptionsdetermines the level of quick shade when the drawing is shaded.
See Also	VCSet3DQShadeOptions, VCSet3DDisplay, VCChangeView3D, VCSetView3D

VCSetSymbolSection

Version	1.2
Description	Specifies that when traversing the drawing database to only step through a specified symbol and not through the drawing area.
Declaration	
C/C++:	extern "C" void WINAPI VCSetSymbolSection(short* iError, char* pName);
Visual Basic:	Declare Sub VCSetSymbolSection Lib "VCMAIN32.DLL" (iError As Integer, ByVal pName As String)
Delphi:	procedure VCSetSymbolSection(var iError: Integer; pName: PChar); far;
Parameters	*pName* - the name of the symbol to step through.
Notes	When setting and getting current entity properties, the current entity is always set using VCFirstEntity, VCNextEntity, or VCSetCurrentEntity. By default these only traverse the drawing entities and not the symbol entities. VCSetEntitySection tells Corel Visual CADD that each current entity selection to only select drawing entities and not entities in symbols. VCSetSymbolSection conversely only steps through entities in symbol definitions and not drawing entities.
See Also	VCSetEntitySection

VCSetUserTool

Version 1.2

Description Creates a new user defined tool and its first prompt.

Declaration

C/C++: extern "C" void WINAPI VCSetUserTool(short iStates, char* szNativeCmd, char* szFirstPrompt);

Visual Basic: Declare Sub VCSetUserTool Lib "VCTOOL32.DLL" (ByVal iStates As Integer, ByVal szNativeCmd As String, ByVal szFirstPrompt As String)

Delphi: procedure VCSetUserTool(iStates: Integer; szNativeCmd: PChar; szFirstPrompt PChar); far;

Parameters *iStates* - the number of steps the tool uses (-1 specifies a continuous tool that ends with esc or a pen-up).
szNativeCmd - the name of a command as defined in cmdext.def.
szFirstPrompt - a string used as the first prompt for the tool.

Notes Developing a user tool requires the tool to be defined in the CMDEXT.DEF file found in the Corel Visual CADD system directory. The format for the file is as follows:.
INSDOOR,ID,c:\vcadd\insdoor.bmp,Insert Door,Insert Door,ExeName;c:\vcadd\vbapps\insdoor.exe;Run;where.
"INSDOOR" is the native command name.
"ID" is the two letter command.
"c:\vcadd\insdoor.bmp" is the bitmap to be used for the button face.
"Insert Door" is the default menu text as appears on any menu.
"Insert Door" is the description prompt as it appears on the command line.
"ExeName;c:\vcadd\vbapps\insdoor.exe;Run" is the script used to execute the *.EXE for the tool, and follows all single line script conventions.

See Also VCGetUserToolLBDown, VCGetUserToolMouseMove, VCSetAlertApp, VCClearAlertApp

VCSetView3D

Version	1.2
Description	Establishes the viewers eye position and the position of the point being viewed.
Declaration	
C/C++:	extern "C" void WINAPI VCSetView3D(short* iError, Point3D* dpEye, Point3D* dpTarget);
Visual Basic:	Declare Sub VCSetView3D Lib "VCMAIN32.DLL" (iError As Integer, dpEye As Point3D, dpTarget As Point3D)
Delphi:	procedure VCSetView3D(var iError: Integer; var dpEye: Point3D; var dpTarget Point3D); far;
Parameters	*dpEye* - the location in 3D space of the viewers position. *dpTarget* - the position in 3D space where the viewer is looking.
Notes	When creating 3D views of a drawing three parameters are required. They are view type, eye location, and viewed position. VCSetProjection3D determines the view type and thus how the lines will be viewed in relation to each other, that is flat, parallel or perspective. VCSetView3D established the distance of the viewer from the viewed location as 3D coordinates and thus the level of perspective exaggeration used or the relative size of the view. VCChangeView3D can allow the users view point to be moved incrementally in certain directions and thus creates a limited "walk-through" functionality. 3D views can be viewed in wireframe or with Corel Visual CADD's built in quick shading. VCSet3DDisplay provides the ability to view the drawing as a quick shade and VCSet3DQShadeOptions determines the level of quick shade when the drawing is shaded.
See Also	VCSet3DQShadeOptions, VCSet3DDisplay, VCChangeView3D, VCSetProjection3D

VCSetWorldZoomAll

Version	1.2
Description	Reduces or enlarges the image as necessary to fill the screen with the entire drawing.
Declaration	
C/C++:	extern "C" void WINAPI VCSetWorldZoomAll(short* iError);
Visual Basic:	Declare Sub VCSetWorldZoomAll Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSetWorldZoomAll(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	Functions the same as the tool command VCZoomAll.
See Also	VCZoomAll, VCSetWorldZoomWindow

VCSetWorldZoomWindow

Version	1.2
Description	Changes the zoom so that a windowed area fills the screen**.**
Declaration	
C/C++:	extern "C" void WINAPI VCSetWorldZoomWindow(short* iError, Point2D* p0, Point2D* p1);
Visual Basic:	Declare Sub VCSetWorldZoomWindow Lib "VCMAIN32.DLL" (iError As Integer, p0 As Point2D, p1 As Point2D)
Delphi:	procedure VCSetWorldZoomWindow(var iError: Integer; var p0: Point2D; var p1 Point2D); far;
Parameters	*p0* - the Point2D structure containg the coordinates for the lower left corner of the window *p1* - the Point2D structure containg the coordinates for the upper right corner of the window
Notes	Functions the same as the tool command except it allows for coordinate entry through the parameter list.
See Also	VCZoomWindow, VCSetWorldZoomAll

VCSortCurrentHatchFillEntity

Version	1.2
Description	Evaluates the boundary for hatch/fill definition.
Declaration	
C/C++:	extern "C" void WINAPI VCSortCurrentHatchFillEntity(short* iError);
Visual Basic:	Declare Sub VCSortCurrentHatchFillEntity Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCSortCurrentHatchFillEntity(var iError: Integer); far;
Parameters	No additional parameters are used with this subroutine.
Notes	VCAddFillEntity and VCAddHatchEntity allow hatch and fill boundaries to be specified by any other entity types available in Corel Visual CADD. A hatch or fill entity is created by adding a reference to the entity type, building the boundary from other entity types and the sorting the boundary to finish the hatch or fill entity. VCSortCurrentHatchFillEntity forces Corel Visual CADD to evaluate the input boundary entities for hatching or filling. The input entities must form a closed boundary.
See Also	VCAddFillEntity, VCAddHatchEntity, VCHatchSelected, VCFillSelected

VCStringToAngle

Version 1.2

Description Converts an input string into radians for use in other API routines.

Declaration

C/C++: extern "C" void WINAPI VCStringToAngle(short* iError, double* pD, char* pS);

Visual Basic: Declare Sub VCStringToAngle Lib "VCMAIN32.DLL" (iError As Integer, pD As Double, ByVal pS As String)

Delphi: procedure VCStringToAngle(var iError: Integer; var pD: Double; pS: PChar);

Parameters *pD* - the returned angle value in radians.
pS - the input string for evaluation.

Notes The Corel Visual CADD API offers several utility routine to assist in capturing user input. VCStringToAngle and VCStringToDist allow an application to utilize some of the built in command line structure available through the Corel Visual CADD command prompt. VCStringToAngle will detect and convert an input string value in decimal degrees or degrees:minute:second format into a radian value for use within the API. VCStringToDist interprets the user entry string and converts the coordinates based on the current units or input units in the string.

See Also VCStringToDist, VCGetUnitConversionFactor, VCAngleToString, VCDistToString

VCStringToDist

Version	1.2
Description	Converts an input string into world coordinates for use in other API routines.
Declaration	
C/C++:	extern "C" void WINAPI VCStringToDist(short* iError, double* pD, char* pS);
Visual Basic:	Declare Sub VCStringToDist Lib "VCMAIN32.DLL" (iError As Integer, pD As Double, ByVal pS As String)
Delphi:	procedure VCStringToDist(var iError: Integer; var pD: Double; pS: PChar);
Parameters	*pD* - the returned distance *pS* - the input string for evaluation
Notes	The Corel Visual CADD API offers several utility routine to assist in capturing user input. VCStringToAngle and VCStringToDist allow an application to utilize some of the built in command line structure available through the Corel Visual CADD command prompt. VCStringToAngle will detect and convert an input string value in decimal degrees or degrees:minute:second format into a radian value for use within the API. VCStringToDist interprets the user entry string and converts the coordinates based on the current units or input units in the string.
See Also	VCStringToAngle, VCAngleToString, VCDistToString, VCGetUnitConversionFactor

VCSymbolPlace

Version	1.2
Description	Allows a loaded symbol definition to be positioned in the drawing.
Declaration	
C/C++:	extern "C" void WINAPI VCSymbolPlace(char* szName);
Visual Basic:	Declare Sub VCSymbolPlace Lib "VCTOOL32.DLL" (ByVal szName As String)
Delphi:	procedure VCSymbolPlace(szName: PChar); far;
Parameters	*szName* - the internal name of the symbol to place as it appears in the symbol list.
Notes	A symbol definition must exist prior to placement in a drawing. The symbol definition can be loaded into the drawing session from disk or created from existing entities. Corel Visual CADD utilizes several symbol formats in addition to the native VCS files. These include AutoCAD block (DWG, DXF) and Generic CADD components (CMP). These are loaded with VCAcadBlockRead and VCOpenCMP commands. An internal symbol definition can also be created within a drawing sessions using VCCreateSymbolDef and any of the VCAdd* commands. The symbol can be added to the drawing database with VCAddSymbolEntity in which the programmer is responsible for handling placement and rubberbanding methods or with VCSymbolPlace in which Corel Visual CADD handles these internally. A symbol definition can have two unique naming conventions. An on disk name used when saved to file(limited to the characters defined by the operating system) and an internal name used to store the name in a Corel Visual CADD drawing session. VCAddSymbolEntity and VCPlaceSymbol both require the internal name not the on disk name. The internal name can be determined from the saved name with VCGetSymbolInternalName.
See Also	VCAcadBlockRead, VCCreateSymbolDef , VCOpenCMP, VCOpenVCS, VCAddSymbolEntity, VCGetSymName, VCGetSymbolName, VCGetSymbolInternalName

VCTerminate

Version	1.2
Description	Unloads all drawing database and settings from memory and ends the Corel Visual CADD sessions.
Declaration	
C/C++:	extern "C" void WINAPI VCTerminate(void);
Visual Basic:	Declare Sub VCTerminate Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCTerminateDialogs; far;
Parameters	No parameters are used for this subroutine.
Notes	If the Corel Visual CADD DLL have been initialized by VCInit, disable them with the VCTerminate subroutine. This method frees memory for other applications and will prevent the loaded DLL from interfering with the operation of other Corel Visual CADD sessions. Whenever an application is to be loaded independent of the Corel Visual CADD interface, the DLL must be initialized with VCInit. This allows the program to access the internal subroutines and functions. VCGetInitCount is used to determine the number of instances of Corel Visual CADD sessions currently active.
See Also	VCInit, VCGetInitCount, VCTerminateDialogs

VCTerminateDialogs

Version	1.2
Description	Removes all dialogs from memory to keep them from displaying.
Declaration	
C/C++:	extern "C" void WINAPI VCTerminateDialogs(void);
Visual Basic:	Declare Sub VCTerminateDialogs Lib "VCDLG32.DLL" ()
Delphi:	procedure VCTerminateDialogs; far;
Parameters	No parameters are used with this subroutine.
Notes	When building an external application based on the Corel Visual CADD engine, it may, or may not, be desirable to display Corel Visual CADD's internal dialogs. If the external application uses it's own dialogs and passes the values or settings to Corel Visual CADD manually than it probably will not be necessary to use the internal dialogs. If however the external application requires the internal dialogs for consistency, or ease of programming, VCInitDialogs will initialize the dialogs for use while VCTerminateDialogs will terminate their use.
See Also	VCInitDialogs, VCInit, VCTerminate

VCThisNameIsCurrentUser

Version	2.0
Description	Returns the name of the current user.
Declaration	
C/C++	extern "C" vbool WINAPI VCThisNameIsCurrentUser(char* szName);
Visual Basic	Declare Function VCThisNameIsCurrentUser Lib "VCMAIN32.DLL" (ByVal szName As String) As Integer
Delphi	function VCThisNameIsCurrentUser(szName: PChar):Boolean; far;
Parameters	*szName* - the name of the current user from the registry settings.
Notes	The current user is taken from values in the Windows registry. This routine simply returns the name listed in the registry as the licensed user for the active session.
See Also	Registry Settings for Corel Visual CADD

VCTimer

Version	1.2
Description	Sends a message to Corel Visual CADD that a timer message has been received from the system.
Declaration	
C/C++:	extern "C" void WINAPI VCTimer();
Visual Basic:	Declare Sub VCTimer Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCTimer; far;
Parameters	No parameters are used in this subroutine.
Notes	This is only used when an external application receives a WM_TIMER from the system. The application would then call VCTimer in order to notify Corel Visual CADD of the message.
See Also	Windows SDK

VCToggle

Version	1.2
Description	Toggles the state of a setting.
Declaration	
C/C++:	extern "C" void WINAPI VCToggle(WORD id);
Visual Basic:	Declare Sub VCToggle Lib "VCMAIN32.DLL" (ByVal id As Integer)
Delphi:	procedure VCToggle(id: Integer); far;
Parameters	*id* - the command id of the command to be toggled. See Appendix A for a listing of native commands.
Notes	This subroutine is valid only for toggle settings. VCToggle strictly toggles the command opposite of what it was previously, it will not explicitly toggle on or off.
See Also	VCIsToggle, Appendix A

VCTruncFrom

Version	2.0
Description	Truncates the database from the specified entity handle.
Declaration	
C/C++	extern "C" void WINAPI VCTruncFrom(short* iError, ENTITYHANDLE StartHere);
Visual Basic	Declare Sub VCTruncFrom Lib "VCMAIN32.DLL" (iError As Integer, ByVal StartHere As Long)
Delphi	procedure VCTruncFrom(var iError: Integer; StartHere: Longint); far;
Parameters	*StartHere* - the entity handle for the entity to begin truncating from.
Notes	The drawing database maintains all entity operation in the database. This includes erased entity information for undo and redo levels. This data is stored until a pack data command or save. In some situation however an application may need only to add an entity temporarily and not have it remain in the database for undo and redo operations. VCTruncFrom allows an application to truncate the database effectively eliminating items from the drawing and not allowing undo levels to get set. For example, an application may need to display a temporary construction line during operation. Since it is not desirable to maintain the entity in the drawing database the application can truncate the drawing from that point.
See Also	VCClearDrawing, VCClearDrawingNoPrompt, VCPackData, VCPurgeErasedEntities

VCUIOff

Version 1.2

Description Turns the user interface (ribalogs) off.

Declaration

C/C++: extern "C" void WINAPI VCUIOff(void);

Visual Basic: Declare Sub VCUIOff Lib "VCMAIN32.DLL" ()

Delphi: procedure VCUIOff; far;

Parameters No parameters are used with this subroutine.

Notes Several of the modify and entity placement commands use a speedbar to change settings and prompt the user for relevant information. In an external application however, it is not necessarily desirable for these to display. VCUIOff will turn off the display of these ribalogs while VCUIOn will turn them back on. Normally the calling application will make the appropriate settings for whatever command will be executed and with the user interface turned off (VCUIOff) will call the appropriate subroutine. If the application then needs to display the next speedbar, it would make a call to VCUIOn to turn the user interface on. Ribalogs will only display in the Corel Visual CADD drawing environment; they will not attempt to display in a Visual BASIC picture box or any similar environment.

See Also VCUIOn

VCUIOn

Version	1.2
Description	Turns the user interface (ribalogs) on after being turned off.
Declaration	
C/C++:	extern "C" void WINAPI VCUIOn(void);
Visual Basic:	Declare Sub VCUIOn Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCUIOn; far;
Parameters	No parameters are used with this subroutine.
Notes	Several of the modify and entity placement commands use a speedbar to change setting and prompt the user for relevant information. In an external application however, it is not necessarily desirable for these to display. VCUIOff will turn off the display of these Ribalogs while VCUIOn will turn them back on. Normally the calling application will make the appropriate settings for whatever command will be executed and with the user interface turned off (VCUIOff) will call the appropriate subroutine. If the application then needs to display the next speedbar, it would make a call to VCUIOn to turn the user interface on. Ribalogs will only display in the Corel Visual CADD drawing environment; they will not attempt to display in a Visual BASIC picture box or any similar environment.
See Also	VCUIOff

VCUnloadUnusedSymDefs

Version	1.2
Description	Removes unused symbol definitions from the drawing session.
Declaration	
C/C++:	extern "C" void WINAPI VCUnloadUnusedSymDefs(short* iError);
Visual Basic:	Declare Sub VCUnloadUnusedSymDefs Lib "VCMAIN32.DLL" (iError As Integer)
Delphi:	procedure VCUnloadUnusedSymDefs(var iError: Integer); far;
Parameters	No parameters are used with this subroutine.
Notes	A symbol definition can be loaded into the drawing session from disk or created from existing entities. Corel Visual CADD utilizes several symbol formats in addition to the native VCS files. These include AutoCAD block (DWG, DXF) and Generic CADD components (CMP). These are loaded with VCOpenDWG and VCOpenCMP commands. An internal symbol definition can also be created within a drawing sessions using VCCreateSymbolDef and any of the VCAdd*Entity commands. While loaded these symbol definitions may or may not have been used in the drawing session. VCUnloadUnusedSymbolDefs removes all unused symbol definitions and frees any subsequent resources
See Also	VCRemoveSymbol

VCUpdateBirdseyeView

Version	2.0
Description	Updates the Birds Eye image.
Declaration	
C/C++	extern "C" void WINAPI VCUpdateBirdseyeView(short* iError, vbool tfRefresh);
Visual Basic	Declare Sub VCUpdateBirdseyeView Lib "VCDLG32.DLL" (iError As Integer, ByVal tfRefresh As Integer)
Delphi	procedure VCUpdateBirdseyeView(var iError: Integer; tfRefresh: Boolean); far;
Parameters	*tfRefresh* - flag for updating the birds eye view. 0 - do not update the birds eye. 1 - update the birds eye.
Notes	The Birds Eye view provides a thumbnail overall view of the active drawing as Window are placed on top and the view changes it is necessary to update the current Birds Eye view.
See Also	VCZoomAllViews, VCZoomRegenAllViews

VCUpdateStatusBar

Version	1.2
Description	Forces Corel Visual CADD to update the status bar.
Declaration	
C/C++:	extern "C" void WINAPI VCUpdateStatusBar();
Visual Basic:	Declare Sub VCUpdateStatusBar Lib "VCMAIN32.DLL" ()
Delphi:	procedure VCUpdateStatusBar; far;
Parameters	No additional parameters are used with this subroutine.
Notes	The status bar contains several messages about cursor position, units and entry mode. After a command is completed or when a command is nested within another, the status line sometimes does not update to reflect the latest information. VCUpdateStatusBar will ensure the information is current when called.
See Also	VCLockMessage

VCUserMatch

Version	1.2
Description	Initiates the match tool to extract the specified setting.
Declaration	
C/C++:	extern "C" void WINAPI VCUserMatch(WORD id);
Visual Basic:	Declare Sub VCUserMatch Lib "VCTOOL32.DLL" (ByVal id As Integer)
Delphi:	procedure VCUserMatch(id: Integer); far;
Parameters	*id* - the command id of the coresponding tool for the setting to extract.
Notes	When issuing the VCUserMatch command, the id of the tool for which the setting applies needs to be passed as the id. For example, to match the color ColorProp would be passed as the id. See appendix A for a listing on command id's.
See Also	Appendix A, VCMatchTool

VCWindowSelect

Version	1.2
Description	Selects any objects located entirely in the specified window.
Declaration	
C/C++:	extern "C" void WINAPI VCWindowSelect(Point2D* dpP0, Point2D* dpP1);
Visual Basic:	Declare Sub VCWindowSelect Lib "VCMAIN32.DLL" (dpP0 As Point2D, dpP1 As Point2D)
Delphi:	procedure VCWindowSelect(var dpP0: Point2D; var dpP1: Point2D); far;
Parameters	*pP0* - the coordinates of one corner of the window. *dpP1* - the coordinates of the second corner of the window.
Notes	Operates the same as the select window tool except allows for input points from the external application. The application can process the points from a mouse down event or code in the coordinates for the selection routine.
See Also	VCObjectSelect , VCClearSelection, VCCrossingSelect, VCDeSelectAll, VCSelectInvert, VCSelectLast

VCWorld2DToScreen

Version	1.2
Description	Converts the world 2D coordinates into screen values recognizable by the Windows API.
Declaration	
C/C++:	extern "C" void WINAPI VCWorld2DToScreen(short* iError, Point2D* dpWorld, Point2D* dpScreen);
Visual Basic:	Declare Sub VCWorld2DToScreen Lib "VCMAIN32.DLL" (iError As Integer, dpWorld As Point2D, dpScreen As Point2D)
Delphi:	procedure VCWorld2DToScreen(var iError: Integer; var dpWorld: Point2D; dpScreen: Point2D); far;
Parameters	*dpWorld* - the input CAD 2D coordinate. *dpScreen* - the returned screen coordinates.
Notes	When working with other API, it is typically necessary to utilize the screen coordinates for event interaction. Corel Visual CADD however utilizes a world coordinate system based on user settings to reference entities in the drawing. VCWorld2DToScreen converts from this world system into screen coordinate for use in other routines.
See Also	VCWorld3DToView3D, VCWorld3DToWorld2D

VCWorld3DToView3D

Version	1.2
Description	Converts the 3D world coordinate into a 3D coordinate of the current view.
Declaration	
C/C++:	extern "C" void WINAPI VCWorld3DToView3D(short* iError, Point3D* dpWorld3D, Point3D* dpView3D);
Visual Basic:	Declare Sub VCWorld3DToView3D Lib "VCMAIN32.DLL" (iError As Integer, dpWorld3D As Point3D, dpView3D As Point3D)
Delphi:	procedure VCWorld3DToView3D(var iError: Integer; var dpWorld3D: Point3D; dpView3D: Point3D); far;
Parameters	*dpWorld3D* - the input CAD 2D coordinate. *dpView3D* - the returned screen coordinates.
Notes	The view coordinate system for a 3D drawing is represented by the target and eye position. This routine converts a real world 3D point into a point defined by the current target and eye position. Orthogonal projections of the current perspective can then be created from the converted points by eliminating the appropriate plane(i.e. x,y,z=0).
See Also	VCWorld2DToScreen, VCWorld3DToWorld2D

VCWorld3DToWorld2D

Version	1.2
Description	Converts the world 3D coordinates into a world 2D value.
Declaration	
C/C++:	extern "C" void WINAPI VCWorld3DToWorld2D(short* iError, Point3D* dpWorld3D, Point2D* dpWorld2D);
Visual Basic:	Declare Sub VCWorld3DToWorld2D Lib "VCMAIN32.DLL" (iError As Integer, dpWorld3D As Point3D, dpWorld2D As Point2D)
Delphi:	procedure VCWorld3DToWorld2D(var iError: Integer; var dpWorld3D: Point3D; dpWorld2D: Point2D); far;
Parameters	*dpWorld3D* - the input CAD 2D coordinate. *dpWorld2D* - the returned screen coordinates.
Notes	The 2D coordinate returned is projected from the current 3D perspective view. In a flat plane, the routine simply strips the z-axis from the point. In perspective views, the 3D coordinate is projected to a flat plane coordinate system on the screen
See Also	VCWorld2DToScreen, VCWorld3DToWorld2D

VCWriteMetafile

Version	2.0
Description	Saves a Windows metafile of the active drawing.
Declaration	
C/C++	extern "C" void WINAPI VCWriteMetafile(char* pName, short iFileType, vbool tfSelectedOnly);
Visual Basic	Declare Sub VCWriteMetafile Lib "VCTRAN32.DLL" (ByVal pName As String, ByVal iFileType As Integer, ByVal tfSelectedOnly As Integer)
Delphi	procedure VCWriteMetafile(pName: PChar; iFileType: Integer; tfSelectedOnly: Boolean); far;
Parameters	*pName* - the file name and path to save. *TfSelectedOnly* - flag to save only selected entities. 0 - use all entities. 1 - use only selected entities.
See Also	VCAcadWriteDWG, VCAcadWriteDXF, VCSaveDrawing

VCZoomAllViews

Version	2.0
Description	Zooms to the drawing extents for all open viewports of the active drawing.
Declaration	
C/C++	extern "C" vbool WINAPI VCZoomAllViews(short* iError);
Visual Basic	Declare Function VCZoomAllViews Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Delphi	function VCZoomAllViews(var iError: Integer):Boolean; far;
Parameters	No additional parameters are used with this subroutine.
Notes	Corel Visual CADD allows for Multiple Document Interface. The MDI child windows may represent a different drawing or a separate view of an existing drawing. When using multiple views of the same drawing it may be necessary to refresh all data in all views at once. Instead of moving to each drawing world individually and issuing a VCZoomRegen command, all the views can be updated with a single call to VCZoomRegenAllViews. An application can also force a complete update by forcing the views to zoom to their fullest extent with VCZoomAllViews.
See Also	VCZoomRegenAllViewsVCZoomRegenAllViews, VCZoomViewVCZoomView, VCZoomAllTool_Reference

VCZoomRegenAllViews

Version	2.0
Description	Redraws entities for all open viewports of the active drawing.
Declaration	
C/C++	extern "C" vbool WINAPI VCZoomRegenAllViews(short* iError);
Visual Basic	Declare Function VCZoomRegenAllViews Lib "VCMAIN32.DLL" (iError As Integer) As Integer
Delphi	function VCZoomRegenAllViews(var iError: Integer):Boolean; far;
Parameters	No additional parameters are used with this subroutine.
Notes	Corel Visual CADD allows for Multiple Document Interface. The MDI child windows may represent a different drawing or a separate view of an existing drawing. When using multiple views of the same drawing it may be necessary to refresh all data in all views at once. Instead of moving to each drawing world individually and issuing a VCZoomRegen command, all the views can be updated with a single call to VCZoomRegenAllViews. If only one view of the drawing exists then this command behaves the same as a VCZoomRegen and simply redraws the active world.
See Also	VCZoomAllViewsVCZoomAllViews, VCZoomViewVCZoomView, VCZoomAllTool_Reference

VCZoomView

Version	1.2
Description	Displays a screen view previously named using the Named View command or the VCNameView API routine.
Declaration	
C/C++:	extern "C" void WINAPI VCZoomView(char* szView);
Visual Basic:	Declare Sub VCZoomView Lib "VCMAIN32.DLL" (ByVal szView As String)
Delphi:	procedure VCZoomView(szView: PChar); far;
Parameters	*szView* - the name of the named view.
Notes	Named views are useful whenever a particular screen view needs to be accessed repeatedly for drawing or editing operations.
See Also	VCNameView, VCZoomSelected, VCZoomPrevious, VCZoomAll

Tool Reference

This chapter provides a list of commands that do not require parameter input. These routines offer the functionality of the Corel Visual CADD interface but allow no control over the operation.

{button ,JI("vcadd32.hlp","Continuous_Line")} **VCAbort**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCAbort(void);

Visual Basic Declare Sub VCAbort Lib “VCMAIN32.DLL” ()

Delphi procedure VCAbort; far;

{button ,JI("vcadd32.hlp","Continuous_Line")} **VCAdjoiningToMEP**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCAdjoiningToMEP(short* iError);

Visual Basic Declare Sub VCAdjoiningToMEP Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCAdjoiningToMEP(iError:Integer);

{button ,JI("vcadd32.hlp","Align")} **VCAlignSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCAlignSelected();

Visual Basic Declare Sub VCAlignSelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCAlignSelected; far;

{button ,JI("vcadd32.hlp","Angular_Dimension")} **VCAngularDim**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCAngularDim(void);

Visual Basic Declare Sub VCAngularDim Lib “VCTOOL32.DLL” ()

Delphi procedure VCAngularDim; far;

{button ,JI("vcadd32.hlp","Two_Point_Arc")} **VCArc2Pt**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCArc2Pt(void);

Visual Basic Declare Sub VCArc2Pt Lib “VCTOOL32.DLL” ()

Delphi procedure VCArc2Pt; far;

{button ,JI("vcadd32.hlp","Three_Point_Arc")} **VCArc3Pt**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCArc3Pt(void);

Visual Basic Declare Sub VCArc3Pt Lib “VCTOOL32.DLL” ()

Delphi procedure VCArc3Pt; far;

{button ,JI("vcadd32.hlp","Array_Copy")} **VCArrayCopySelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCArrayCopySelected();

Visual Basic Declare Sub VCArrayCopySelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCArrayCopySelected; far;

{button ,JI("vcadd32.hlp","Attach_Attribute")} **VCAttributeAttach**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCAttributeAttach(void);

Visual Basic Declare Sub VCAttributeAttach Lib “VCTOOL32.DLL” ()

Delphi procedure VCAttributeAttach; far;

{button ,JI("vcadd32.hlp","Create_Attribute")} **VCAttributeCreate**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCAttributeCreate(void);

Visual Basic Declare Sub VCAttributeCreate Lib “VCTOOL32.DLL” ()

Delphi procedure VCAttributeCreate; far;

{button ,JI("vcadd32.hlp","Attribute_Edit")} **VCAttributeEdit**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCAttributeEdit(void);

Visual Basic Declare Sub VCAttributeEdit Lib “VCTOOL32.DLL” ()

Delphi procedure VCAttributeEdit; far;

{button ,JI("vcadd32.hlp","Attribute_Embed")} **VCAttributeEmbed**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCAttributeEmbed(void);

Visual Basic Declare Sub VCAttributeEmbed Lib “VCTOOL32.DLL” ()

Delphi procedure VCAttributeEmbed; far;

{button ,JI("vcadd32.hlp","Attribute_Move")} **VCAttributeMove**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCAttributeMove(void);

Visual Basic Declare Sub VCAttributeMove Lib “VCTOOL32.DLL” ()

Delphi procedure VCAttributeMove; far;

{button ,JI("vcadd32.hlp","Attach_Attribute")} **VCAttributeMultiAttach**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCAttributeMultiAttach(void);

Visual Basic Declare Sub VCAttributeMultiAttach Lib “VCTOOL32.DLL” ()

Delphi procedure VCAttributeMultiAttach; far;

{button ,JI("vcadd32.hlp","Boolean")} **VCBooleanAdd**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCBooleanAdd(short* iError);

Visual Basic Declare Sub VCBooleanAdd Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCBooleanAdd(var iError Integer); far;

{button ,JI("vcadd32.hlp","Boolean")} **VCBooleanIntersect**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCBooleanIntersect(short* iError);

Visual Basic Declare Sub VCBooleanIntersect Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCBooleanIntersect(var iError Integer); far;

{button ,JI("vcadd32.hlp","Boolean")} **VCBooleanSubtract**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCBooleanSubtract(short* iError);

Visual Basic Declare Sub VCBooleanSubtract Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCBooleanSubtract(var iError Integer); far;

{button ,JI("vcadd32.hlp","Single_Bezier_Curve")} **VCBezEdit**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCBezEdit();

Visual Basic Declare Sub VCBezEdit Lib “VCTOOL32.DLL” ()

Delphi procedure VCBezEdit; far;

{button ,JI("vcadd32.hlp","Single_Bezier_Curve")} **VCBezierSingle**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCBezierSingle(void);

Visual Basic Declare Sub VCBezierSingle Lib “VCTOOL32.DLL” ()

Delphi procedure VCBezierSingle; far;

{button ,JI("vcadd32.hlp","Fill_Boundary")} **VCBoundaryFill**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCBoundaryFill(void);

Visual Basic Declare Sub VCBoundaryFill Lib “VCTOOL32.DLL” ()

Delphi procedure VCBoundaryFill; far;

{button ,JI("vcadd32.hlp","Hatch_Boundary")} **VCBoundaryHatch**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCBoundaryHatch(void);

Visual Basic Declare Sub VCBoundaryHatch Lib “VCTOOL32.DLL” ()

Delphi procedure VCBoundaryHatch; far;

{button ,JI("vcadd32.hlp","Chamfer")} **VCChamfer**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCChamfer(void);

Visual Basic Declare Sub VCChamfer Lib “VCTOOL32.DLL” ()

Delphi procedure VCChamfer; far;

{button ,JI("vcadd32.hlp","Two_Point_Circle")} **VCCircle2Pt**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCCircle2Pt(void);

Visual Basic Declare Sub VCCircle2Pt Lib “VCTOOL32.DLL” ()

Delphi procedure VCCircle2Pt; far;

{button ,JI("vcadd32.hlp","Three_Point_Circle")} **VCCircle3Pt**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCCircle3Pt(void);

Visual Basic Declare Sub VCCircle3Pt Lib “VCTOOL32.DLL” ()

Delphi procedure VCCircle3Pt; far;

{button ,JI("vcadd32.hlp","Diameter_Circle")} **VCCircleDiameter**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCCircleDiameter(void);

Visual Basic Declare Sub VCCircleDiameter Lib “VCTOOL32.DLL” ()

Delphi procedure VCCircleDiameter; far;

{button ,JI("vcadd32.hlp","Cmd_Cleardrawing")} **VCClearScreen**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCClearScreen(void);

Visual Basic Declare Sub VCClearScreen Lib “VCMAIN32.DLL” ()

Delphi procedure VCClearScreen; far;

{button ,JI("vcadd32.hlp","Clear_Select")} **VCClearSelection**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCClearSelection(void);

Visual Basic Declare Sub VCClearSelection Lib “VCMAIN32.DLL” ()

Delphi procedure VCClearSelection; far;

{button ,JI("vcadd32.hlp","Close_Contour")} **VCCloseContour**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCCloseContour();

Visual Basic Declare Sub VCCloseContour Lib “VCMAIN32.DLL” ()

Delphi procedure VCCloseContour; far;

{button ,JI("vcadd32.hlp","Continuous_Bezier_Curve")} **VCContBezier**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCContBezier(void);

Visual Basic Declare Sub VCContBezier Lib “VCTOOL32.DLL” ()

Delphi procedure VCContBezier; far;

{button ,JI("vcadd32.hlp","Linear_Copy")} **VCCopySelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCCopySelected();

Visual Basic Declare Sub VCCopySelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCCopySelected; far;

{button ,JI("vcadd32.hlp","Trim_Intersection")} **VCCornerTrim**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCCornerTrim(void);

Visual Basic Declare Sub VCCornerTrim Lib “VCTOOL32.DLL” ()

Delphi procedure VCCornerTrim; far;

{button ,JI("vcadd32.hlp","Cut")} **VCCut**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCCut();

Visual Basic Declare Sub VCCut Lib “VCMAIN32.DLL” ()

Delphi procedure VCCut; far;

{button ,JI("vcadd32.hlp","Spline_Curve")} **VCCurve**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCCurve(void);

Visual Basic Declare Sub VCCurve Lib “VCTOOL32.DLL” ()

Delphi procedure VCCurve; far;

{button ,JI("vcadd32.hlp","DATUM_DIM")} **VCDatum**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCDatum(short* iError);

Visual Basic Declare Sub VCDatum Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCDatum(var iError Integer); far;

{button ,JI("vcadd32.hlp","Clear_Select")} **VCDeSelectAll**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDeSelectAll(void);

Visual Basic Declare Sub VCDeSelectAll Lib “VCMAIN32.DLL” ()

Delphi procedure VCDeSelectAll; far;

{button ,JI("vcadd32.hlp","Diameter_Dimension")}**VCDiameterDim**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDiameterDim(void);

Visual Basic Declare Sub VCDiameterDim Lib “VCTOOL32.DLL” ()

Delphi procedure VCDiameterDim; far;

{button ,JI("vcadd32.hlp","Align_Drawing")} **VCDigConfig**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDigConfig();

Visual Basic Declare Sub VCDigConfig Lib “VCTOOL32.DLL” ()

Delphi procedure VCDigConfig; far;

{button ,JI("vcadd32.hlp","Align_Drawing")} **VCDigDrawingAlign**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDigDrawingAlign();

Visual Basic Declare Sub VCDigDrawingAlign Lib “VCTOOL32.DLL” ()

Delphi procedure VCDigDrawingAlign; far;

{button ,JI("vcadd32.hlp","Dim_Edit")} **VCDimEdit**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDimEdit(void);

Visual Basic Declare Sub VCDimEdit Lib “VCTOOL32.DLL” ()

Delphi procedure VCDimEdit; far;

{button ,JI("vcadd32.hlp","Dimension_Arc_Move")} **VCDimMoveArc**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDimMoveArc(void);

Visual Basic Declare Sub VCDimMoveArc Lib “VCTOOL32.DLL” ()

Delphi procedure VCDimMoveArc; far;

{button ,JI("vcadd32.hlp","Dim_Edit")} **VCDimMoveLine**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDimMoveLine(void);

Visual Basic Declare Sub VCDimMoveLine Lib “VCTOOL32.DLL” ()

Delphi procedure VCDimMoveLine; far;

{button ,JI("vcadd32.hlp","Dimension_Text_Move")} **VCDimMoveText**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDimMoveText(void);

Visual Basic Declare Sub VCDimMoveText Lib “VCTOOL32.DLL” ()

Delphi procedure VCDimMoveText; far;

{button ,JI("vcadd32.hlp","DATUM_DIM")} **VCDimPoint**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCDimPoint(short* iError);

Visual Basic Declare Sub VCDimPoint Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCDimPoint(var iError Integer); far;

{button ,JI("vcadd32.hlp","Dimension_Text_Slide")} **VCDimSlideText**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDimSlideText(void);

Visual Basic Declare Sub VCDimSlideText Lib “VCTOOL32.DLL” ()

Delphi procedure VCDimSlideText; far;

{button ,JI("vcadd32.hlp","Change")} **VCEdit**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCEdit();

Visual Basic Declare Sub VCEdit Lib "VCTOOL32.DLL" ()

Delphi procedure VCEdit; far;

{button ,JI("vcadd32.hlp","Ellipse")} **VCEllipse**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCEllipse(void);

Visual Basic Declare Sub VCEllipse Lib “VCTOOL32.DLL” ()

Delphi procedure VCEllipse; far;

{button ,JI("vcadd32.hlp","Elliptical_Start_Span_Arc")} **VCEllipticalArcStartSpan**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCEllipticalArcStartSpan(void);

Visual Basic Declare Sub VCEllipticalArcStartSpan Lib “VCTOOL32.DLL” ()

Delphi procedure VCEllipticalArcStartSpan; far;

{button ,JI("vcadd32.hlp","Erase_Last")} **VCEraseLast**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCEraseLast();

Visual Basic Declare Sub VCEraseLast Lib “VCTOOL32.DLL” ()

Delphi procedure VCEraseLast; far;

{button ,JI("vcadd32.hlp","Erase")} **VCEraseSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCEraseSelected();

Visual Basic Declare Sub VCEraseSelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCEraseSelected; far;

{button ,JI("vcadd32.hlp","Explode")} **VCExplode**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCExplode(void);

Visual Basic Declare Sub VCExplode Lib “VCTOOL32.DLL” ()

Delphi procedure VCExplode; far;

{button ,JI("vcadd32.hlp","Extend_Single")} **VCExtend**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCExtend(void);

Visual Basic Declare Sub VCExtend Lib “VCTOOL32.DLL” ()

Delphi procedure VCExtend; far;

{button ,JI("vcadd32.hlp","Fill_Selection")} **VCFillSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCFillSelected(void);

Visual Basic Declare Sub VCFillSelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCFillSelected; far;

{button ,JI("vcadd32.hlp","Fillet")} **VCFillet**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCFillet(void);

Visual Basic Declare Sub VCFillet Lib “VCTOOL32.DLL” ()

Delphi procedure VCFilletRadiusRibalog; far;

{button ,JI("vcadd32.hlp","Fit_Scale")} **VCFitScaleSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCFitScaleSelected();

Visual Basic Declare Sub VCFitScaleSelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCFitScaleSelected; far;

{button ,JI("vcadd32.hlp","Grid_Origin")} **VCGridOrigin**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCGridOrigin(void);

Visual Basic Declare Sub VCGridOrigin Lib “VCTOOL32.DLL” ()

Delphi procedure VCGridOrigin; far;

{button ,JI("vcadd32.hlp","Hatch_Selection")} **VCHatchSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCHatchSelected(void);

Visual Basic Declare Sub VCHatchSelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCHatchSelected; far;

{button ,JI("vcadd32.hlp","Irregular_Polygon")} **VCIrregularPolygon**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCIrregularPolygon(void);

Visual Basic Declare Sub VCIrregularPolygon Lib “VCTOOL32.DLL” ()

Delphi procedure VCIrregularPolygon; far;

{button ,JI("vcadd32.hlp","Leader")} **VCLeader**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCLeader(void);

Visual Basic Declare Sub VCLeader Lib “VCTOOL32.DLL” ()

Delphi procedure VCLeader; far;

{button ,JI("vcadd32.hlp","Leader")} **VCLeaderEdit**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCLeaderEdit(void);

Visual Basic Declare Sub VCLeaderEdit Lib "VCTOOL32.DLL" ()

Delphi procedure VCLeaderEdit; far;

{button ,JI("vcadd32.hlp","Single_Line")} **VCLine**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCLine(void);

Visual Basic Declare Sub VCLine Lib “VCTOOL32.DLL”()

Delphi procedure VCLine; far;

{button ,JI("vcadd32.hlp","Continuous_Line")} **VCLineContinuous**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCLineContinuous(void);

Visual Basic Declare Sub VCLineContinuous Lib “VCTOOL32.DLL” ()

Delphi procedure VCLineContinuous; far;

{button ,JI("vcadd32.hlp","Linear_Dimension")} **VCLinearDim**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCLinearDim(void);

Visual Basic Declare Sub VCLinearDim Lib “VCTOOL32.DLL” ()

Delphi procedure VCLinearDim; far;

{button ,JI("vcadd32.hlp","Customizing_Line_Types")} **VCLoadLnt**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCLoadLnt();

Visual Basic Declare Sub VCLoadLnt Lib "VCTOOL32.DLL" ()

Delphi procedure VCLoadLnt; far;

{button ,JI("vcadd32.hlp","Match_Entity")} **VCMatchEntity**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMatchEntity();

Visual Basic Declare Sub VCMatchEntity Lib “VCTOOL32.DLL” ()

Delphi procedure VCMatchEntity; far;

{button ,JI("vcadd32.hlp","Match_Tool")} **VCMatchTool**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMatchTool();

Visual Basic Declare Sub VCMatchTool Lib “VCTOOL32.DLL” ()

Delphi procedure VCMatchTool; far;

{button ,JI("vcadd32.hlp","Measure")} **VCMeasureAngle2**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMeasureAngle2(void);

Visual Basic Declare Sub VCMeasureAngle2 Lib “VCTOOL32.DLL” ()

Delphi procedure VCMeasureAngle2; far;

{button ,JI("vcadd32.hlp","Measure")} **VCMeasureAngle3**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMeasureAngle3(void);

Visual Basic Declare Sub VCMeasureAngle3 Lib “VCTOOL32.DLL” ()

Delphi procedure VCMeasureAngle3; far;

{button ,JI("vcadd32.hlp","Measure_Area")} **VCMeasureArea**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMeasureArea(void);

Visual Basic Declare Sub VCMeasureArea Lib “VCTOOL32.DLL” ()

Delphi procedure VCMeasureAreaRibalog; far;

{button ,JI("vcadd32.hlp","Measure")} **VCMeasureDistance**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMeasureDistance(void);

Visual Basic Declare Sub VCMeasureDistance Lib “VCTOOL32.DLL” ()

Delphi procedure VCMeasureDistance; far;

{button ,JI("vcadd32.hlp","Mirror")} **VCMirrorSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMirrorSelected();

Visual Basic Declare Sub VCMirrorSelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCMirrorSelected; far;

{button ,JI("vcadd32.hlp","Double_Line")} **VCMLine**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMLine(void);

Visual Basic Declare Sub VCMLine Lib “VCTOOL32.DLL” ()

Delphi procedure VCMLine; far;

{button ,JI("vcadd32.hlp","Break")} **VCModBreak**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCModBreak(void);

Visual Basic Declare Sub VCModBreak Lib “VCTOOL32.DLL” ()

Delphi procedure VCModBreak; far;

{button ,JI("vcadd32.hlp","Move_Point")} **VCMovePoint**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMovePoint(void);

Visual Basic Declare Sub VCMovePoint Lib “VCTOOL32.DLL” ()

Delphi procedure VCMovePoint; far;

{button ,JI("vcadd32.hlp","Move")} **VCMoveSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMoveSelected();

Visual Basic Declare Sub VCMoveSelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCMoveSelected; far;

{button ,JI("vcadd32.hlp","Multiple_Copy")} **VCMultipleCopy**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCMultipleCopy();

Visual Basic Declare Sub VCMultipleCopy Lib "VCTOOL32.DLL" ()

Delphi procedure VCMultipleCopy; far;

{button ,JI("vcadd32.hlp","New_Handle")} **VCNewHandle**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCNewHandle();

Visual Basic Declare Sub VCNewHandle Lib “VCTOOL32.DLL” ()

Delphi procedure VCNewHandle; far;

{button ,JI("vcadd32.hlp","Offset")} **VCOffset**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCOffsetMEP(short* iError);

Visual Basic Declare Sub VCOffset Lib “VCTOOL32.DLL” ()

Delphi procedure VCOffsetRibalog; far;

{button ,JI("vcadd32.hlp","Offset")} **VCOffsetPnt**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCOffsetPnt();

Visual Basic Declare Sub VCOffsetPnt Lib “VCTOOL32.DLL” ()

Delphi procedure VCOffsetPnt; far;

{button ,JI("vcadd32.hlp","ORD_DIM")} **VCOrdinateDim**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCOrdinateDim(short* iError);

Visual Basic Declare Sub VCOrdinateDim Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCOrdinateDim(var iError Integer); far;

{button ,JI("vcadd32.hlp","Paste")} **VCPaste**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCPaste();

Visual Basic Declare Sub VCPaste Lib “VCTOOL32.DLL” ()

Delphi procedure VCPaste; far;

{button ,JI("vcadd32.hlp","Continuous_Line")} **VCPenUp**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCPenUp(void);

Visual Basic Declare Sub VCPenUp Lib “VCMAIN32.DLL” ()

Delphi procedure VCPenUp; far;

{button ,JI("vcadd32.hlp","Place_Symbol")} **VCPlaceCurrentSymbol**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCPlaceCurrentSymbol();

Visual Basic Declare Sub VCPlaceCurrentSymbol Lib “VCTOOL32.DLL” ()

Delphi procedure VCPlaceCurrentSymbol; far;

{button ,JI("vcadd32.hlp","Point")} **VCPoint**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCPoint(void);

Visual Basic Declare Sub VCPoint Lib “VCTOOL32.DLL” ()

Delphi procedure VCPoint; far;

{button ,JI("vcadd32.hlp"," Customizing_Line_Types")} **VCPurgeLnt**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCPurgeLnt();

Visual Basic Declare Sub VCPurgeLnt Lib "VCTOOL32.DLL" ()

Delphi procedure VCPurgeLnt; far;

{button ,JI("vcadd32.hlp","CREATE_REFFRAME")} **VCRFSize**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFSize(short* iError);

Visual Basic Declare Sub VCRFSize Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRFSize(var iError Integer); far;

{button ,JI("vcadd32.hlp","Zoom_All")} **VCRFZoomAll**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFZoomAll(short* iError);

Visual Basic Declare Sub VCRFZoomAll Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRFZoomAll(var iError Integer); far;

{button ,JI("vcadd32.hlp","Zoom_Window")} **VCRFZoomArea**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFZoomArea(short* iError);

Visual Basic Declare Sub VCRFZoomArea Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRFZoomArea(var iError Integer); far;

{button ,JI("vcadd32.hlp","Zoom_In")} **VCRFZoomIn**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFZoomIn(short* iError);

Visual Basic Declare Sub VCRFZoomIn Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRFZoomIn(var iError Integer); far;

{button ,JI("vcadd32.hlp","Zoom_Out")} **VCRFZoomOut**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFZoomOut(short* iError);

Visual Basic Declare Sub VCRFZoomOut Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRFZoomOut(var iError Integer); far;

{button ,JI("vcadd32.hlp","Pan")} **VCRFZoomPan**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFZoomPan(short* iError);

Visual Basic Declare Sub VCRFZoomPan Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRFZoomPan(var iError Integer); far;

{button ,JI("vcadd32.hlp","Zoom_Previous")} **VCRFZoomPrevious**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFZoomPrevious(short* iError);

Visual Basic Declare Sub VCRFZoomPrevious Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRFZoomPrevious(var iError Integer); far;

{button ,JI("vcadd32.hlp","Redraw")} **VCRFZoomRegen**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFZoomRegen(short* iError);

Visual Basic Declare Sub VCRFZoomRegen Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRFZoomRegen(var iError Integer); far;

{button ,JI("vcadd32.hlp","Zoom_Value")} **VCRFZoomValue**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFZoomValue(short* iError);

Visual Basic Declare Sub VCRFZoomValue Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRFZoomValue(var iError Integer); far;

{button ,JI("vcadd32.hlp","Zoom_View")} **VCRFZoomView**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFZoomView(short* iError);

Visual Basic Declare Sub VCRFZoomView Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRFZoomView(var iError Integer); far;

{button ,JI("vcadd32.hlp","CREATE_REFFRAME")} **VCRefFrameCreate**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRefFrameCreate(short* iError);

Visual Basic Declare Sub VCRefFrameCreate Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRefFrameCreate(var iError Integer); far;

{button ,JI("vcadd32.hlp","PLACE_REFFRAME")} **VCRefFramePlace**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRefFramePlace(short* iError);

Visual Basic Declare Sub VCRefFramePlace Lib "VCTOOL32.DLL" (iError As Integer)

Delphi procedure VCRefFramePlace(var iError Integer); far;

{button ,JI("vcadd32.hlp","Center_Polygon")} **VCRPolygonCenter**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCRPolygonCenter(void);

Visual Basic Declare Sub VCRPolygonCenter Lib “VCTOOL32.DLL” ()

Delphi procedure VCRPolygonCenter; far;

{button ,JI("vcadd32.hlp","Side_Polygon")} **VCRPolygonSide**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCRPolygonSide(void);

Visual Basic Declare Sub VCRPolygonSide Lib “VCTOOL32.DLL” ()

Delphi procedure VCRPolygonSide; far;

{button ,JI("vcadd32.hlp","Radial_Copy")} **VCRadCopySelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCRadCopySelected();

Visual Basic Declare Sub VCRadCopySelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCRadCopySelected; far;

{button ,JI("vcadd32.hlp","Radial_Dimension")} **VCRadialDim**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCRadialDim(void);

Visual Basic Declare Sub VCRadialDim Lib “VCTOOL32.DLL” ()

Delphi procedure VCRadialDim; far;

{button ,JI("vcadd32.hlp","Two_Point_Rectangle")} **VCRectangle2Pt**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCRectangle2Pt(void);

Visual Basic Declare Sub VCRectangle2Pt Lib “VCTOOL32.DLL” ()

Delphi procedure VCRectangle2Pt; far;

{button ,JI("vcadd32.hlp","Three_Point_Rectangle")} **VCRectangle3Pt**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCRectangle3Pt(void);

Visual Basic Declare Sub VCRectangle3Pt Lib “VCTOOL32.DLL” ()

Delphi procedure VCRectangle3Pt; far;

{button ,JI("vcadd32.hlp","Redo")} **VCRedo**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCRedo(void);

Visual Basic Declare Sub VCRedo Lib “VCTOOL32.DLL” ()

Delphi procedure VCRedo; far;

{button ,JI("vcadd32.hlp","Redraw_Window")} **VCRegenArea**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCRegenArea(void);

Visual Basic Declare Sub VCRegenArea Lib “VCTOOL32.DLL” ()

Delphi procedure VCRegenArea; far;

{button ,JI("vcadd32.hlp","Rotate")} **VCRotateSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCRotateSelected();

Visual Basic Declare Sub VCRotateSelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCRotateSelected; far;

{button ,JI("vcadd32.hlp","Scale")} **VCScaleSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCScaleSelected();

Visual Basic Declare Sub VCScaleSelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCScaleSelected; far;

{button ,JI("vcadd32.hlp","Seed_Fill")} **VCSeedFill**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCSeedFill();

Visual Basic Declare Sub VCSeedFill Lib "VCTOOL32.DLL" ()

Delphi procedure VCSeedFill; far;

{button ,JI("vcadd32.hlp","Seed_Hatch")} **VCSeedHatch**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCSeedHatch();

Visual Basic Declare Sub VCSeedHatch Lib "VCTOOL32.DLL" ()

Delphi procedure VCSeedHatch; far;

{button ,JI("vcadd32.hlp","Select")} **VCSelect**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCSelect(void);

Visual Basic Declare Sub VCSelectl Lib “VCMAIN32.DLL” ()

Delphi procedure VCSelect; far;

{button ,JI("vcadd32.hlp","Select_Adj")} **VCSelectAdjoining**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectAdjoining(void);

Visual Basic Declare Sub VCSelectAdjoining Lib “VCTOOL32.DLL” ()

Delphi procedure VCSelectAdjoining; far;

{button ,JI("vcadd32.hlp","Select_All")} **VCSelectAll**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectAll(void);

Visual Basic Declare Sub VCSelectAll Lib “VCMAIN32.DLL” ()

Delphi procedure VCSelectAll; far;

{button ,JI("vcadd32.hlp","Select_Crossing")} **VCSelectCrossingWindow**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectCrossingWindow(void);

Visual Basic Declare Sub VCSelectCrossingWindow Lib “VCTOOL32.DLL” ()

Delphi procedure VCSelectCrossingWindow; far;

{button ,JI("vcadd32.hlp","Invert_Select")} **VCSelectInvert**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectInvert(void);

Visual Basic Declare Sub VCSelectInvert Lib “VCMAIN32.DLL” ()

Delphi procedure VCSelectInvert; far;

{button ,JI("vcadd32.hlp","Select_Last")} **VCSelectLast**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectLast(void);

Visual Basic Declare Sub VCSelectLast Lib “VCMAIN32.DLL” ()

Delphi procedure VCSelectLastEntity; far;

{button ,JI("vcadd32.hlp","Select_Last")} **VCSelectLastEntity**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectLastEntity();

Visual Basic Declare Sub VCSelectLastEntity Lib “VCTOOL32.DLL” ()

Delphi procedure VCSelectLastEntity; far;

{button ,JI("vcadd32.hlp","Select_Last")} **VCSelectLastObject**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectLastObject(short iError);

Visual Basic Declare Sub VCSelectLastObject Lib “VCTOOL32.DLL” (ByVal iError as Integer)

Delphi procedure VCSelectLastObject;(var iError: Integer) far;

{button ,JI("vcadd32.hlp","Select")} **VCSelectObject**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectObject(void);

Visual Basic Declare Sub VCSelectObject Lib “VCTOOL32.DLL” ()

Delphi procedure VCSelectObject; far;

{button ,JI("vcadd32.hlp","Select_Window")} **VCSelectWindow**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectWindow(void);

Visual Basic Declare Sub VCSelectWindow Lib “VCTOOL32.DLL” ()

Delphi procedure VCSelectWindow; far;

{button ,JI("vcadd32.hlp","Extend_Multiple")} **VCSelectionExtend**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectionExtend(void);

Visual Basic Declare Sub VCSelectionExtend Lib “VCTOOL32.DLL” ()

Delphi procedure VCSelectionExtend; far;

{button ,JI("vcadd32.hlp","Trim_Multiple")} **VCSelectionTrim**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSelectionTrim(void);

Visual Basic Declare Sub VCSelectionTrim Lib “VCTOOL32.DLL” ()

Delphi procedure VCSelectionTrim; far;

{button ,JI("vcadd32.hlp","Set_Basepoint")} **VCSetBasepoint**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSetBasepoint(void);

Visual Basic Declare Sub VCSetBasepoint Lib “VCTOOL32.DLL” ()

Delphi procedure VCSetBasepoint; far;

{button ,JI("vcadd32.hlp","Snap_Center")} **VCSnapArcCenter**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapArcCenter(void);

Visual Basic Declare Sub VCSnapArcCenter Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapArcCenter; far;

{button ,JI("vcadd32.hlp","Snap_Object")} **VCSnapCloseGeom**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapCloseGeom(void);

Visual Basic Declare Sub VCSnapCloseGeom Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapCloseGeom; far;

{button ,JI("vcadd32.hlp","Snap_Closest")} **VCSnapClosestPoint**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapClosestPoint(void);

Visual Basic Declare Sub VCSnapClosestPoint Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapClosestPoint; far;

{button ,JI("vcadd32.hlp","Snap_Closest")} **VCSnapEndPoint**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapEndPoint(void);

Visual Basic Declare Sub VCSnapEndPoint Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapEndPoint; far;

{button ,JI("vcadd32.hlp","Snap_Intersection")} **VCSnapInt**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapInt(void);

Visual Basic Declare Sub VCSnapInt Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapInt; far;

{button ,JI("vcadd32.hlp","Snap_Last_Point")} **VCSnapLastPoint**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapLastPoint(long lParam, WORD wParam);

Visual Basic Declare Sub VCSnapLastPoint Lib “VCTOOL32.DLL” (ByVal lParam As Long, ByVal wParam As Integer)

Delphi procedure VCSnapLastPoint(lParam Longint; wParam Integer); far;

{button ,JI("vcadd32.hlp","Snap_Between_2_Points")} **VCSnapMid2Points**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapMid2Points(void);

Visual Basic Declare Sub VCSnapMid2Points Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapMid2Points; far;

{button ,JI("vcadd32.hlp","Snap_Midpoint")} **VCSnapMidPoint**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapMidPoint(void);

Visual Basic Declare Sub VCSnapMidPoint Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapMidPoint; far;

{button ,JI("vcadd32.hlp","Snap_Near_Point")} **VCSnapNearPoint**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapNearPoint(long lParam, WORD wParam);

Visual Basic Declare Sub VCSnapNearPoint Lib “VCTOOL32.DLL” (ByVal lParam As Long, ByVal wParam As Integer)

Delphi procedure VCSnapNearPoint(lParam Longint; wParam Integer); far;

{button ,JI("vcadd32.hlp","Snap_Parallel")} **VCSnapParallel**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapParallel(void);

Visual Basic Declare Sub VCSnapParallel Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapParallel; far;

{button ,JI("vcadd32.hlp","Snap_Percentage")} **VCSnapPercent**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapPercent();

Visual Basic Declare Sub VCSnapPercent Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapPercent; far;

{button ,JI("vcadd32.hlp","Snap_Perpendicular")} **VCSnapPerpendicular**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapPerpendicular(void);

Visual Basic Declare Sub VCSnapPerpendicular Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapPerpendicular; far;

{button ,JI("vcadd32.hlp","Snap_Quadrant")} **VCSnapQuad**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapQuad(void);

Visual Basic Declare Sub VCSnapQuad Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapQuad; far;

{button ,JI("vcadd32.hlp","Snap_Tangent")} **VCSnapTangent**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSnapTangent(void);

Visual Basic Declare Sub VCSnapTangent Lib “VCTOOL32.DLL” ()

Delphi procedure VCSnapTangent; far;

{button ,JI("vcadd32.hlp","Stretch")} **VCStretchSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCStretchSelected();

Visual Basic Declare Sub VCStretchSelected Lib “VCTOOL32.DLL” ()

Delphi procedure VCStretchSelected; far;

{button ,JI("vcadd32.hlp","Cmd_Symcreate")} **VCSymbolCreate**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSymbolCreate();

Visual Basic Declare Sub VCSymbolCreate Lib “VCTOOL32.DLL” ()

Delphi procedure VCSymbolCreate; far;

{button ,JI("vcadd32.hlp","Explode_Symbol")} **VCSymbolExplode**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSymbolExplode(void);

Visual Basic Declare Sub VCSymbolExplode Lib “VCTOOL32.DLL” ()

Delphi procedure VCSymbolExplode; far;

{button ,JI("vcadd32.hlp","Text_Editor")} **VCTextManager**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCTextManager();

Visual Basic Declare Sub VCTextManager Lib “VCTOOL32.DLL” ()

Delphi procedure VCTextManager; far;

{button ,JI("vcadd32.hlp","Text_Line")} **VCTextTool**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCTextTool();

Visual Basic Declare Sub VCTextTool Lib “VCTOOL32.DLL” ()

Delphi procedure VCTextTool; far;

{button ,JI("vcadd32.hlp","Tracking")} **VCTracking**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCTracking();

Visual Basic Declare Sub VCTracking Lib “VCTOOL32.DLL” ()

Delphi procedure VCTracking; far;

{button ,JI("vcadd32.hlp","Trim_Single")} **VCTrim**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCTrim(void);

Visual Basic Declare Sub VCTrim Lib “VCTOOL32.DLL” ()

Delphi procedure VCTrim; far;

{button ,JI("vcadd32.hlp","Undo")} **VCUndo**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCUndo(void);

Visual Basic Declare Sub VCUndo Lib “VCTOOL32.DLL” ()

Delphi procedure VCUndo; far;

{button ,JI("vcadd32.hlp","Undo_Vertex")} **VCUndoLastVertex**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCUndoLastVertex(void);

Visual Basic Declare Sub VCUndoLastVertex Lib “VCTOOL32.DLL” ()

Delphi procedure VCUndoLastVertex; far;

{button ,JI("vcadd32.hlp","Single_Line")} **VCUpdateTool**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCUpdateTool();

Visual Basic Declare Sub VCUpdateTool Lib “VCTOOL32.DLL” ()

Delphi procedure VCUpdateTool; far;

{button ,JI("vcadd32.hlp","Stretch")} **VCWindowStretch**

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCWindowStretch();

Visual Basic Declare Sub VCWindowStretch Lib "VCTOOL32.DLL" ()

Delphi procedure VCWindowStretch; far;

{button ,JI("vcadd32.hlp","Zoom_All")} **VCZoomAll**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCZoomAll(void);

Visual Basic Declare Sub VCZoomAll Lib “VCMAIN32.DLL” ()

Delphi procedure VCZoomAll; far;

{button ,JI("vcadd32.hlp","Zoom_Window")} **VCZoomArea**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCZoomArea(void);

Visual Basic Declare Sub VCZoomArea Lib “VCTOOL32.DLL” ()

Delphi procedure VCZoomArea; far;

{button ,JI("vcadd32.hlp","Zoom_In")} **VCZoomIn**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCZoomIn(void);

Visual Basic Declare Sub VCZoomIn Lib “VCTOOL32.DLL” ()

Delphi procedure VCZoomIn; far;

{button ,JI("vcadd32.hlp","Zoom_Out")} **VCZoomOut**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCZoomOut(void);

Visual Basic Declare Sub VCZoomOut Lib “VCTOOL32.DLL” ()

Delphi procedure VCZoomOut; far;

{button ,JI("vcadd32.hlp","Pan")} **VCZoomPan**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCZoomPan(void);

Visual Basic Declare Sub VCZoomPan Lib “VCTOOL32.DLL” ()

Delphi procedure VCZoomPan; far;

{button ,JI("vcadd32.hlp","Zoom_Previous")} **VCZoomPrevious**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCZoomPrevious(void);

Visual Basic Declare Sub VCZoomPrevious Lib “VCMAIN32.DLL” ()

Delphi procedure VCZoomPrevious; far;

{button ,JI("vcadd32.hlp","Redraw")} **VCZoomRegen**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCZoomRegen(void);

Visual Basic Declare Sub VCZoomRegen Lib “VCMAIN32.DLL” ()

Delphi procedure VCZoomRegen; far;

{button ,JI("vcadd32.hlp","Zoom_Selected")} **VCZoomSelected**

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCZoomSelected(void);

Visual Basic Declare Sub VCZoomSelected Lib “VCMAIN32.DLL” ()

Delphi procedure VCZoomSelected; far;

{button ,JI("vcadd32.hlp","Zoom_Value")} **VCZoomValue**

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCZoomValue(void);
Visual Basic	Declare Sub VCZoomValue Lib “VCTOOL32.DLL” ()
Delphi	procedure VCZoomValue; far;

Dialog Reference

This chapter focuses on using the Corel Visual CADD interface to enhance your application. Corel Visual CADD relies on ribalogs displayed during a command operation. The Application Programming Interface allows the use of built in ribalog and dialog boxes or custom ribalogs from an external application.

VCChangeRefFrameNameDlg

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCChangeRefFrameNameDlg(short* iError);

Visual Basic Declare Sub VCChangeRefFrameNameDlg Lib "VCDLG32.DLL" (iError As Integer)

Delphi procedure VCChangeRefFrameNameDlg(var iError: Integer); far;

VCChangeRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCChangeRibalog();

Visual Basic Declare Sub VCChangeRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCChangeRibalog; far;

VCCloseRibalog

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCCloseRibalog(short* iError);

Visual Basic Declare Sub VCCloseRibalog Lib "VCDLG32.DLL" (iError As Integer)

Delphi procedure VCCloseRibalog(var iError: Integer); far;

VCDBLineSettingsRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDBLineSettingsRibalog();

Visual Basic Declare Sub VCDBLineSettingsRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCDBLineSettingsRibalog; far;

VCDimArrowRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDimArrowRibalog();

Visual Basic Declare Sub VCDimArrowRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCDimArrowRibalog; far;

VCDimDisplayRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCDimDisplayRibalog();

Visual Basic Declare Sub VCDimDisplayRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCDimDisplayRibalog; far;

VCDimExtRibalog

Version 1.2

Declaration

C/C++	extern "C" void WINAPI VCDimExtRibalog();
Visual Basic	Declare Sub VCDimExtRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCDimExtRibalog; far;

VCDimLeaderRibalog

Version	1.2
Declaration	
C/C++:	extern "C" void WINAPI VCDimLeaderRibalog();
Visual Basic:	Declare Sub VCDimLeaderRibalog Lib “VCDLG32.DLL” ()
Delphi:	procedure VCDimLeaderRibalog; far;

VCDimLineRibalog

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCDimLineRibalog();
Visual Basic	Declare Sub VCDimLineRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCDimLineRibalog; far;

VCDimStringsRibalog

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCDimStringsRibalog();
Visual Basic	Declare Sub VCDimStringsRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCDimStringsRibalog; far;

VCDimTextRibalog

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCDimTextRibalog();
Visual Basic	Declare Sub VCDimTextRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCDimTextRibalog; far;

VCDimToleranceRibalog

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCDimToleranceRibalog();
Visual Basic	Declare Sub VCDimToleranceRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCDimToleranceRibalog; far;

VCFilletRadiusRibalog

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCFilletRadiusRibalog();
Visual Basic	Declare Sub VCFilletRadiusRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCFilletRadiusRibalog; far;

VCFilterRibalog

Version	1.2
Declaration	

C/C++ extern "C" void WINAPI VCFilterRibalog();

Visual Basic Declare Sub VCFilterRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCFilterRibalog; far;

VCHatchSettingsRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCHatchSettingsRibalog();

Visual Basic Declare Sub VCHatchSettingsRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCHatchSettingsRibalog; far;

VCLayerMgr

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCLayerMgr(short* iError);

Visual Basic Declare Sub VCLayerMgr Lib “VCDLG32.DLL” (iError As Integer)

Delphi procedure VCLayerMgr(var iError: Integer); far;

VCMeasureAngleRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMeasureAngleRibalog();

Visual Basic Declare Sub VCMeasureAngleRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCMeasureAngleRibalog; far;

VCMeasureAreaRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMeasureAreaRibalog();

Visual Basic Declare Sub VCMeasureAreaRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCMeasureAreaRibalog; far;

VCMeasureDistRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCMeasureDistRibalog();

Visual Basic Declare Sub VCMeasureDistRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCMeasureDistRibalog; far;

VCObjectInfo

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCObjectInfo(short* iError);

Visual Basic Declare Sub VCObjectInfo Lib “VCDLG32.DLL” (iError As Integer)

Delphi procedure VCObjectInfo(var iError: Integer); far;

VCOffsetRibalog

Version 1.2

Declaration

C/C++	extern "C" void WINAPI VCOffsetRibalog();
Visual Basic	Declare Sub VCOffsetRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCOffsetRibalog; far;

VCOrthoAngleRibalog

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCOrthoAngleRibalog();
Visual Basic	Declare Sub VCOrthoAngleRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCOrthoAngleRibalog; far;

VCPlot

Version	2.0
Declaration	
C/C++	extern "C" void WINAPI VCPlot(short* iError);
Visual Basic	Declare Sub VCPlot Lib “VCDLG32.DLL” (iError As Integer)
Delphi	procedure VCPlot(var iError: Integer); far;

VCPlotDlg

Version	2.0
Declaration	
C/C++	extern "C" void WINAPI VCPlotDlg(short* iError);
Visual Basic	Declare Sub VCPlotDlg Lib “VCDLG32.DLL” (iError As Integer)
Delphi	procedure VCPlotDlg(var iError: Integer); far;

VCPrint

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCPrint(short* iError);
Visual Basic	Declare Sub VCPrint Lib “VCDLG32.DLL” (iError As Integer)
Delphi	procedure VCPrint(var iError: Integer); far;

VCPrintDlg

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCPrintDlg(short* iError);
Visual Basic	Declare Sub VCPrintDlg Lib “VCDLG32.DLL” (iError As Integer)
Delphi	procedure VCPrintDlg(var iError: Integer); far;

VCPropertiesRibalog

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCPropertiesRibalog();
Visual Basic	Declare Sub VCPropertiesRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCPropertiesRibalog; far;

VCRFBirdsEye

Version	2.0
Declaration	

C/C++ extern "C" void WINAPI VCRFBirdsEye(short* iError);

Visual Basic Declare Sub VCRFBirdsEye Lib “VCDLG32.DLL” (iError As Integer)

Delphi procedure VCRFBirdsEye(var iError: Integer); far;

VCRFLayerMgr

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCRFLayerMgr(short* iError);

Visual Basic Declare Sub VCRFLayerMgr Lib “VCDLG32.DLL” (iError As Integer)

Delphi procedure VCRFLayerMgr(var iError: Integer); far;

VCSetRefFrameNameDlg

Version 2.0

Declaration

C/C++ extern "C" void WINAPI VCSetRefFrameNameDlg(short* iError);

Visual Basic Declare Sub VCSetRefFrameNameDlg Lib "VCDLG32.DLL" (iError As Integer)

Delphi procedure VCSetRefFrameNameDlg(var iError: Integer); far;

VCScriptAssignRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCScriptAssignRibalog();

Visual Basic Declare Sub VCScriptAssignRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCScriptAssignRibalog; far;

VCSymbolMgr

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSymbolMgr(short* iError);

Visual Basic Declare Sub VCSymbolMgr Lib “VCDLG32.DLL” (iError As Integer)

Delphi procedure VCSymbolMgr(var iError: Integer); far;

VCSymCreateRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSymCreateRibalog();

Visual Basic Declare Sub VCSymCreateRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCSymCreateRibalog; far;

VCSymPlaceRibalog

Version 1.2

Declaration

C/C++ extern "C" void WINAPI VCSymPlaceRibalog();

Visual Basic Declare Sub VCSymPlaceRibalog Lib “VCDLG32.DLL” ()

Delphi procedure VCSymPlaceRibalog; far;

VCTextLineRibalog

Version 1.2

Declaration

C/C++	extern "C" void WINAPI VCTextLineRibalog();
Visual Basic	Declare Sub VCTextLineRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCTextLineRibalog; far;

VCTextSettingsRibalog

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCTextSettingsRibalog();
Visual Basic	Declare Sub VCTextSettingsRibalog Lib “VCDLG32.DLL” ()
Delphi	procedure VCTextSettingsRibalog; far;

VCUpdateDialog

Version	1.2
Declaration	
C/C++	extern "C" void WINAPI VCUpdateDialog(short* iError);
Visual Basic	Declare Sub VCUpdateDialog Lib “VCDLG32.DLL” (iError As Integer)
Delphi	procedure VCUpdateDialog(var iError: Integer); far;

Common Development Tasks

This chapter details a step by step instruction for common tasks in the Application Programming Interface. Most of the examples provided can be included directly into an application. However, by following the simplified steps and making appropriate modifications, virtually any application can be achieved.

Chapter Conventions

CAPS	Declared variables and parameter return values.
Italic	Value to be set.
Text	Example code.

Call Declarations

The Corel Visual CADD API contains four basic parts in the declaration: the Visual CADD API Name, the Library Location, the Parameter List, and the Return Value. The following routine will be used as an example for description:

Declare Function VCGetCurEntAtbRecCount Lib "VCMAIN32.DLL" (iError As Integer, ByVal iWhichAtb As Integer) As Integer

Note: The 16 bit versions of these DLL's don't have the "32" in the DLL name.

Corel Visual CADD API Name: The Corel Visual CADD API has been simplified by providing descriptive names for each of the routines. For example, VCGetCurEntAtbRecCount() indicates how many attributes are attached to a symbol. Other calls such as VCSetCurrentErased() erase the current entity from a drawing.

Library Location: The declarations for the Corel Visual CADD API are contained in a set of four library files called **VCMAIN32**, **VCTRANS32**, **VCTOOL32** and **VCDIALOG32**. The names of these files correspond directly to the DLL in which the routine itself is stored. Since all of these declarations are available for direct inclusion into your application, the library locations are rarely a concern to the programmer, but are provided in case you wish to include a minimal set of declarations in your application:

- **VCMAIN32** contains the majority of the database routines, such as entity creation and system settings, and is a more-or-less a general purpose library.
- **VCTRANS32** contains all the file reading and writing (translation) routines. For example, a call to load an AutoCAD 3D file is represented in this library.
- **VCTOOL32** contains tool commands that are available directly through the Corel Visual CADD interface, such as 2-point lines and circles.
- **VCDIALOG32** contains all of the built in dialogs that show up while working in Corel Visual CADD, such as the Layer Manager and the Symbol Manager.

Parameter List: When working with the Corel Visual CADD API, it is necessary to pass information to Visual CADD about the specific information you want to set or have returned. This is reflected in the parameter list for each routine. Different routines will require different parameters. For example, in a sample declaration such as VCGetCurEntAtbRecCount, you must specify the attribute index in order to retrieve the record count.

Return Value: The return value is the end result for the routine if it is declared as a function. For example, a sample declaration like VCGetLineTypeIndex() returns the current line type property index number.. Other routines may return information that is related in some way to the parameter information being passed by the function. For example, the name of a drawing is passed back as a parameter with the VCGetDrawingName() routine, and its return value is the number of characters in that name. Remember that procedures (sometimes called subroutines) do not have a return value.

The one common ground for most of these routines (both functions and procedures) is the **iError** value. This value represents the success or failure of the function. Some calls to set properties will only return an iError value since no information is needed on return. An iError value of 0 is true or succeed, while all other values other than 0 is failed or false.

Parameter Details

Most of the functions listed utilize a specific set of parameters which are needed by the routine in order to return the information requested. Please see the specific call for more information on the required parameters. The following parameters are discussed in more detail and apply to all of the Corel Visual CADD API routines in one way or another: iError, distances, angles, toggles, strings, user data, and special types.

iError - This is set depending on the success or failure of the function.
0 - Succeeded.
1 - Failed: Usually due to an invalid drawing world. Please see the specific routine for more detailed information.

distance - All distances are stored in the Corel Visual CADD database in inches. When retrieving or setting distance values, you need to convert them into the proper units. VCGetUnitConversionFactor() returns a multiplier that can be used to convert the values based on the current unit setting in Corel Visual CADD.

angles- All angles are stored in radians in the Corel Visual CADD drawing database. When retrieving or setting angle values, you need to convert to the appropriate display format, typically degrees.

toggle - Most of the Get/Set calls simply return a toggle state for the specified setting. The values returned are 1, indicating “on,” “checked” or “true," and 0, indicating “off,” “unchecked” or “false.”

string - Calls to retrieve a string value also return the length of the string. Visual Basic requires fixed length strings for return values. These can then be trimmed to the returned string length. In some languages, a “Null” value can be passed into the routine in place of the string variable, allowing the call to only return the string length. The string variable can then be allocated before call the function again.

User Data - Attaches or retrieves data of the specified type for the current entity. User data may be attached to any drawing entity or a drawing header and used for storage of entity information, drawing information, custom settings, or indices to external tables. User data can be of the variable types double, float, long, short or byte. In addition to these types, a user defined type of “chunk” may also be stored. A chunk can be any size and is simply a pointer to a memory location. The size of the chunk is also passed to Corel Visual CADD so that it can retrieve the appropriate amount of data from the specified memory location.

Special Types **-** There are various special cases for calls which return either a double or a user-defined variable type. Visual BASIC and Delphi do not allow user defined types to be passed by value, therefore they can not call these routines. The solution is to utilize the “BP” routine which operates the same as the original routine, but accepts the user-defined data type passed by pointer (or reference).

Adding Entities to the Database

The API provides several methods for adding entities to the drawing database. These methods include tool operations and direct code. This section focuses on methods used to add entities directly through code. Any user input and interaction in these situations are handled by the application. The tool commands allow an application to launch the drawing tool while the Corel Visual CADD handles all the user input. Whether adding a single point or a set of complex Bezier curves, the new entity is always appended to the end of the database. This allows direct access to the newly added entity.

All the commands to add entities to the drawing require the entry of points to define the geometry. These points are passed in a packed coordinate pair consisting of an x and y double value(x, y, z for 3D entities). These types are defined by the following samples. All type declarations are found in Appendix C.

```
Type POINT2D
   x As Double
   y As Double
End Type
```

```
Type POINT3D
   x As Double
   y As Double
   z As Double
End Type
```

These coordinates can be altered after the entity has been added to modify the geometry based on a changing set of criteria. The input coordinates can come directly from the code or as a results of user interaction within the interface. See the user tool section of this task guide for details on capturing user events within the Corel Visual CADD interface.

In addition to adding entities directly to the drawing database, the commands can be used to create complex symbol definitions and hatch boundaries. The routines require the entry of an iSymbolIndex describing the definition that is to be added to the entity. The parameter values are described for each routine of the calls and correspond to the following rule. Use a -1 to add the entities directly to the drawing database, -2 to create a hatch\fill boundary object and $\geq$ 0 for creating a symbol index. A new symbol definition is created by calling VCCreateSymbolDef which returns an index value to be used as the iSymbolIndex for the add routines. To modify an existing symbol the iSymbolIndex can be found with VCGetSymbolIndex by passing the internal name of the symbol. When used as the iSymbolIndex argument for the add commands, the entity will be added to the symbol instead of the drawing. See the Symbols section of this task guide for details on building symbol definitions.

Adding a Single Entity

Most of the Corel Visual CADD entities are considered single. That is the entity requires a specific number of placement points to be passed through the API. Each of these input points are then used to define specific aspects of the entity. For example, a circle requires a center point and a radius point. The routine VCAddCircleEntity takes an input parameter for each of these points. The following steps show the method for adding and displaying an entity through code. The A - Z Reference should be consulted to determine the proper number of input points required by each entity.

Steps:

1) Optional: Set properties for the new entity.

While all the properties can be modified after the entity has been added to the database, it is convenient to set the properties prior to the operation. The properties for entities typically include color, layer, line type and line width. These can be set with the API calls VCSetColorIndex, VCSetLayerIndex, VCSetLineTypeIndex and VCSetLineWidthIndex, respectively. Complex entities such as dimensions, text, fills and hatches require specific setting routines for the entity type. See the A-Z Reference for detail on these specific routines.

The following code sets the color property to *blue*, the width index to *3* and the layer index to 30:

```
Call VCSetColorIndex(IERROR, 9)
Call VCSetLineWidthIndex(IERROR, 3)
Call VCSetLayerIndex(IERROR, 30)
```

2) Add the entity to the database.

To add the entity to the drawing simply specify the point location which can be determined either directly through code or as a results of user input. The point entries are then passed to the add command to place the entity. Note each entity requires a different number of input points to define the geometry. For example, a line entity requires two point locations for the end points while an elliptical arc requires seven points to define the placement. Verify the number of entries and the placement location in the A-Z Reference.

The following code adds a line from *0,0* to *10,10* based on values established through code. Note the points have been defined as a Point2D and contain a x and y value.:

```
ENDPOINT1.X = 0
ENDPOINT1.Y = 0
ENDPOINT2.X = 10
ENDPOINT2.Y = 10

Call VCAddLineEntityBP(IERROR, -1, ENDPOINT1, ENDPOINT2)
```

3) Optional: Move to the new entity in the database.

The entity is always appended to the end of the database and given an entity handle one greater than the previous last entity. It may be necessary to move to the new entity for tracking ID, checking properties or adding data to the new entity. Each entity in the drawing database is referenced based on an entity handle or ID allowing applications to quickly access entities with VCSetCurrentEntity.

The following code moves to the newly placed entity and sets it as the current entity:

```
Call VCLastEntity(IERROR, ENTITYHANDLE)
Call VCSetCurrentEntity(IERROR, ENTITYHANDLE)
```

4) Optional: Draw the new entity to the screen.

New entities are not automatically displayed on the screen after being added to the database. In order to draw the entity immediately, use VCDrawCurrentEntityVCDrawCurrentEntity. The entity will be drawn during the next redraw or paint event.

The following code draws the new entity immediately to the screen:

```
Call VCDrawCurrentEntity(IERROR)
```

Adding a Continuous Entity

The Corel Visual CADD engine supports both continuous lines and beziers. These entities allow for a virtually unlimited number of points to define the geometry. Due to these varying point counts, a reference to the entity must be added an then construction points are used to define the placement of the new definition. The following steps demonstrate the sequence required for adding these special entities.

Steps:

1) Optional: Set the properties for the new entity.
Please see Adding a Single Entity for setting properties prior to adding to the drawing database.

2) Add a reference to the entity to the database.
Continuous entities allow for a multiple number of points defining the geometry. These points must be set after the entity has been referenced in the database. Like the single entities, the defining points can be defined through code or through interaction within the interface.

The following code adds a reference to a *Continuous Bezierr* entity:

Call VCAddContinuousBezierEntity(IERROR, -1)

3) Move to the new entity in the database.
In order to add placement points to the geometry, the new entity must be set current. This ensures the point operations are applied to the correct entity. Each entity in the drawing database is referenced based on a handle or ID allowing applications to quickly access entities with VCSetCurrentEntity.
The following code moves to the newly placed entity and sets it as the current entity:

Call VCLastEntity(IERROR, ENTITYHANDLE)
Call VCSetCurrentEntity(IERROR, ENTITYHANDLE)

4) Set the points for the continuous entity.
The points defining the geometry should be passed through an array. Since the entity allows for an infinite number of points within the array, the application must indicate the number of points being added. This is done through the VCSetCurrentEntityPoint command which allows point entry from an array along with the number of points defined within the array.

The following code sample adds *10* random points to the *Continuous Bezier* entity:

```
For IINDEX = 1 To 10
        POINTARRAY(IINDEX).X = IINDEX * Rnd(5000) * 100
        POINTARRAY(IINDEX).Y = IINDEX * Rnd(5000) * 100
        Call VCSetCurrentEntityPoint(IERROR, 10, POINTARRAY(IINDEX))
Next IINDEX
```

5) Optional: Draw the new entity to the screen
Call VCDrawCurrentEntity(IERROR)

Adding a Hatch/Fill Entity

Hatch and fill entities behave very similar to the continuous entities in that a definition must be placed in the database and then entities and points used to define the definition. The difference is that a hatch entity boundary can be defined by any of the other geometry entities. For example, a circle and a rectangle can be used to define the boundaries of the pattern. Therefore, when adding hatch entity, the application must add a reference to the hatch and then add single or continuous entities to define the boundary. The defined boundary must be a closed element in order to complete the process. The following steps illustrate this concept in detail.

Steps:

1) Optional: Set the properties for the new entity.
Hatch settings differ from the standard line type, color and width settings. Specific settings calls are required to alter the color, pattern rotation and scale. See the A-Z Reference for detail on these specific routines.

The following code sets the hatch pattern to *ZigZag* with a scale of *0.5*, rotation angle of *0.0* and color to *blue*:

Call VCSetHatchColor(IERROR, 9)
Call VCSetHatchRot(IERROR, 0#)
Call VCSetHatchScale(IERROR, 0.5)
Call VCSetHatchName(IERROR, “ZigZag”)

2) Add a reference to the entity to the database.
Hatch entities allow different combinations of entities to form the boundary. These entities need to be added to an existing hatch reference. After the reference is placed in the drawing file, entities are added to the reference and sorted to verify the new boundary.

The following code adds a reference to a *Hatch* entity:

Call VCAddHatchEntity(IERROR, -1)

Move to the new entity in the database.
Call VCLastEntity(IERROR, ENTITYHANDLE)
Call VCSetCurrentEntity(IERROR, ENTITYHANDLE)

3) Add entities to create the boundary.
The added entries must create a closed boundary item for the hatch. These boundaries can be made of either single or continuous entities.

The following code creates a hatch boundary based on a circle entity. Note the -2 for iSymbolIndex in order to add the entities to the hatch definition.

CENTERPOINT.X = 0
CENTERPOINT.Y = 0
RADIUSPOINT.X = 0
RADIUSPOINT.Y = 10

Call VCAddCircleEntityBP(IERROR, -2, CENTERPOINT, RADIUSPOINT)

4) Optional: Use a continuous entity.
The added entries must create a closed boundary item for the hatch. These boundaries can be made of either single or continuous entities. Please see Adding a Continuous Entity for more information.

5) Sort the Hatch/Fill Entity.
After the boundary has been created, the Corel Visual CADD engine must parse the boundary and determine if it is closed. Only closed boundaries can be hatched or filled.

The following code hatches the boundaries if they are closed:

Call VCSortCurrentHatchFillEntity(IERROR)

6) Optional: Draw the new entity to the screen
Call VCDrawCurrentEntity(IERROR)

Adding a Text Entity

Text entities are a special case of single entities that simply allow for a single input placement point and require a string to be set prior to adding the entity to the database. The following steps illustrate this concept.

Steps:

1) Optional: Set the properties for the new entity.
Text settings differ from the standard color, line width , layer and line type. The text settings are supported through individual routines for each setting required. The main property settings for a text entity is the font which can be a True Type or Vector style. Display parameters are available depending on the particular font being used. For example, Vector fonts utilize an Italic value to specify the exact slant angle required while True Types only allow an italic flag to be set but not the actual slant angle. These settings are detailed in the A-Z Reference and should be noted when working with text entities.

The following code sets the text layer to *8* and the text color to *9*.

Call VCSetTextLayer(IERROR, 8)
Call VCSetTextColor(IERROR, 9)

2) Set the string to add.
Typically, when adding text directly through code, user interaction and input is not desired or available. In these cases, when the Corel Visual CADD interface is bypassed, the text string needs to be set directly from the external application.

The following code sample sets the text string *"This is a sample text string"* for adding to the drawing:

Call VCSetTextString(IERROR, "This is a sample text string")

3) Add the entity to the database.
Text entities only require an insertion point for the lower left corner of the string.

The following code adds a text entity to the drawing at *0,0*:

INSERTIONPOINT.X = 0
INSERTIONPOINT.Y = 0

Call VCAddTextEntityBP(IERROR, -1, INSERTIONPOINT)

4) Optional: Move to the new entity in the database.
Call VCLastEntity(IERROR, ENTITYHANDLE)
Call VCSetCurrentEntity(IERROR, ENTITYHANDLE)

5) Optional: Draw the new entity to the screen.
Call VCDrawCurrentEntity(IERROR)

Adding a Reference Frame Entity

Reference Frame entities enable you to display the contents of one file within another. You can use the frames to layout drawings for printing or to create overlays. Typically this feature is implemented to provide Paper Space/Model Space capability in which a real world scale drawing is referenced into a paper space title block. The reference frame entity is then set to a specific scale for output. The printed or plotted output is then based on a paper size drawing which directly reflects the desired output.

Step:

1) Set the file name to add as the Reference Frame
The reference frame contains a drawing that is to be referenced inside active drawing. The path and filename for the drawing is required to define the entity.

The following code references a drawing named SAMPLE.VCD from the active directory:

Call VCSetRefFrameName(“SAMPLE.VCD”)

2) Optional: Set the desired frame properties.
A reference frame has specific properties that apply only to this entity type. These include boundary color, line width, display, offset, rotation and scale. Each of these properties can be altered after the entity has been placed however it is convenient to set these prior to adding the entity.

The following code sets the boundary display off and the rotation to 45 deg.:

Call VCSetRefFrameDrawBoundary(IERROR, 0)

Call VCSetRefFrameRot(IERROR, PI/4)

NOTE: All angles must be entered in radians. PI/4 represents 45 deg. In radians.

3) Bind the data if desired.
The referenced drawing can either be bound or linked to the reference frame entity. Bound data causes the reference frame to store the vector information directly in the active drawing while linked data stores only a reference to the external file. Linked drawings will reflect changes made after the file has been added. Bound data will cause the drawing size to enlarge since all the reference file information is not stored in the new drawing.

The following code specifies for the reference frame to contain linked data:

Call VCSetRefFrameIsDataBound(IERROR, 1)

4) Optional: Set the reference frame size.
A reference frame defaults to the size of the file that is being referenced. In situations where several reference frames are used to reference a drawing, it may be desirable to specify the frame size in order to have the referenced drawing fit to a certain position.

The following code sets the frame height and width to one unit each:

```
POINT2D DPHEIGHTWIDTH

DPHEIGHTWIDTH.X = 1
DPHEIGHTWIDTH.Y = 1
```

Call VCSetRefFrameViewWidthHeight(IERROR, DPHEIGHTWIDTH)

NOTE: The X value of the Point2D represents the height while the Y value represents the width.

5) Add the reference frame.
The reference frame entity can now be added to the drawing database.

The following codes adds the reference frame entity to the drawing at 0,0:

Call VCAddRefFrameEntity(IERROR, -1, DPPOINT)

Symbol Operations

Symbols are a collection of entities that have been grouped under a single definition. This allows the grouped entities to be placed multiple times within a drawing. In addition, symbols behave as a “primitive” entities and can be manipulated the same as lines, arcs and circles. Typically, symbols are used to reflect building parts such as a door, sink, or window.

Instead of maintaining a large set of symbols reflecting all the possible property combinations, a symbol definitions can be parametrically generated directly through the API. For example, a door can be created from a set of property inputs from the application interface or a set of database rules. When created through code, the symbols handle, or placement point, is defined by the drawing origin. Therefore, when creating a symbol it is necessary to build the part around the origin or transpose the entities when finished. If not, inconsistencies in the placement point can occur. For example, a symbol is generated about the coordinate pair 10,10 instead of the origin. The code then requires the new symbol entity to be placed in the drawing at the coordinate pair 50,50. Since the symbol definition was created in error, the code would shift the symbol to 60,60 instead of the desired placement point.

Symbol definitions are defined by an internal name and a set of entities grouped under that name. The internal name can consist of thirty-two characters to define the symbol within the Corel Visual CADD interface. Symbols can also be saved to disk for use in other drawings. When working with symbol definitions, the index or internal name is required. However, since the saved disk name is typically known and not the internal name, use VCGetSymbolInternalName to determine the internal name.

Loading a Symbol

In order to work with a symbol inside a drawing sessions, the symbol must be loaded into memory. An application or user can then use tools to manipulate and place the symbol throughout the drawing. The following code demonstrates the process for loading a symbol directly through code.

Steps:

1) Retrieve the internal name for the symbol.

When working with symbol entities, the index or the internal name are required. Since the index changes as new symbols are loaded and others are deleted, typically the internal name should be used. As noted, the internal name may not be known but can be found from the one saved on disk.

The following code retrieves the internal name for a symbol from the one saved on disk. Since this example utilizes Visual Basic, the returned string must be trimmed:

```
iSTRINGLENGTH = VCGetSymbolInternalName(IERROR, SAVEDDISKNAME, FIXEDINTERNALNAME)
TRIMMEDINTERNALNAME = Left(SAVEDDISKNAME, iSTRINGLENGTH)
```

2) Optional: Test if the symbol is already loaded.

3) Load the symbol based on the internal name.

When working with symbol, the symbol must be loaded into memory within the drawing session prior to any placement commands. When loading new symbols into the session, an application should check if the definition has already been loaded by another application or the user. The symbol must be loaded in order to be placed or manipulated. The symbol can be loaded multiple times but will keep a single definition in memory.

The following code tests if the symbol has been loaded and loads the definition.

```
If VCIsSymbolLoaded(TRIMMEDINTERNALNAME) = 0 Then
 Call VCOpenVCS(SAVEDDISKNAME)
End If
```

1) Optional: Follow with the symbol place operation.

For more information, please see Placing a Symbol

Placing a Symbol

Symbols behave as a primitive entity in the drawing. Therefore placing them from the interface or code is the same as placing a line. Two methods for symbol placement are provided through the API. The first allows a user to specify the location while Corel Visual CADD handles all rubberbanding and placement options. The second allow an application to fully control the placement by directly setting the placement point through code. The following example shows both of these methods to place a loaded symbol entity.

Steps:

1) Optional: Load the symbol.

2) Optional: Place the symbol directly through code.

The symbol can be added directly to the drawing database or another symbol definition directly through code. This method requires the same steps presented in the adding a single entity. Once a symbol definition has been loaded, the symbol must be selected from the internal definition buffer. Loaded symbols are available to all drawings within the sessions and are referenced by an internal name or index. In order to place the symbol, it must be the currently active symbol definition.

The following code sets the active symbol definition based on the internal name and places it at the origin *(0,0):*

PLACEMENTPOINT.X = 0

PLACEMENTPOINT.Y = 0

Call VCSetSymName(IERROR, SYMBOLINTERNALNAME)
Call VCAddSymbolEntityBP(IERROR, -1, PLACEMENTPOINT)

3) Optional: Place the symbol using a tool command.

The API provides several options when placing a symbol through code. If the application requires user input for the placement point then VCSymboPlace can be used. This routine initiate a tool in the drawing interface and immediately allows the user to begin dragging the symbol to the desired location. Corel Visual CADD handles all rubberbanding and preview events in this case.

The following code initiates the symbol placement tool and allows the user to select the placement point:

Call VCSymbolPlace(SYMBOLINTERANLNAME)

Creating a Symbol

As mentioned, symbols can be generated directly from code. The process is similar to adding entities to the drawing with one minor change. A symbol definition must be created with VCCreateSymbolDef. After the definition has been created, any of the previous operation, can be used by simply substituting the proper symbol index. The following code demonstrate this process in detail.

Steps:

1) Create a symbol name.

Create a reference to the symbol definition. The definition allows the code to add a symbol to currently active drawing session based on an internal name.

The following code create a new definition with the name "*NewSymbolDefintion*":

ISYMBOLINDEX = VCCreateSymbolDef(IERROR, "NewSymbolDefintion")

2) Add the entities directly to the symbol definition.

The returned symbol index can then be used to add entities directly to the definition. Use the methods discussed in the "Adding Entities to the Database" section.

The following code adds entities directly to the new symbol definition.

```
CENTERPOINT.X = 0
CENTERPOINT.Y = 0
RADIUSPOINT.X = 0
RADIUSPOINT.Y = 10

Call VCAddCircleEntityBP(IERROR, ISYMBOLINDEX, CENTERPOINT, RADIUSPOINT)
Call VCAddLineEntityBP(IERROR, ISYMBOLINDEX, CENTERPOINT, RADIUSPOINT)
```

3) Optional: Save the symbol definition to disk.

When a symbol definition is created it is simply loaded into the current drawing session. In order to utilize the definition in other drawing sessions, it must be saved to disk.

The following code saves the symbol to a *.VCS file in the path *"c:\vcadd\symbols\savevcs.vcs":*

```
Call VCSaveVCS("NewSymbolDefinition", "c:\vcadd\symbols\savevcs.vcs")
```

Modifying a Symbol Definition

Modifying an existing symbol entity requires the same steps as used in the creation process. In this case an application determines the symbol index and then any of the entity operation can be used to add to the definition. The following code demonstrate the process:

Steps:

1) Retrieve the symbol index.

In order to add entities directly to a symbol definition, the symbol index must be used. Since the symbol index can change as new symbols are loaded and others deleted, it should be determined from the internal name.

The following code retrieves the symbol internal name:

ISYMBOLINDEX = VCGetSymbolIndex(IERROR, SYMBOLNAME)

2) Add the entities directly to the symbol definition.

The returned symbol index can then be used to add entities directly to the definition.

The following code adds an arc to an existing definition from the point *0,0; 10,10; 20,0*:

```
ARCSTARTPOINT.X = 0
ARCSTARTPOINT.Y = 0
ARCMIDPOINT.X = 10
ARCMIDPOINT.Y = 10
ARCENDPOINT.X = 20
ARCENDPOINT.Y = 0
```

Call VCAddArcEntityBP(IERROR, ISYMBOLINDEX, ARCSTARTPOINT, ARCMIDPOINT, ARCENDPOINT)

Parsing a Symbol Definition

Parsing is a method for moving from one entity to the next within the drawing database. A full detail of the steps are provided later in these guide. Typically, an application may parse a symbol definition to change the entity properties that make up the symbol or remove an existing entity from the symbol. The following code demonstrates the steps for parsing the definition of a symbol. Please refer to the Database Operation section later in the guide for complete details and steps for parsing the drawing database.

Steps:

1) Set the symbol section for parsing.

When querying information about entities, it is necessary to parse the database for the desired item and set it as the active handle. Symbols behave as “primitive” entities that are present within the drawing database. In order to access information about these subentities, an application needs to set the parsing selection to the symbol definition prior to using the database routines. This allows an application to retrieve and set values pertaining to entities within a symbol definition.

The following code sets the parsing section for a symbol name “*InternalSymbolName*”:

```
Call VCSetSymbolSection(IERROR, “InternalSymbolName”)
```

2) Use the database parsing routines.

When querying information from a symbol definition, it is necessary to parse within the definition to retrieve the properties. Any of the parsing methods provided through the API may be used.

The following code parse the symbol entity and return a count for the number of entities in that symbol:

```
If VCFirstEntity(IERROR, ENTITYTYPE) Then
  Do While IERROR = 0
    LCOUNT = LCOUNT + 1
    If VCNextEntity(IERROR, ENTITYTYPE) Then
    End If
  Loop
End If
MsgBox "Total Symbol Entity Count: " & CStr(LCOUNT), 64, "Sample Files"
```

3) Reset the parsing routines to the entity section.

After completion of the symbol parsing routine, reset the entity section to the drawing.

The following code resets the parsing section to the drawing:

```
Call VCSetEntitySection(IERROR)
```

Retrieving a Symbol Count

Typically an application may build a Bill of Materials based on entities in the drawing database. Symbols are used to represent specific objects to track such a bolt or nut type. A Bill of Material would then need to track the number of bolts placed in the drawing. The API provides a fast method for retrieving this information and from the drawing.

Steps

1) Retrieve the symbol definition count.
Symbol definitions are loaded into the drawing session as new drawings are load or by specific commands to load them.

The following code retrieve a count for the number of currently loaded symbol definitions:

```
ISYMDEFCOUNT = VCGetSymbolDefCount()
```

2) Parse the symbol definitions and retrieve the placement count.

After retrieving the number of definitions loaded in the session, an application can then simply move through the list and get the necessary information.

The following code parses the symbol definition and retrieves the number of placements for each definition.

```
For IINDEX = 0 To ISYMDEFCOUNT
        IPLACEMENTS = VCGetSymbolPlacementCount(IINDEX)
Next IINDEX
```

Utilizing a Custom Interface

Through the API an application can create a custom interface. This interface only the specific tools that you want your application to have. When creating an interface using only the Corel Visual CADD DLL's, the engine should be initialized and a drawing world set. The rest of the interface is under complete control from the developer. For example, an application may only need to provide viewing capabilities. In this situation, the application may simply create a drawing screen and limit the tool set to the zoom functions, thus limiting the user from making any changes to the drawing.

There are several steps in the process for creating a custom interface. These steps should followed in order to create the interface desired for the application.

Steps:

1) Initialize the engine in the applications startup routine.
In order to use the Corel Visual CADD engine from a custom interface, the application must initialize the DLL's if Visual CADD isn't already running. Once the engine has been initialized, the full power of Corel Visual CADD is available.
The following code initializes the Corel Visual CADD graphics engine:

Call VCInit

2) Optional: Initialize dialogs in the applications startup routine.
It may be necessary to provide some of the interface functionality from Corel Visual CADD directly into an application. For example, an application may need the print functionality built into Corel Visual CADD. Instead of recreating the print dialog, an application can simply use the built in dialog directly in the custom interface. Corel Visual CADD will handle all the print output in this situation with no modification from the developer.

The following code can be used to initialize the dialogs from the Corel Visual CADD interface:

Call VCInitDialogs

3) Create a drawing world from a control hWnd.
In order to load a drawing , a valid drawing world must be specified. This drawing world does not have to be visible and can simply be used as a memory holder for the loaded drawing. The drawing world is created through a Windows hWnd. The drawing screen can be any control that has the ability to display information. Corel Visual CADD references the drawing with a world handle index. Since Corel Visual CADD has a multiple document interface, these handles should be used to reflect both the active drawing and drawing area. As an application creates new worlds, the handles should be stored for maintaining valid drawings.

The following code creates a world and sets it as the active drawing:

```
NEWWORLD = VCNewWorld(FORM.HWND)
Call VCSetCurrWorld(NEWWORLD)
```

4) Optional: Provide a mouse down event handler in the control.
The application should provide event handlers for all the Windows events it wants to receive. For example, a mouse down event is required if the application allows the user to interact in the drawing with drawing tools. A simple file translator however has no need to provide event handling within the drawing world.

The following code should be placed in the mouse down event for the application in order to send messages to the Corel Visual CADD engine:

Call VCLButtonDown2(x, y)

5) Optional: Provide mouse move event handler in control.
The application should provide event handlers for all the Windows events it wants to receive. For example, a mouse down event is required if the application allows the user to interact with the drawing and drawing tools. A simple file translator however has no need to provide event handling within the drawing world. Mouse events should be tracked to allow tool operations to utilize preview and rubberband events.

The following code should be placed in the mouse move event for the application in order to send messages to the Corel Visual CADD engine:

Call VCMouseMove2(X, Y)

6) Optional: Set message handle from a control hWnd.
The Corel Visual CADD interface uses several displays to enhance the users understanding of the drawing process. These include message or prompt handles, coordinate display and ribalogs. Each of these displays item can be included directly into your custom interface with little programming effort by providing the hWnd for the appropriate control. An edit box or static label are examples.

The following code set a message handler within the custom interface:
Call VCSetMessageHandle(IERROR, HWND)

7) Optional: Set an X,Y coordinate display.
The Corel Visual CADD interface uses several displays to enhance the users understanding of the drawing process. These include message or prompt handles, coordinate display and ribalogs. Each of these display item can be included directly into your custom interface with little programming effort.

The following code sets a coordinate display handler within the custom interface:
Call VCSetXYHandle(IERROR, HWND)

8) Optional: Set a ribalog handle from a controls hWnd.
The Corel Visual CADD interface uses several displays to enhance the users understanding of the drawing process. These include message or prompt handles, coordinate display, and ribalogs. Each of these display item can be included directly into your custom interface with little programming effort. The context sensitive ribalogs give the user settings during a tool operation. If these are not made available to the user then the code should handle all the settings prior to activation of the tool. For example, the VCCopySelected command initiates the copy tool from the interface. In Corel Visual CADD, a ribalog displays a setting for the number of copies desired. If the ribalog is not made available to the user then the code should set the number of copies prior to activating the tool.

The following code sets the ribalog handle for displaying the ribbings:
Call VCSetDialogFrameHwnd(HWND)

9) Optional: Provide the tools for the interface.
The interface can provide any tool desired. The easiest method for applying the tools is utilize any of the tool commands provided in the API. The custom interface can however utilize user tools built for specific tasks.

The following code provides the line tool on a command button:
Call VCLine

10) Destroy the engine in the applications close routine when finished.
Upon close, the application should free up memory by disabling the Corel Visual CADD engine.
Call VCTerminate

11) Optional: Destroy the dialogs in the applications close routine.
Close the dialogs if they were used in the custom interface.
Call VCTerminateDialogs

Creating a User Tool

User tools are powerful additions to the standard tool set provided directly through the Corel Visual CADD interface. User tools offer complete control over all Window events and operations. The advantage generated by users tools stems from this control over the messaging events. By capturing these events, the application can create custom event driven tools for their application. Any messaging not retrieved by the user tool is automatically handled by Corel Visual CADD.

There are several steps involved in creating a user defined tool. The definition for routines displayed here are contained in the Alphabetical Listing of Functions and Subroutines. In addition, some variable declarations have been omitted.

Steps:

1) Set the level of event messaging in the initial application routine.
An application can register in any or all of the Corel Visual CADD messaging loops. However, in certain situations it may only be necessary to retrieve mouse events. These event registers are passed as a parameter to VCSetAlertApp. See the A-Z Reference for details on the values.

The following code registers an application into the Corel Visual CADD messaging loop to retrieve all events:

Call VCSetAlertApp(IERROR, FORM.HWND, 0)

2) Set the begin level for an undo sequence.
Corel Visual CADD allows unlimited levels of undo and redo operations. In order for an external application to take advantage of these levels, it should set a begin and end level for an undo operation. This allows the user to undo or redo the operation after the original action.

The following code sets the initial begin operation level:

Call VCBeginOperation(IERROR)

3) Set the user tool in the applications initial routine.
After the application has been registered into the messaging loop, the tool itself must be created in order for Corel Visual CADD to interpret the operation. This process tells Corel Visual CADD how many steps are required for the tool completion, the prompts a user follows while activating the tool and the native command name for launching the specified tool.

The following code sets the initialization routine for the user tool with an unlimited number of entry points(-1) with a tool name of "*Select Example*" and an initial prompt "*Pick an entity*":

Call VCSetUserTool(-1, "Select Example", "Pick an entity")

4) Optional: Set the prompts for each event returned.
If your tool requires multiple events, it is necessary to provide prompts for each action the user must provide while working within the tool.

The following code sets the second and third prompt to reflect the selection list tool:

Call VCSetPrompt(2, "Pick entity # 2")
Call VCSetPrompt(3, "Pick entity # 3")

5) Optional: Set the cursor type required by the application.
Depending on the type of tool being creating a different cursor may be desired. For example, this selection routine should probably utilize a special selection cursor to let the user know it is in a selection mode. A draw tool should probably utilize the standard cross hair cursor the user is accustomed to seeing within the Corel Visual CADD interface. See the A-Z Reference for cursor types.

The following code specifies the cursor to be changed to a selection arrow during the tool operation:

Call VCSetCursor(IERROR, "IDC_NORMAL")

6) Optional: Retrieve the mouse down events.
Depending on the events your application is register(see step 1), the tool can respond to the event in any manner desired.

The flowing code takes the mouse down event and selects the closest object to the mouse down:

Call VCUserToolLBDown(IERROR, DOWNPOINT)
Call VCObjectSelect(DOWNPOINT)

7) Optional: Retrieve the mouse move events.
This example does not require the mouse move event. Typically, the mouse move event would be used to create rubberbanding as the user works with the tool.

The following code is given for reference only and is not required by this particular tool:

```
Call VCUserToolMouseMove(IERROR, MOVEPOINT)
```

8) Optional: Retrieve the key press events.
The user tool should generally provide an escape or abort event while within the tool operation. This typically is done through a keypress event for the <ESC> key. The application can also retrieve general keypress events for processing internal routines. For example, an application could capture a required angle input from the command line or offer multiple solutions based on the input key.

The following code provides an escape event for the user during the tool operation. VCAbortOperation ends the undo levels and VCAppExit cleans up the messaging loop:

```
If (KEYASII = 256) Then
        Call VCAbortOperation
        Call VCAppExit
End If
```

9) Set the end of the undo sequence.
Corel Visual CADD allows unlimited levels for undo and redo. In order for an external application to take advantage of these levels, it should set a begin and end level for an undo operation. This allows the user to undo or redo the operation after the original action.

The following code sets the end operation level:

```
Call VCEndOperation(IERROR)
```

10) Optional: Reset the cursor.
If the cursor was changed for the tool operation, then it should be reset to the default cursor in the Corel Visual CADD interface. This is done by passing a NULL string the routine.

The following code resets the cursor type to the default:

```
Dim NULLSTRING As String
Call VCSetCursor(IERROR, NULLSTRING)
```

11) Clear the application from the messaging loop.
After the application has completed, the tool should be cleared from the messaging loop.

The following code clears the application for the messaging loop:

```
Call VCClearAlertApp(IERROR, FORM.HWND)
```

12) Clean up the application on close.
In order to free residual effects of the tool, the application should utilize VCAppExit on the closing event.

The following code ends the application and clears the tool from memory:

```
Call VCAppExit
```

Using the Corel Visual CADD Interface

The Corel Visual CADD interface makes extensive use of built in speed bars. Your application can access these same ribalogs as needed. However, in most cases, you will need to create your own input form for retrieving settings and properties for your application. Corel Visual CADD allows you to create a custom interface and utilize the speedbar area directly in Corel Visual CADD. This method has numerous advantages over a normal dialog. The user already has an understanding of the Corel Visual CADD interface and its functionality. The following steps can be utilized to create a custom speed bar for interaction in Corel Visual CADD.

Steps:

1) Retrieve the ribalog size and coordinates.

The API will return the current screen location for the bounding rectangle in the main speedbar. An application can then resize dialogs based on this information. The values are returned in an iPoint2D structure containing the x , y pixel screen coordinates for the bar. Screen coordinates in Windows are referenced from the upper left corner and increase in size as moved right and down. In some languages screen coordinates are referenced in twips instead of the standard pixels. In these cases, a conversion should be used based on values from the language.

The following code retrieves the screen coordinates for the ribalog:

Call VCGetRibalogSize(IERROR, RIBALOGORG, RIBALOGSIZE)

2) Retrieve the status bar coordinates.

The API will also return the coordinates of the status bar. The application can use this information to display custom information during the operation of a tool. However, note that an application has full control over the status prompt of the Corel Visual CADD interface. In cases where only prompt information needs to be displayed use VCSetPrompt. However if the application needs to provide direct access to custom status information then displaying a status form can be used.

The following code retrieves the screen coordinates for the status bar:

Call VCGetStatusBarSize(IERROR, STATUSORG, STATUSSIZE)

3) Resize the application form based on the coordinates.

The form should be resized based on the values returned in the previous examples. The process for doing this is language dependent. Consult the language guide for your compiler for details on sizing a form at run time.

The following code presents the straight forward method of resizing a Visual Basic form based on these values:

```
GETTWIPSX = Screen.TwipsPerPixelX
GETTWIPSY = Screen.TwipsPerPixelY
FORM.Top = (RIBALOGORG.Y + 1) * GETTWIPSX
FORM.Left = RIBALOGORG.X * GETTWIPSY
FORM.Width = RIBALOGSIZE.X * GETTWIPSX
FORM.Height = (RIBALOGSIZE.Y - 2) * GETTWIPSY
```

Attribute Manipulation

Attributes are non-graphical data attached to symbol entities in the drawing database. Using attributes to represent Bill of Materials and other information is being replaced by User Data methods. However, in many situations there are existing CAD drawings containing attribute information .

Retrieving Attributes

The process for retrieving attributes runs on three levels. These levels are based on the complexity of the attached attribute. A symbol can contain any number of attributes, each of which can contain any number of labels and values. Therefore, when moving through the drawing database to retrieve attributes, an application should first filter the symbols and then move through each attribute and each label one at a time until all have been retrieved. The process can then continue on to the next symbol in the drawing and repeat the attribute search.

Step:

1) Retrieve the number of attribute definitions attached.
A symbol can have an unlimited number of attributes attached . These attribute definitions contain labels and values defining the definition. This step should be used to determine the number so a proper loop can be set up to parse through all the attributes.

The following code retrieves the number of attributes and sets up a loop structure to begin parsing the attributes.

```
CURRENTATBCOUNT = VCGetCurEntAtbCount(IERROR)
If (CURRENTATBCOUNT <> 0) Then
For ATBINDEX = 0 To CURRENTATBCOUNT - 1
```

2) Retrieve the number of labels in the current attribute definition.
Similar to the step above but this sets up a loop structure for parsing the current attribute.

The following example sets up a loop structure to parse the current attribute definition.

```
CURRENTATBRECCOUNT = VCGetCurEntAtbRecCount(IERROR, ATBINDEX)
For RECORDINDEX = 0 To CURRENTATBRACCOUNT - 1
```

3) Retrieve the label from the definition.
Once inside the attribute definition, the program can then retrieve the label and value. The label represents the name or tag for the attribute. For example, if the attribute read MANUFACTURER: COREL then MANUFACTURER represents the label and COREL represents the value.

The following code retrieve the label at the current loop index.

```
RETURNEDSTRINGLENGTH = VCGetCurEntAtbRecLabel(IERROR, ATBINDEX, RECORDINDEX, RETURNEDVALUE)
```

4) Retrieve the value from the definition.
Once inside the attribute definition, the program can then retrieve the label and value. The label represents the name or tag for the attribute. For example, if the attribute read MANUFACTURER: NUMERA then MANUFACTURER represents the label and Numera represents the value.

```
RETURNEDSTRINGLENGTH = VCGetCurEntAtbRecValue(IERROR, ATBINDEX, RECORDINDEX, RETURNEDVALUE)
```

5) Loop through the attribute definition.
The application should continue through the attribute definition until all the required data is retrieved.

6) Loop through the symbol.
If the symbol has multiple attributes attached , the application should parse through the all the attributes on that symbol before moving to the next symbol placement.

Creating Attributes

An attribute is defined by a name and contains both labels and values. The labels represent the tag for the attribute value such as “Type”. The value represents the data that is stored in with the label. For example, the previous sentence described a label of “Type”. A value for this label may be “Maple”. The value stores the specific information describing the data.

Steps:

1) Create the initial attribute values.

In order to create an attribute at least one set of label and value must be set. This data is referenced by an index of zero in the attribute definition.

The following code adds an attribute definition named “MYATB” to the current drawing session and sets the initial label to “Type” and value to “Default”:

Call VCAddAtbDef(IERROR, "MYATB", "Type", "Default")

2) Set any subsequent label and values in the attribute.

An attribute can contain up to 256 label and value sets. Each of these can represent any set of non-graphical data required. Each attribute label and value must be set in order. For example, an application can set the values for the second record set (iRec = 1) and then for the third record set (iRec = 2). It can not skip a set, it must add the items in sequential order starting with 0 for the initial set.

The following code adds four more label and value record sets:

Call VCSetAtbDefLabelValue(iErr, "MYATB", "Label1", "Value1", 1)

Call VCSetAtbDefLabelValue(iErr, "MYATB", "Label2", "Value2", 2)

Call VCSetAtbDefLabelValue(iErr, "MYATB", "Label3", "Value3", 3)

Call VCSetAtbDefLabelValue(iErr, "MYATB", "Label4", "Value4", 4)

User Data Tasks

User data is a powerful tool for attaching and accessing information to entities. This data can range from parametric entity ID values to SQL string statements for accessing a database. This data can be in many forms including all the numeric type formats and a chunk or pointer to a assignment of memory containing the desired information. In order to set aside a space of memory for attaching and accessing user data, a name must be specified for the section. This name provides a unique location for information specific to your application. Numera Software will provide a User Data ID for commercial developers to ensure unique entity definition segments.

Attaching User Data

User data can be attached to entities or the drawing header. The information attached is then stored in the drawing for later reference and use.

Steps:

1) Set the required unique user data name in the application initialization routine.

In order to set up a space of memory for each entity and the drawing header, a user data name should specified to reference the information.

The following code sets a user data name of “Sample Data”:

Call VCSetUserDataName(IERROR, “Sample Data”)

2) Attach the information to the drawing header or to the current entity.

In order to access information in the drawing header, make a call to VCSetHeaderData. Otherwise, user data is attached to the current entity. The current entity can be set in a number of ways including any parsing routine. The current entity can be accessed directly through the API.

The following code specifies that the user data is attached to the drawing header rather than a specific header:

Call VCSetHeaderData(IERROR)

The following code should be used to return the user data to the drawing entities themselves:

Call VCSetEntitySection(IERROR)

3) Add or set the desired user data values to the segment location.

The Corel Visual CADD API allows two methods for attaching the user data information. The easiest method is to simply use the add routines which add the data to the next available slot in the user data segment. The API also allows user data to be set and placed at a specific index within the user data slot. The benefits of the second method become apparent when working with multiple data values being attached to the entity. As more data is attached, a hierarchical system can be set up to access the information desired.

The following code adds a user data short (*0*), double(*25.5*) and chunk(“*My name is Sam*”) value to a defined entity:

```
UDShort = 0
UDFloat = 25.5
UDChunk = “This is the chunk data”
Call VCAddCurrentEntityUserDataShort(IERROR, UDShort)
Call VCAddCurrentEntityUserDataFloat(IERROR, UDFloat)
Call VCAddCurrentEntityUserDataChunk(IERROR, UDCHUNK, Len(UDCHUNK))
```

User Data Retrieval

User data is retrieved from both entities and the drawing header. Typically, an application knows the order and type of data attached. However, when working with custom data not attached by your application the API can be used to filter out the types and placement positions.

Steps

1) Set the required unique user data name in the application initialization routine.

Please see Attaching User Data.

2) Retrieve the information from the drawing header or the current entity.

To access information in the drawing header, the user data section needs to call VCSetHeaderData. Otherwise, user data is accessed from the current entity. The current entity can be set in a number of ways including any parsing routine. The current entity can be accessed directly through the API.

The following code specifies to retrieve the user data from the drawing header rather than a specific header:

```
Call VCSetHeaderData(IERROR)
```

The following code should be used to return the user data from the drawing entities themselves:

```
Call VCSetEntitySection(IERROR)
```

3) Retrieve the user data count for the unique user data ID.

This count returns the number of valid user data segments attached to the entity. This information can then used to loop through the retrieval process.

The following code retrieves the amount of data attached at the current location.

```
IATTACHEDDATACOUNT = VCGetCurrentEntityUserDataCount(IERROR)
```

For each index, retrieve the type of user data information attached. This step can be avoided if the application remembers and accesses the information all in the same order each time.

The following code retrieves the user data kind at the specified index. Typically this would be done in a loop statement to check all the data on the entity.

```
For IINDEX = 0 To IATTACHEDDATACOUNT - 1
IUDKIND = VCGetCurrentEntityUserDataKind(IERROR, IINDEX)
Select Case IUDKIND

End Select
Next IINDEX
```

4) Retrieve the data from the specified index.

If the data type is not known then typically this would involve a statement to retrieve the proper entity type.

The following code works in conjunction with the previous values to retrieve the data into the proper type

```
Select Case iUDKind
Case FLOAT
VCGetCurrentEntityUserDataFloat(IERROR, IINDEX, rFloatValue)
Case SHORT
VCGetCurretnEntityUserDataShort(IERROR, IINDEX, rShortValue)
Case CHUNK
rSize = VCGetCurrEntUserDataChunkSize(IERROR, IINDEX)
VCGetCurrentEntityUserDataChnunk(IERROR, IINDEX, rChunkValue)
End Select
```

Adding User Data to a Drawing Header

User data is a powerful feature that can be implemented to store specific information with entities. It may be necessary to also store information within the drawing header. A drawing property sheet in which specific data such as total editing time or a drawing description or held in the header for review and document management is a good example. Corel Visual CADD allows an application to attach user data either to a saved drawing in which the information is saved with the drawing and only accessible when that drawing is open. The property sheet example is a demonstration of this situation in which the added information is specifically related to that drawing. Other situation may require an application to create a default or environment setting available and loaded each time a drawing is made active or created. A special flag to set initial values in an application is a good example. An application can store this general information in the Corel Visual CADD default environment which is then automatically loaded each time a drawing is activated.

Steps:

1) Set the required unique user data name in the application initialization routine.
In order to set space in memory on each entity or in the drawing header, a user data name should be specified to reference the information.

The following code sets a user data name of “Sample Data”:

```
Call VCSetUserDataName(IERROR, “Sample Data”)
```

2) Set the user data information to the drawing header.
In order to access information in the drawing header, make a call to VCSetHeaderData. Otherwise, user data is attached to the current entity.

The following code specifies that the user data is attached to the drawing header rather than a specific header:

```
Call VCSetHeaderUser(IERROR)
```

3) Determine where the data will get attached.
User data will only get attached to the default environment if the active drawing has not been saved. If a drawing has been saved then data will attach itself to the drawing and only be accessible when that drawing is active. Therefore it is necessary to attach user data to an non-saved drawing when creating a default application setting. An application can determine if a drawing has been saved by checking the drawing name with VCGetDrawingName. Typically, it is not necessary to track the actual drawing name, instead an application can simply verify a drawing has been saved if the return value for VCGetDrawingName is greater than zero. If it is equal to zero then the drawing has not been saved and the data will be attached as setting to the default environment.

The following code checks if the active drawing has been saved.

```
If (VCGetDrawingName() > 0) Then
    ‘THE DRAWING HAS BEEN SAVED
End IF
```

4) Attach the data.
After determining where the data will be attached any user data routine can be used to add data to the drawing. The Corel Visual CADD API allows two methods for attaching the user data information. The easiest method is to simply use the add routines which add the data to the next available slot in the user data segment. The API also allows user data to be set and placed at a specific index within the user data slot. The benefits of the second method become apparent when working with multiple data values being attached to the entity. As more data is attached, a hierarchical system can be set up to access the information desired.

The following code attaches user data to a saved drawing. The attached information will only be available when the drawing is active. This would represent situations where the data is specific to the drawing and not an environment setting.

```
If (VCGetDrawingName() > 0) Then
    Call VCAddCurrentEntityUserDataShort(IERROR, 1)
End IF
```

The following code attaches user data to the drawing environment. The attached information is then saved with all drawing and the default environments.

```
If (VCGetDrawingName() = 0) Then
  Call VCAddCurrentEntityUserDataShort(IERROR, 1
End IF
```

NOTE: The slight difference in the previous code is represented by the greater than (>) and equal (=) signs.

5) Reset the data back to the entity definitions.
The user data task will remain in the drawing header until it is switched back to the entity section.

The following code switches the parsing section back to the entity definitions.
Call VCSetEntitySection(IERROR)

Command Line Interaction

Corel Visual CADD provides direct access to the command line for your application to use. Typically, command interaction is handled better directly through API routines. However, certain situations call for the user to have direct access to the command line. Information that can be captured through the command line includes two letter command sequences, number entry, and prompt text. By capturing the command line, the external application can process the key event before Corel Visual CADD manipulates the event. An example of this is applications utilize special two letter command structures that overwrite the built in commands.

Steps:

1) Set the application to accept key board input.

In order to retrieve any keyboard activity from within the Corel Visual CADD interface, an alert application must be set. This allows Corel Visual CADD to pass the key press events to that application. Once the application receives the events it can process the input prior to Corel Visual CADD. In this manner an application could over right the 2-letter command structure or retrieve values directly from the prompt.

The following code registers the application in the Corel Visual CADD messaging loop.

```
Call VCSetAlertApp(IERROR, HWND, iCode)
```

2) Capture the first command line character in the application keypress event.

An application will receive all key press events once registered with the messaging loop. The application should then filter out the desired key strokes. In many situations an application may simply need the <ESC> key sequence in order to terminate an applications process. In others it may need to retrieve a distance input or rotation angle from the command line. In these situations an application can retrieve the current key with the applications process and use VCGetCmdStr to return the previous input for the command line.

The following code placed in the Key_Press event handler of the application will retrieve the current key value and use VCGetCmdStr to retrieve the values prior to this.

```
IRETURNEDSTRINGLENGTH = VCGetCmdStr(IERROR, RETURNEDSTRING)
```

The following sample code demonstrates a typical scenario to capture an input value after the <ENTER> key.

```
If KeyAscii = 13 Then
 RETURNEDSTRINGLENGTH = VCGetCmdStr(IERROR, ENTEREDVALUE)
End If
```

3) Clear the application from the messaging loop.

As with any user tool, the application should be cleared from the messaging loop on exit. This frees the application from the Corel Visual CADD messaging loop and terminates any event processing from the Visual CADD interface.

The following code clears the application from the messaging loop.

```
Call VCClearAlertApp(IERROR, CMDLINE.HWND)
```

Error Checking

It is important for error checking to be handled through the API. While each call returns an IERROR value that can be checked, it is typically used in a debug environment rather than in actual code testing. There are, however, several error checking routines that should be implemented in order to ensure proper stability. These routines range from simple version checking routines to in-depth world checking before any drawing interaction begins.

Version checking

When running external applications, it is a good idea to check the version to be sure that all the API‘s needed by the application are supported in the version in use. Corel Visual CADD version numbers are broken into four parts. These are from greatest to least significance; Major, Minor and Dot. Major and minor are the most important and should always be checked. For example this API document was designed around Corel Visual CADD 2.0.1(major version 2, minor version 0, minor dot version 1). Dot versions are typically maintenance release versions.

Steps:

Check the version numbers.

The following code retrieves the version numbers for error checking:

```
IMAJOR = VCGetMajorVersion()
IMINOR = VCGetMinorVersion()
IDOT = VCGetMinorDotVersion()
If (IMAJOR >= 2) And (IMINOR >= 0) And (IDOT >= 0) Then
Else
 MsgBox "ERROR CODE MESSAGE FOR VERSION"
End If
```

Valid World Checking

The most common cause for API routines to fail is an invalid drawing world. Corel Visual CADD provides a Multiple Document Interface (MDI) that allows several drawings opened at once. Each of these drawings are opened into separate drawing worlds. The world handles begin at zero and are incremented with each new drawing opened. The Corel Visual CADD API provides the ability to create new drawings both in the Visual CADD interface and in an external application. Whenever interacting with any of these drawing worlds it is necessary to validate the current world.

Steps:
Check the current world.

The following code checks the current drawing world in order to make sure it is valid.:

```
IVALIDWORLD = VCIsCurrentWorldValid()
If (IVALIDWORLD = 0) Then
   MsgBox "ERROR MESSAGE FOR INVALID WORLD"
End If
```

Initialization Check

When creating an external application, the DLL's must be initialized in order to activate API routines. An application that relies on the Corel Visual CADD interface should also check to ensure the Visual CADD engine has been activated. In order to check if the DLL's have been initialized, an application should call VCGetInitCount.

Steps:

Check the initialization count.

The following code checks the DLL initialization count:

```
IINITCOUNT = VCGetInitCount()
If (IINITCOUNT = 0) Then
 MsgBox "ERROR MESSAGE FOR INITIALIZATION COUNT"
End If
```

Database Operations

The API provides all the necessary parsing routines for accessing information in the drawing database. In order to access information from entities within the drawing, an application will have to use a database parsing routine. This routine can be a specific entity call such as VCSetCurrentEntity or a general parsing routine such as VCFirstEntity. The parsing routines can range from a simple translator that moves through the database from beginning to end in order to write to a specified format or it can be a routine to modify existing properties with the database. Anytime an entity is added to the database it is placed at the end of the database and given a unique ID. This ID can be used to access information or modify properties directly after the operation. The following samples will demonstrate the methods for moving through the database to obtain the desired entity properties. These methods include a simple parse through each entity in the drawing to a complex filtering method for obtaining specific information quickly.

Parsing the Database

The simplest form of drawing database interaction is to parse from the first entity in the file and work through each entity until the file is finished. This parsing method requires the two routines VCFirstEntity and VCNextEntity.

Move to the first entity in the drawing data base and loop through the database.
The following code demonstrates the proper method for implementing these routines to search the drawing file:

```
If VCFirstEntity(IERROR, ENTITYTYPE) Then
  Do While IERROR = 0
    'Do some sort of processing
    If VCNextEntity(IERROR, ENTITYTYPE) Then
    End If
  Loop
End If
```

Parsing a Filtered Entity List

When a parsing routine is needed to find specific entities or entity types in the database, a selection filter routine can be implemented. For example, a database search may only require entities on layer 10 or symbols name “TESTSYMBOL”. Instead of using a slow parsing routine that checks the entire database, a selection filter can be implemented to pick the entities directly out the database. This method is much easier and faster in finding specific types on entities.

1) Optional: Turn filter highlight off.
The filtering method works of a specialized selection filter. Since it may not be desirable to display this filtered list to the end user, the application should turn highlight information while processing the entities.

The following code turns the selection highlight off:

```
Call VCSetHighlight(IERROR, 1)
```

2) Optional: Reset the filter.
The application should reset the filter to clear out any existing filter information. By resetting the filter an application is assured no other information in either the user settings or an application settings are used.

The following code resets the selection filter:

```
Call VCFilterReset(IERROR)
```

3) Set the new filter criteria.
The filter can be applied on a variety of entity types, names, colors, layers and widths. You can use any combination of these filter settings. The entity type Ids are presented in Appendix B.

The following example demonstrates a filter for symbols with the name “TestSymbol” that are on layer 10:

```
Call VCSetFilterKind(IERROR, 7)
Call VCSetFilterName(IERROR, “TestSymbol”)
Call VCSetFilterLayer(IERROR, 10)
```

4) Set the filter active.
The following code sets the active filter:

```
Call VCSetFilterActive(IERROR, 1)
```

5) Select the entities.
The parsing filter works on a selection list. Since the filter has been set to select specific types of entities, when you select all of the entities in the database, only those that meet the parsing filter will be selected.
The following code selects all entities through code:

```
Call VCSelectAll
```

6) Parse the filtered selection.
After setting the filter list, specialized parsing procedures are available to move through the entities in this list. These are parsing routines are analogous to the standard parsing routines.

This following code parsed through the filtered selection list:

```
If VCFirstSelected(IERROR, ENTITYTYPE) Then
  Do While IERROR = 0
    ‘Do some sort of processing
    If VCNextSelected(IERROR, ENTITYTYPE) Then
    End If
  Loop
End If
```

7) Deselect the entities.
The selection list should be cleared after the processing has finished.

The following code deselects all the entities in the drawing:

```
Call VCDeselectAll
```

8) Turn the filter off.
The following code turns the selection filter off:

Call VCSetFilterActive(IERROR, 0)

9) Optional: Reset the filter.
The filter should be reset after the operation to clean up any existence of the application from the user interface.

The following code resets the selection filter:
Call VCFilterReset(IERROR)

10) Optional: Turn the highlight display.
The filtering method works of a specialized selection filter. Since it may not be desirable to display this filtered list to the end user the application should turn highlight information of while processing the entities.

The following code turns the selection highlight off:
Call VCSetHighlight(IERROR, 1)

Parsing an On Screen List

The API also provides access to parsing entities that are currently on-screen. This allows an application to base the parsing list on only those items present at the current zoom. This is beneficial in situations where specific areas of the drawing need to parsed and filtered. If the current zoom encompass the entire area, then this method simply behaves the same as the standard parsing method, moving through each entity in the drawing.

1) Reset the on screen list with the new zoom

When working with display and zoom related parsing methods, the items selection list needs to be reset prior to each parsing string. This ensures the current zoom, those changed directly through code or from the user interface, are reflected in the database search.

The following code resets the screen list for the current zoom:

```
Call VCResetOnScreenList(IERROR)
```

2) Parse the list.

The routines to parse the list coincide with standard methods of moving from the first item in the list to the next until the end of the list is reached.

The following code parses the new screen list:

```
If VCFirstOnScreen(IERROR, ENTITYTYPE) Then
  Do While IERROR = 0
    'Do some sort of processing
    If VCNextOnScreen(IERROR, ENTITYTYPE) Then
    End If
  Loop
End If
```

Parsing an Expanded List

When the normal parsing routines are used, certain entities are treated as solid entities. For example, the normal parsing routines will parse symbol and hatch entities as a single object even though they are made of several entity types. A symbol definition can be parsed separately. Please see the Symbol section of this guide. However, when creating a translator that does not allow certain complex objects such as symbols and attached patterns, it may be necessary to parse inside the entities and gain information from the defining entity. This can be accomplished by exploding the objects and then parsing them with normal routines. Remember that this can have adverse affects on the original drawing. Instead by using VCFirstEntityExpand and VCNextEntityExpand, the entire entity set, even those within complex objects, can be parsed by the application.

Parse the expanded list.
The routines to parse the list coincide with standard parsing methods of moving from the first item in the list to the next until the end of the list is reached.

```
If VCFirstEntityExpand(IERROR, ENTITYTYPE) Then
  Do While IERROR = 0
    'Do some sort of processing
    If VCNextEntityExpand(IERROR, ENTITYTYPE) Then
    End If
  Loop
End If
```

Retrieving Entity Properties

There are two methods for retrieving the properties of an entity. Once an entity has been set as current, the properties can be retrieved. The first method is to match the properties of the entity and then use the appropriate routine to retrieve the system settings. The second method allows an application to directly access the properties from the entity definition.

Steps:

1) Optional: Save the current system settings.
When working with entity properties, it is generally desirable to maintain the current settings in a temporary buffer and restore the information on close. This allows a seamless integration into the interface and consistency with the tolls a user is currently working with.

The following code stores the settings into a temporary buffer:

Call VCSaveSettings

2) Parse the list for the desired entity.
The standard parsing structures presented in the previous sections are generally used to move through the entities and retrieve the properties. However, in order to retrieve property information, an entity need only to be set as current within the database. This can done by the parsing methods or directly if the entity handle is known.

The following code demonstrates setting the current entity based on a known ID. Typically, this ID is found through another parsing method. This code sets the current entity as number 10.

VCSetCurrentEntity(IERROR, 10)

3) Optional: First Method. Match the entity properties in the settings and use a settings retrieval routine.
This method requires the application to match all the entity properties and set them as the global variable in the user interface. Typically, when using this method, it is good practice to save the current user settings prior to processing and then retiring them on exit.

The following code matches the current entity properties and retrieves the color index corresponding to the entity property.

Call VCMatchCurrentEntity(IERROR)
ICOLOR = VCGetColorIndex(IERROR)

4) Optional: Second Method. Use a current entity routine to get the desired property.
The second method does not require the application to retrieve the entity property into a global variable. The information can be retrieved directly from the entity property settings.

The following code retrieve the color setting directly from the entity property.

ICOLOR = VCGetCurrentEntityColor(IERROR)

5) Optional: Restore the settings.
When working with entity properties it is generally desirable to maintain the current set settings in a temporary buffer and restore the information on close back to these values. This allows a seamless integration into the interface and consistency with the tolls a user is currently working with.

The following code restores the settings from the temporary buffer:

Call VCRestoreSysSettings

Parsing a Hatch/ Fill Boundary

Hatch and fill entities are defined by a boundary made up of different entities. For example, a hatch can be defined by a rectangle and a circle. Both are part of the hatch definition and are maintained as subentities in the Corel Visual CADD database. An application can access these boundary entities to retrieve properties, set new value and effectively recreate the hatch entity. In order to access these entities an application needs to set up a specialized parsing function to move through the entities.

Steps:

1) Retrieve the ENTITYHANDLE for the hatch or fill entity.

In order to access the boundaries of a hatch or fill entity, an application needs to retrieve the handle for the entity. This handle is usually retrieved with VCGetCurrEntityHandle after utilizing another database parsing routine. Please see the rest of the database parsing section for details on the methods available for accessing entity information.

2) Set a parsing loop and determine the number of contours defining the boundary.

The hatch boundary can be defined by multiple paths. Using the previous example of the rectangle and circle, the hatch entity would be defined by two (2) contours, one for the rectangle and one for the circle. The number of contours can be retrieved with the API routine VCGetEntityContourCount.

The following example returns a count for the number of contours defining the entity. It assumes a known entity handle calculated through other database parsing methods:

```
For ICONTOURINDEX = 0 To VCGetEntityContourCount(iError, CURRHANDLE) - 1
```

3) Set a parsing loop and determine the number of entities defining each contour.

Each contour is defined by separate entities. The rectangle and circle example will return the following values as the application loops through the contours. It will return four (4) for the first contour, there are four lines defining the rectangle. It will return a one (1) for the one circle entity defining the second contour.

The following example returns the number of entities defining each contour and loops though the list:

```
For ISUBCOUNT = 0 To VCGetEntitySubEntityCount(iError, CURRHANDLE, ICONTOURINDEX) - 1
```

4) Set the subentity and retrieve the desired entity information.

The subentity is then set using VCSetCurrentEntitySubEntity. Once the subentity is set the standard query functions may be used to retrieve the entity information.

The following code sets the current subentity:

```
Call VCSetCurrentEntitySubEntity(IERROR, ICONTOURINDEX, ISUBCOUNT)
```

Modifying Existing Entities

It is possible to directly change the properties of any existing entity directly through the API. By combining commands that set entity properties or recreate the entity, drawing modifications can easily be done. It is not necessary to duplicate or recreate an entity in order to change existing properties. Sometimes, though, it can be desirable to recreate the entity in order to create a new entity that will be added to the end of the database. This entity then takes on a new entity handle for manipulation. Many times it is necessary to delete the old entity whenever an entity is duplicated. The application should call VCSaveSettings when changing entity and system variables. This will save the user's current settings to a temporary buffer. These settings can then be restored to the user's default with VCRestoreSettings.

Changing Entity Properties Using Eattr

Most of the Corel Visual CADD entities are defined by the four basic properties of color, line type, line width and layer. Some entities however are defined by a more complex set of properties and special routines are required to modify these settings. These special entities include hatches, dimensions, and text entities. In order to modify these entities another method must be used. Typically this would be accomplished by a duplication method presented later. When working with basic entity types however a quick routine is available to change the basic property settings of the entity. This method is VCChangeSelected , and allows an application to change the properties of selected items without parsing the drawing database and recreating the entities themselves. This method will allow for undo levels automatically by the routines execution.

Step:

1) Define the property values for the Eattr structure.

The Eattr structure contains integer based definitions for the basic entity properties. Instead of using individual routines to set the color, line type, line width and layer values, a single structure is built to hold these values. This structure is then passed to the VCChangeSelected command to change the property values for the entities.

The following code sets the color to 9, the layer to 1, the line width to 0 and the line type to 0:

```
EATTR.color = 9
EATTR.layer = 1
EATTR.linetype = 0
EATTR.linewidth = 0
```

2) Change the properties of the selected items.

The routine will automatically change all entities in the current selection list. There is no need to parse the database and explicitly recreate the entities with the new properties.

The following code changes the selected entities to the predefined properties:

```
Call VCChangeSelected(EATTR)
```

Applying Settings to an Entity

The simplest method to alter the properties of an existing entity is to change the system entity properties and then apply them to an entity. The following steps illustrate this method in detail.

Steps:

1) Move to the desired entity in the database.
This can be done with all the general parsing routines or directly with a set entity commands.

The following code moves to the first entity in the selection list:

```
If VCFirstSelected(IERROR, ENTITYTYPE) Then
Do While IERROR = 0
```

2) Save the current settings.
It is generally desirable to maintain the user default settings prior to changing the values. These settings can then be restored at the end of the change command.

The following code saves the current user settings to a temporary buffer:

```
Call VCSaveSettings
```

3) Set the desired new properties.
Any or all entity properties may be changed.

The following code sets the color index to 10, the line width index to 2 and the layer index to 1:

```
Call VCSetColorIndex(IERROR, 10)
Call VCSetLineWidthIndex(IERROR, 2)
Call VCSetLayerIndex(IERROR, 1)
```

4) Apply the settings to the current entity.
After the new settings have been made it is necessary to apply them to the current entity.

The following code applies the new settings to the current entity. This method does not allow any undo levels to be set. Please refer to the duplicating an entity section for other methods:

```
Call VCApplySettingsToCurrentEntity(IERROR)
```

5) Restore the user default settings.
The user settings should be returned to the position prior to the operation. VCRestoreSettings resets the settings based on the temporary buffer.

Duplicating an Entity

The properties of an existing entity can be changed by recreating the desired entity. While methods to retrieve the entity points and properties are available to allow this task, an easier routine, VCDuplicate, can be implemented to change only those properties desired and append the database with this completely new entity. This offers advantages since the operation can include undo and redo event handlers for the user. The following steps show how to duplicate an entity.

Steps:

1) Move to the desired entity in the database.
This can be done with all the general parsing routines or directly with a set entity command.

2) Save the current settings.
It is generally desirable to maintain the user default settings prior to changing the values. These settings can then be restored at the end of the change command.

The following code saves the current drawing settings to a temporary buffer:

```
Call VCSaveSettings
```

3) Set the desired new properties.
Any entity property can be set and changed with this operation. Certain entities have special properties relevant only to their type. For example, hatches contain pattern information while a dimension contains arrow setting values that are only valid for their respective entity type. These properties and other entity specific properties can be found in the A-Z Reference Guide.

The following code sets the color property to *blue*, the width index to *3* and the layer index to *30*:

```
Call VCSetColorIndex(IERROR, 9)
Call VCSetLineWidthIndex(IERROR, 3)
Call VCSetLayerIndex(IERROR, 30)
```

4) Store the current entity handle for reference.
Since the duplicated entity is appended to the end of the database, it is necessary to keep track of the current entity handle so that you can quickly move back to the original entity.

The following code stores the current entity handle in a temporary variable:

```
CURRHANDLE = VCGetCurrentEntityHandle(IERROR)
```

5) Duplicate the entity.
By duplicating the entity, the application is recreating the entity with all the new settings. The new entity is then appended to the end of the database.

The following code duplicates the current entity and applies the current system settings to the new entity:

```
Call VCDuplicate(IERROR, CURRHANDLE)
```

6) Draw the new entity.
After the entity is duplicated, it is not immediately drawn to the screen. The code should move to the new entity and draw it after it has been added.

The following code moves to the last entity in the database and draws it to the screen:

```
Call VCLastEntity(IERROR, LASTHANDLE)
Call VCSetCurrentEntity(IERROR, LASTHANDLE)
Call VCDrawCurrentEntity(IERROR)
```

7) Erase the old entity.
Typically it is necessary to erase the old entity from the drawing after the new one has been drawn.

The following code moves to the old entity and sets it as erased in the database.

```
Call VCSetCurrentEntity(IERROR, CURRHANDLE)
Call VCSetCurrentErased(IERROR)
Call VCDrawCurrentEntity(IERROR)
```

8) Optional: Restore the user default settings.

The user settings should be returned after the duplicating operations are completed. VCRestoreSettings resets the settings stored in the temporary buffer.

Duplicating an Entity with Transformation

The previous task allows any properties to be altered but allows for no input for altering the actual geometry of the entity. While other routines such as VCSetCurrentEntityPoint can be used to move the current entity to a new location, the API also provides routines that allow total manipulation of the entity geometry and properties. These API calls allows for a translation point, scale value, or rotation to be applied directly to a newly created entity. When working with these routines, it should be noted that all rotation and scaling are done via the drawing origin(0,0). Therefore, when rotating an entity, the net rotation should first be translated from the drawing origin, reset, and then returned to the starting position. The steps provided below will demonstrate this task by utilizing several VCDuplicateWithTransform statements to move and rotate the entity. While this is a complicated task, these steps should illustrate fully on how to transform entities through API calls.

Transformations allow the application to shift the entity from its current state using features such as scaling, shift in position, a rotation angle or a combination of any of these transformations all the while maintaining geometry and property information.

Steps:

1) Move to the desired entity in the database.
This can be done with all the general parsing routines or directly with a set entity command. For more information on parsing routines, please see Database Operations.

2) Save the current settings.
It is generally desirable to maintain the user default settings prior to changing the values. These settings can then be restored at the end of the change command.

The following code saves the current drawing settings to a temporary buffer:

Call VCSaveSettings

3) Set the desired new properties.
Any entity property can be set and changed easily through a variety of API calls. Certain entities have special properties relevant only to their type. For example, hatches contain pattern information while a dimension contains arrow setting values that are only valid for their respective entity type. These properties and other entity specific properties can be found in the A-Z Reference Guide.

4) Store the current entity handle for reference.
Since the duplicated entity is appended to the end of the database, it is necessary to keep track of the current entity handle so that you can quickly move back to the original entity.

The following code store the current entity handle in a temporary variable:

```
CURRHANDLE = VCGetCurrentEntityHandle(IERROR)
```

5) Optional: Apply a translation.
The translation moves an entity a specified distance in the x and y direction. This distance is input through a Point2D structure containing the x any y values. This value is the distance that the pair is to be moved, not the new coordinates for the new pair. For example, if the pair is (1,1) the entity will shift 1 unit in the x direction and 1 unit in the y. It will not move to the coordinates 1,1.

The following code retrieves the current constriction points and uses these values to shift the entity to the drawing origin.

```
Call VCGetCurrentEntityPointBP(IERROR, 0, CURRENTPOINT)
ORIGINPOINT.X = -CURRENTPOINT.X
ORIGINPOINT.Y = - CURRENTPOINT.Y
SCALEVALUE.X = 1
SCALEVALUE.Y = 1
ROTATIONVALUE = 0#
Call VCDuplicateWithTransform(IERROR, CURRHANDLE, ORIGINPOINT, SCALEVALUE, ROTATIONVALUE)
```

6) Optional: Delete the old entity.
Typically it is necessary to erase the old entity from the drawing.

The following code moves to the old entity and sets it as erased in the database.

```
Call VCSetCurrentEntity(IERROR, CURRHANDLE)
Call VCSetCurrentErased(IERROR)
Call VCDrawCurrentEntity(IERROR)
```

7) Optional: Retrieve the new entity handle.

For example, if the process is not completed and application needs to rotate the entity, the new entity handle should be retrieved.

The following code retrieves the handle for the new entity.
Call VCLastEntity(IERROR, CURRHANDLE)

8) Optional: Scale the entity.
The entity can be scaled to reflect changes in the entity property type. The scale value sizes the entity either up or down proportionally.

The following code scale the entity.
ORIGINPOINT.X = 0
ORIGINPOINT.Y = 0
SCALEVALUE.X = 12
SCALEVALUE.Y = 12
ROTATIONVALUE = 0#
Call VCDuplicateWithTransform(IERROR, CURRHANDLE, ORIGINPOINT, SCALEVALUE, ROTATIONVALUE)

9) Optional: Erase the old entity

10) Optional: Retrieve the new entity handle.

11) Optional: Rotate the entity.
The entity can be rotated any value from the original angle. For example a line slanted at 45 degrees can be rotated ± any angle directly without measuring the existing angle. The angle value is added or subtracted from the existing rotation from the horizontal. Any angle input should be in radians. The rotation occurs about the origin, therefore in cases where the entity is not located at the origin, the entity should be translated to the origin, rotated, and then translated back to the original position.

The following code rotates the entity 45 degrees.
ORIGINPOINT.X = 0
ORIGINPOINT.Y = 0
SCALEVALUE.X = 0
SCALEVALUE.Y = 0
ROTATIONVALUE = 0.7854
Call VCDuplicateWithTransform(IERROR, CURRHANDLE, ORIGINPOINT, SCALEVALUE, ROTATIONVALUE)

12) Optional: Erase the old entity

13) Optional: Retrieve the new entity handle.

14) Optional: Translate the entity back to the original point.

The following code moves the rotated item bask to its original location.
ORIGINPOINT.X = 0
ORIGINPOINT.Y = 0
SCALEVALUE.X = 0
SCALEVALUE.Y = 0
ROTATIONVALUE = 0.7854
Call VCDuplicateWithTransform(IERROR, CURRHANDLE, ORIGINPOINT, SCALEVALUE, ROTATIONVALUE)

15) Optional: Erase the old entity

16) Optional: Retrieve the new entity handle.

17) Optional: Restore the original settings.

Customizing Corel Visual CADD

The Corel Visual CADD interface can be fully customized. This includes the two-letter key structure, menu including context sensitive popup menu and the toolbars.

Custom Commands

Corel Visual CADD supports the definition of custom commands or user defined tools. These commands are contained in the CMDEXT.DEF file located in the Corel Visual CADD system directory. All other custom files, such as the TOOLPAL.CST and the custom menus, get their information from the CMDEXT.DEF. The format for this file is as follows:

Native Command, Two Letter Command, Bitmap File, Name to appear on menu, Status Line Description, Script

For example, a window erase command would look like this:

WINERASE,WE,D:\BITMAP\BITMAP.BMP,&WINDOW ERASE,PLACE WINDOW,SW;@;@;ER;

Native Command

This is the internal command name for the custom command. This can be used in the *.CST or *.MNU files to place the command on a button or in a custom menu. This parameter should be a single word, 12 characters or less. In our example, WINERASE is the name of the Native command.

Two Letter Command

This is the two (or three) letter shortcut that will provide access to the command via the keyboard. If the two letter command is already used in the ALIAS.CMD that command will have precedence over the one defined in CMDEXT.DEF. If using three letters, be aware that if another command uses the same first two letters the other command will also have precedence. All commands must start with a letter.

Bitmap File

This is the path and name of the bitmap to be placed on the button if the icon is placed on a toolbar. Corel Visual CADD uses 20x20 pixel buttons and will shrink or grow any bitmap to this size. If a path is not designated, Corel Visual CADD will search in the custom directory as designated in the path settings for the bitmap. For more information on customizing toolbars, see "Customizing Toolbars".

Menu Description

This specifies the text that will appear on a menu if the custom command is placed in the custom menu. The "&" Symbol placed before a letter designates the letter that will be the "Hot Key" or the keyboard shortcut for that item on the menu. The "Hot Key" will appear underlined.

Status Line Description

When the user passes the cursor over the command on the menu or an icon button, the status line description will appear to give the user an idea of the tools function or prompt the user for information.

Script

The is the actual script the native command performs. (For more information on writing scripts see Creating Command Aliases). As in the SCRIPT.DEF file, all commands must be delimited by semi-colons (;).

To run an executable file:

EXENAME - this defines the name of an executable to be run when the RUN command is used. If no path is set, then Corel Visual CADD will search in the custom directory as designated in the path settings for the executable file.

RUN - executes the program as designated by EXENAME

An example of this might be:

HATCHCHANGE,HC,C:\HATCH.BMP,HATCH &CHANGE ,SELECT HATCH, EXENAME;HATCH.EXE;RUN;

This uses an external program to execute the hatch change instead of an internal script. While in some cases it may not be advantageous to use an external program, external programs are the most are the easiest way to have complicated user tools load up.

Creating Command Aliases

The icons, speed bars, pull-down and popup menu provide an interface that makes it easy to learn and use Corel Visual CADD. However, many users prefer the productivity gains that can be accomplished by accessing commands directly with two or three-letter keyboard commands. Corel Visual CADD not only allows most commands and functions to be accessed through two or three-letter keyboard entry, but also allows you to invent your own two or three-letter “aliases” for the commands! These aliases may be used directly by keyboard entry, or in scripts.

When you load Corel Visual CADD, a text file called ALIAS.CMD is read into memory. This file includes a list of all Visual CADD “native” command names, each next to its 2 letter command (if any). When you type a 2 letter command, Corel Visual CADD checks the list of aliases for a match. If it finds one, then it executes the native command associated with that alias. If there is no alias or native command name matching what you type, Visual CADD ignores the input.

Native command names are longer than the alternative two or three-letter alias commands (note: the Enter key is not required to end either alias or native commands), so alias commands names are preferable for efficient keyboard entry. However, because native commands are “hard coded” into Corel Visual CADD, they are preferable for use in scripts that will be distributed to other Visual CADD users who may not be using the same alias names.

To customize the alias command names, load ALIAS.CMD into a text editor such as the Windows Notepad program, the DOS Edit program, or any word processor (using text mode). Note that each line in the file starts with the currently assigned alias name, followed by a comma, then the native command. name. The commands appear in the ALIAS.CMD file in the following format:

[Alias, Native Command]
PO,Point
LS, Single Line
LC, Continous Line
R2,Rect2
R3,Rect3
...

You may change the alias name to any two or three-letter text you wish. Make sure, however, that these letters do not conflict with the starting letters of any other alias or native command name (you can easily check this by using the search function in your text editor).

For example, suppose you want to type “RE3” instead of “R3” to start the 3-point rectangle command. First, note that the listing for that command reads “R3,Rect3” in ALIAS.CMD. Simply change “R3” to “RE3” (do *not* alter the native command name, Rect3), and save the change under the same file name. The next time you start Corel Visual CADD, RE3 will be the new short name for the 3-point rectangle.

Tip

- Most of the native command names are self-explanatory. However, if you need more information, all native commands are explained in the text file called NATIVE.TXT.

Custom Menus

Menus can be altered, added, deleted or rearranged. Native and Custom Commands may be added or removed from existing menus. Custom menus are saved in ASCII text files with .MNU extension. Custom menus can be loaded into Corel Visual CADD using several different methods. The Load Menu command is designed specifically to load custom menus automatically via command line or Registry settings (it will not load mouse menus or toolbars).

The format for this file is as follows:

POPUP "&File"// Define start of Popup Menu "File"

FileOpen// Native Command

"&New File", FileNew//"Description", Native Command

Separator// Separator

"&Close", 2405//"Description", Native Command ID

Separator// Separator

FileSave// Native Command

FileSaveAs// Native Command

.

.

FileExit// Native Command

POPUPEND

POPUP "&Edit"// Define New Popup menu "Edit"

Undo// Native Command

Redo// Native Command

SEPARATOR// Separator

.

.

All commands are either a native command or a custom command defined in the CMDEXT.DEF file. If the item is a native command, the menu description is stored internally. If the command is defined in CMDEXT.DEF, then the menu description is taken from the Menu Description parameter in the CMDEXT.DEF file.

"Description", Native Command

Same as above, but overrides the menu description of the native command with specified text.

"Description", Native Command ID

Same as above, but uses an associated Corel Visual CADD Native Command ID. It is not recommended using these IDs as they are not published and can change without notice. They are documented and supported for compatibility with Microsoft Windows Menu ASCII format.

POPUP "Menu Name"

Defines the start of a popup menu with name "NAME". All commands following POPUP and preceding POPUPEND will be included in the popup menu

POPUPEND

Defines the end of specified popup menu

SEPARATOR

Creates a separator line in the menu

NOTE:

The following items are optional but are included for compatibility with Microsoft Windows Menu ASCII format.

MENUITEM Native Command

Used before a Native Command. Has no effect

{

Same as POPUP

}

Same as POPUPEND

Custom Mouse Menus

When creating custom tools it may be necessary to create or modify the context sensitive mouse menus to be used with the tool. This menu appears when the user clicks the right mouse button in the drawing area while using a tool. Customization of mouse menus is accomplished using ASCII text files: a pointer file (MOUSEMNU.DEF) and an ASCII menu file for each tool.

MOUSEMNU.DEF

MOUSEMNU.DEF is an ASCII text file which contains all the current Corel Visual CADD tools and their default menu files.. It resides in the System Path and can be modified to include custom commands. (For more information on Custom Commands, please see Custom Commands.) Corel Visual CADD will use the specified *Menu File* if it can find it. When the user clicks the right button, Visual CADD will determine if a *Menu File* has been mapped to the tool. If so, Visual CADD will load the *Menu File* and create a popup menu on the fly. If *Menu File* not found, the tool's default Menu is used.

The format is as follows for **MOUSEMNU.DEF**:

Native Command, Path\Menu File
Native Command, Path\Menu File
Native Command, Path\Menu File
.
.

Example:

SymPlace,C:\VCADD\MENU\SYMPLACE.POP
LineCont,C:\VCADD\MENU\LINECONT.POP
LineSingle,C:\VCADD\MENU\LINESING.POP
LineDbl,C:\VCADD\MENU\LINEDBL.POP
Point,MENU\POINT.POP (see note)

NOTE: The Menu File path can be explicitly specified, C:\VCADD\MENU\SYMPLACE.POP. Or if the drive is not explicitly specified, then Corel Visual CADD will contact the Menu File to the System Path. For instance, if the System Path is C:\VCADD and Menu File is MENU\SYMPLACE.POP, then Corel Visual CADD will look for the Menu File in the directory C:\VCADD\MENU\.

Menu File

The *Menu File* is the ASCII text file that contains the actual menu information and design for a particular tool's mouse menu. The Menu File does not necessarily need to be named the same as the tool menu it is defining, (although it might be a good idea) but it must end in the extension .**POP**. The mouse menu can contain any of Corel Visual CADD's native commands or custom commands. Custom commands can also be defined locally or 'on the fly' for commands that are only available as long as this menu is active. **.CMDEXT** defines the beginning of the section in which custom commands may be defined. Format of custom commands within mouse menus is the same as in Visual CADD's CMDEXT.DEF with the exception of the bitmaps and 2 letter command, which are ignored. When Custom commands are used, Corel Visual CADD will search for commands defined in the *.CmdExt* section of the Mouse Menu first, then search commands defined in the file *CMDEXT.DEF,* and lastly Visual CADD's default native commands.

All menu items including Popup, Popupend and seperator are available in mouse menus as well. For more information on those items, please Custom Menu.

The following sample menu file defines the mouse menu illustrated in Figure 1.

Sample Menu File:

```
SYMPLACE.MNU
        OK                      // See Note below
        Match                   // See Note below
        SymLast                 // VCADD Native command
        Track                   // VCADD Native command
        NewHandle               // VCADD Native command
        Separator               // Separator Line
        SYMROT90                // Custom command defined below
        SYMROT45                // Custom command defined below
        SYMROT0                 // Custom command defined below
        Separator
        POPUP "Flip"            // Start Popup menu with name "Flip"
                SYMFLIPX                // Custom command defined below
                SYMFLIPY                // Custom command defined below
        POPUPEND                // End Popup menu
.CmdExt                         //Local Custom Command Area
```

SYMROT90, , ,Symbol Rotate +90,Rotate +90,SymRot;$SymRot+90; //Custom Commands

SYMROT45, , ,Symbol Rotate +45,Rotate +45,SymRot;$SymRot+45;

SYMROT0, , ,Symbol Rotate = 0,Rotate = 0,SymRot;0;

SYMFLIPX, , ,Symbol Flip X,Flip X,SymScX;-$SymScX;

SYMFLIPY, , ,Symbol Flip Y,Flip Y,SymScY;-$SymScY;

Note:

Some Corel Visual CADD mouse menus, particularly those involving settings that may be matched (such as rotation, height, etc.), have **OK** and/or MATCH hard coded at the top of the mouse menu. These cannot be removed.

Custom Toolbars

You can easily rearrange or add to the command buttons in the main speed bar and tool bar. The icons to be displayed in these bars are listed files MAINSBAR.CST and TOOLPAL.CST, which can be easily edited with any text editor. These files can be found in your Corel Visual CADD system directory.

To edit the toolbars, open the correct file in an ASCII text editor, such as Microsoft Windows Notepad, DOS's EDIT, etc. Make your changes based on the below information then save the file. The next time you open Corel Visual CADD, your changes will take effect.

A "//" (without the quotes) can be placed in front of a native command to act as a "REM". In other words, Corel Visual CADD will not place commands preceded by a // on a toolbar. It will simply ignore them.

MAIN SPEEDBAR

The following listing in MAINSBAR.CST would display a row of command buttons for the File New, File Open, File Save As, and Clear Drawing commands, respectively:

SEPARATOR

FileNew

FileOpen

FileSave

SEPARATOR

Clear

The commands are listed in native command format. The word SEPARATOR instructs Corel Visual CADD to place a small space in the sequence.

TOOL PALETTE

The following listing for TOOLPAL.CST creates a column of command buttons in the tool bar. Command names in the same row create a pop-out menu.

Selection

LineCont,LineSingle,LineDbl,Point

Rect2,Rect3

RPolyCen,RPolySide,IPoly

The last three lines create pop-out menus of additional tool buttons. Each button in the pop-out menus is specified by the comma-separated list of native command names. You can rearrange buttons, or add as many command buttons in the tool bar and pop-out menus as will fit on the screen.

Note:

Individual Properties, i.e. linetype, linewidth, color, or layer, can be placed on the Main Speedbar and will appear as dropdown combo boxes.

Registry Settings

The registry location for add-on utilities for Corel Visual CADD for Windows 95 and Windows NT should be standardized. This allows the technical support to help diagnose problems encountered during the running of Corel Visual CADD with add on software. This standard is not required but it is requested that all add on applications write to this registry location.

Windows 95 and Windows NT use the system registry instead of the INI file found in previous versions. The registry can be used to hold default values and settings required by an application. Corel Visual CADD stores these same default settings in the HKEY_CURRENT_USER key in the directory SOFTWARE\VISUAL CADD\2.0\. There are several sub directories used by various aspects of the program. All add on application should place information in the ADDON directory by creating an application specific sub directory. The following code presented in Visual Basic demonstrates how to write to the registry and this particular location.

Declarations: The following provide the constant and type declaration required to read from and write to the Windows registry. These should be placed in the Form_Declaration section for Visual Basic applications. NOTE: C/C++ applications simply include the standard <window.h>.

```
Const HKEY_LOCAL_MACHINE = &H80000002
Const HKEY_CURRENT_USER = &H80000001
Const ERROR_SUCCESS = 0&
Const KEY_QUERY_VALUE = &H1
Const KEY_ENUMERATE_SUB_KEYS = &H8
Const KEY_NOTIFY = &H10
Const SYNCHRONIZE = &H100000
Const READ_CONTROL = &H20000
Const KEY_SET_VALUE = &H2
Const STANDARD_RIGHTS_ALL = &H1F0000
Const KEY_CREATE_LINK = &H20
Const KEY_CREATE_SUB_KEY = &H4
Const KEY_ALL_ACCESS = ((STANDARD_RIGHTS_ALL Or KEY_QUERY_VALUE Or KEY_SET_VALUE Or
KEY_CREATE_SUB_KEY Or KEY_ENUMERATE_SUB_KEYS Or KEY_NOTIFY Or KEY_CREATE_LINK) And (Not
SYNCHRONIZE))
Const STANDARD_RIGHTS_READ = (READ_CONTROL)
Const REG_OPTION_NON_VOLATILE = 0          ' Key is preserved when system is rebooted
Const REG_SZ = 1
Const KEY_READ = ((STANDARD_RIGHTS_READ Or KEY_QUERY_VALUE Or KEY_ENUMERATE_SUB_KEYS Or
KEY_NOTIFY) And (Not SYNCHRONIZE))
'type defintion for windows API
Private Type SECURITY_ATTRIBUTES
        nLength As Long
        lpSecurityDescriptor As Long
        bInheritHandle As Boolean
End Type
```

These constants are defined in the Windows API help. In order to write to the appropriate registry location, a string constant should be used. The following is an example taken from the HATCH CHANGE sample application. You will want to refer to the registry settings used during the install of Corel Visual CADD for the actual locations.

```
Const REGISTRY_STRING = "SOFTWARE\VISUAL CADD\2.0\ADDON\HATCH CHANGE"
```

NOTE: An application should create its own sub directory under ADDON. The previous line creates a directory HATCH CHANGE. Your application should create a specific directory to store settings and defaults.

Reading: The following code can be used to read information from the registry. It utilizes the string value defined above to open a key holding the X, Y value of the last screen position. The code should be placed in the application startup routine. In Visual Basic this is generally the Form_Load or Sub Main procedures.

```
Dim hKey As Long
If (RegOpenKeyEx(HKEY_CURRENT_USER, REGISTRY_STRING, 0, KEY_READ, hKey) = ERROR_SUCCESS) Then
    If (RegQueryValueEx(hKey, "HatchXY", 0, REG_SZ, szHatchXY, 10) = ERROR_SUCCESS) Then
        iStringToken = InStr(szHatchXY, ",")
        FRM_HCHANGE.Left = CInt(Left(szHatchXY, iStringToken - 1))
        FRM_HCHANGE.Top = CInt(Mid(szHatchXY, iStringToken + 1))
    End If
    lReVal = RegCloseKey(hKey)
End If
```

The code uses the Window API routines to retrieve information from the registry.

```
Private Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As Long, ByVal samDesired As Long, phkResult As Long) As Long

Private Declare Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQueryValueExA" (ByVal hKey As Long, ByVal lpValueName As String, ByVal lpReserved As Long, lpType As Long, ByVal lpData As Any, lpcbData As Long) As Long              ' Note that if you declare the lpData parameter as String, you must pass it By Value.
```

Writing: The following code demonstrates writing to the registry. It is taken from the HATCH CHANGE sample application and places the current X, Y location into the registry.

```
Dim hKey As Long

Dim lDisposition As Long

Dim saSecurity As SECURITY_ATTRIBUTES

 Dim szOutputRegKey As String

szOutputRegKey = CStr(FRM_HCHANGE.Left) & "," & CStr(FRM_HCHANGE.Top)

If (RegCreateKeyEx(HKEY_CURRENT_USER, REGISTRY_STRING, 0, REGISTRY_STRING,
REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, saSecurity, hKey, lDisposition) = ERROR_SUCCESS) Then
    If (RegSetValueEx(hKey, "HatchXY", 0, REG_SZ, szOutputRegKey, Len(CStr(FRM_HCHANGE.Left))) =
ERROR_SUCCESS) Then
    End If
    Call RegCloseKey(hKey)
End If
```

The code uses the following Windows API routines to write information to the registry.

```
Private Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hKey As Long) As Long

Private Declare Function RegCreateKeyEx Lib "advapi32.dll" Alias "RegCreateKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String, ByVal Reserved As Long, ByVal lpClass As String, ByVal dwOptions As Long, ByVal samDesired As Long, lpSecurityAttributes As SECURITY_ATTRIBUTES, phkResult As Long, lpdwDisposition As Long) As Long

Private Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA" (ByVal hKey As Long, ByVal lpValueName As String, ByVal Reserved As Long, ByVal dwType As Long, ByVal lpData As String, ByVal cbData As Long) As Long              ' Note that if you declare the lpData parameter as String, you must pass it By Value.
```

Tool ID

Point	2101	Point
LineSingle	2102	Single Line
LineCont	2103	Continuous Line
LineDbl	2116	Double Line
Rect2	2104	2 point Rectangle
Rect3	2105	3 point Rectangle
Circle3	2106	3 point Circle
Circle2	2107	2 point Circle
CircDiam	2122	Diameter Circle
Arc3	2108	3 point Arc
Arc2	2109	2 point Arc
Ellipse	2110	Ellipse
BezierSingle	2111	Single Bezier
BezierCont	2112	Continuous Bezier
Curve	2121	Spline Curve
RPolyCen	2113	Center Regular Polygon
RPolySide	2114	Side Regular Polygon
IPoly	2115	Irregular Polygon
EllArc	2117	Elliptical Arc
UndoVertex	2125	Undo Vertex
UndoDim	2126	Undo Last Dim
ContLineEx	1191	Explode Cont. Lines
HatchBnd	2359	Hatch Boundary
HatchSel	2352	Hatch Selected
SeedHatch	2340	Seed Hatch
FillBnd	2360	Fill Boundary
FillSel	2353	Fill Selected
SeedFill	2341	Seed Fill
SymOpen	2407	Load Symbol
SymSave	2408	Save Symbol
SymPlace	2120	Symbol Place
NewHandle	2457	New Handle
SymReplace	2525	Symbol Replace
SymRemove	2529	Symbol Remove
SymCreate	2351	Symbol Create
SymExplode	2124	Symbol Explode
SymLast	2119	Last Symbol
SymCount	2320	Symbol Count
AttCreate	2365	Attribute Create
AttAttach	2364	Attribute Attach
AttEmbed	2367	Attribute Embed
AttOpen	2415	Attribute Open
AttSave	2416	Attribute Save
BackRD	2215	Backward Redraw
Regen	2201	Redraw
RegenArea	2202	Redraw Window
ZmArea	2203	Zoom Window
ZmIn	2204	Zoom In
ZmOut	2205	Zoom Out
ZmPan	2206	Pan
ZmAll	2207	Zoom All
ZmSel	2208	Zoom Selected
ZmPrev	2209	Zoom Previous

ZmView	2212	Zoom View
ZmValue	2211	Zoom Value
NameView	2213	Name View
BirdsEye	2423	Birds-Eye View
NewView	2422	New View
ZmAllView	2217	Zoom All Views
RegenAllView	2218	Redraw All Views
Copy	2231	Linear Copy
RadCopy	2248	Radial Copy
ArrayCopy	2249	Array Copy
MultiCopy	2230	Multiple Copy
EraseLast	2257	Erase Last
Erase	2232	Erase
Mirror	2233	Mirror
Move	2234	Move
Rotate	2235	Rotate
MovePt	2236	Move Point
Fillet	2237	Fillet
IntTrim	2238	Intersection Trim
Stretch	2239	Stretch
WinStretch	2265	Window Stretch
Trim	2240	Trim
Extend	2241	Extend
Break	2242	Break
MTrim	2243	Selection Trim
MExtend	2259	Selection Extend
MTrim1	2266	Selection Trim
Chamfer	2244	Chamfer
Align	2246	Align
FitScale	2247	FitScale
Scale	2256	Scale
Offset	2252	Offset
Explode	2250	Explode
CloseContr	2254	Close Contour
Change	2255	Change
Edit	2431	
Boolean	2262	Boolean Two Paths
Undo	2440	Undo
Redo	2441	Redo
CBCut	2442	Cut
CBCopy	2443	Copy
CBPaste	2444	Paste
Clear	2409	Clear Drawing
PackData	2309	Pack Data
PurgeLnt	2325	
LoadLnt	2326	
Track	2450	Track
Penup	2451	Pen Up
Selection	2449	Selection
Filter	2453	Filter
SelRibalog	2463	Selection Bar...
SelAll	2446	All
SelClear	2447	Clear List
SelInvert	2448	Invert Selection
SelWin	2454	Window
SelObj	2456	Object

SelCross	2460	Crossing
SelAdj	2461	Adjoining
SelLast	2470	Last
SelLay	2458	Layer
SelLastObj	2485	
QSearch	2216	Quick Search
SnPerp	2151	Snap Perpendicular
SnPara	2162	Snap Parallel
SnTangent	2152	Snap Tangent
SnClosestPt	2154	Snap Closest Point
SnMidPt	2155	Snap Midpoint
SnMid2Pts	2164	Snap Middle
SnObject	2156	Snap Object
SnIntersect	2157	Snap Intersection
SnNearPt	2158	Snap Nearest Point
SnCenter	2160	Snap to Center of Arc and Polygons
SnQuad	2159	Snap to Nearest Quadrant of Circle
SnLastPt	2161	Snap Last Point
SnPercent	2153	Snap Percentage
Absolute	6127	M.E. Absolute
Relative	6128	M.E. Relative
Basepoint	6129	M.E. Basepoint
SetBasePt	2214	Set Basepoint
AutoFillet	2541	Auto Fillet
OrthoMode	2542	Ortho Mode
CursorFree	2543	Cursor Free
FilletRad	2532	Fillet Radius
IncSnap	2163	Incremental Snap
ChamferDist	2536	Chamfer Dist
GridOrg	2540	Grid Origin
GridSize	2537	Grid Size
SnapGrid	2539	Snap Grid
GridDisp	2538	Grid Display
DimLin	2181	Linear Dimension
DimAng	2182	Angular Dimension
DimRad	2184	Radial Dimension
DimDia	2185	Diameter Dimension
DimOrd	2196	Ordinate Dimension
Datum	2180	Datum Dimension
Leader	2186	Leader
DimMoveTxt	2187	Dimension Text Move
DimSlideTxt	2188	Dimension Text Slide
DimEdit	2191	Dimension Edit
DimHorz	2192	Dim Direction: Horizontal
DimVert	2193	Dim Direction: Vertical
DimAlign	2194	Dim Direction: Aligned
DimAtAngle	2195	Dim Direction: At an Angle
DimSingle	2544	Dim Mode : Single
DimCumul	2545	Dim Mode : Cumulative
DimPart	2546	Dim Mode : Partitioned
DimHorzTxt	2591	Horizontal Text
DimInlineTxt	2592	In-Line Text
ProxFixed	2590	Proximity Fixed
DatumOff	1230	Datum OFF
DatumXY	1231	Datum (XY)
DatumX	1232	Datum (X only)

DatumY	1233	Datum (Y only)
MeasDist	2301	Measure Distance/Angles
MeasArea	2302	Measure Area
DigMode	2310	Tablet Mode
DigAlign	2312	Align Drawing
DigScale	2311	Trace Scale
MatchEnt	2305	Match Entity
MatchTool	2306	Match Tool
Text	2354	Text Line
TextEditor	2356	Text Editor
FileNew	2400	New Drawing
FileOpen	2401	Open Drawing
FileSave	2402	Save Drawing
FileSaveAs	2403	Drawing Save As
FileExit	2404	Exit Corel Visual CADD
FileClose	2405	Close Drawing
FilePrint	2406	Print Drawing
FilePlot	2412	Plot Drawing
FileMerge	2410	Merge Drawing
FileRun	2420	Run Executable File
FileName	1200	
PrintSetup	2411	Printer Setup
FileSend	2429	Send the current drawing through electronic mail
RFCreate	2428	Reference Frame
RFPlace	2430	Reference Frame
RFDispBnd	2620	Display Boundary
RFTrans	2624	Display Transparent
LoadStyle	2413	Open Style
SaveStyle	2414	Save Style
SaveEnv	2316	Save Environment
WinExec	2318	
LoadMenu	2419	Load Menu
LoadAscii	2426	Load Text File
Reset	2427	
DimCh	2170	Dimension Change
TextCh	2172	Text Change
HatchCh	2173	Hatch Change
HatchSet	2501	Hatch Settings
TextSet	2521	Text Settings
DBSet	2534	DB Line Settings
OrthoSet	2533	Ortho Angle
SelSet	2463	Selection Bar...
LayMgr	2513	Layer Mgr
SymMgr	2506	Symbol Manager
TabOptions	2517	Settings...
ScriptAssign	2366	Assign Script
ObjInfo	2504	Object Info
LayDisplay	2481	Layer Display
LayHide	2482	Layer Hide
LayPropDlg	2486	
TabGeneral	2552	General
TabCnstrnt	2553	Constraint
TabSystem	2551	System
TabPath	2554	Path
TabNumeric	2556	Numeric
TabText	2548	Text/Atb

TabHatch	2549	Hatch/Fill
TabDim	2550	Dimension
TabDimText	2555	Dim Text
TabLeader	2601	Leader
TabImpExp	2530	Import/Export Settings
DimTextSet	2507	Dim Text Settings
DimArrowSet	2508	Arrow Settings
DimExtSet	2509	Extension Settings
DimLineSet	2510	Dim Line Direction
DimTolSet	2511	Tolerance Settings
DimLeadSet	2519	Leader Text
DimStrSet	2520	Dim String Settings
DimDispSet	2515	Dim Display Settings
DimScaleSet	2653	Dim Scale
DimTextAlign	2605	Dim Text Align
ColorProp	2465	
LayerProp	2466	
TypeProp	2467	
WidthProp	2468	
Properties	3102	Properties
SetColor	1176	
SetLayer	1178	
SetType		1177
SetWidth	1180	
SymName	1102	
SymRot		1103
SymScale	1104	
SymScX	1105	
SymScY	1106	
SymExp	1107	
LTScaleW	1192	
LTScaleD	1193	
TextColor	1110	
TextLay		1111
TextLnSp	1112	
TextChSp	1109	
TextJust	1113	
TextHeight	1114	
TextRot	1115	
TextAspect	1116	
TextBold	1117	
TextItalic	1118	
TextFont	1119	
TextStr	1120	
FontConv	2317	Font Converter
OrAngVar	1125	
FilletRVar	1126	
Update	3373	
GridSizeX	1123	
GridSizeY	1124	
SymSnapOn	1129	
SymSnapOff	1171	
AllLayEdOn	1172	
AllLayEdOff	1173	
AllLaySnOn	1174	
AllLaySnOff	1175	

AutFilletOn	1134	
AutFilletOff	1135	
SolidOn		1136
SolidOff		1137
WallWidth1	1138	
WallWidth2	1139	
RPolyNSides	2535	Num Sides
RPolyIn	6130	Inscribed
RPolyCrcm	1142	Circumscribed
OrthoOn	1143	
OrthoOff	1144	
CursFreeOn	1145	
CursFreeOff	1146	
IncSnapOn	1147	
IncSnapOff	1148	
IncSnapSize	1150	
FilletPrvOn	1154	
FilletPrvOff	1155	
OffsetFixOn	1156	
OffsetFixOff	1157	
ChamDist1	1158	
ChamDist2	1159	
OffsetDist	1160	
PercSnapVal	1161	
NumCopies	1162	
RadCopies	1163	
NumRows	1164	
SetAngle	1166	
RadSpanAngle	1167	
SetScaleXY	1168	
SetScaleX	1169	
SetScaleY	1170	
UIOff	1181	
UIOn	1182	
HatchName	1183	
HatchRot	1185	
HatchColor	1186	
HatchScale	1184	
FillColor		1188
ExeName	1190	
ConstPts	2565	Construction Points
HandlePts	2566	Handle Points
CurveTanPts	2567	Curve Tangent Points
Hilite	2476	Selection Highlight
AllLayEdit	2478	All Layers Edit
SnapLayer	2479	Snap To All Layers
SymSnap	2480	Snap Inside Symbols
ArrowFlip	2600	Arrow Flip
ShowDash	2607	Show Dash
UseLayProp	1243	Use Layer Properties
UseSymClr	1244	Use Placement Color
DllRun	2323	
DllName	1194	
DllFunName	1195	
DllCmdLine	1196	
OleClass	2327	

OleMethod	2328		
OleDllName	1197		
OleClassName	1198		
OleFunName	1199		
OleCmdLine	1202		
WinVert		2376	Tile Windows Vertically
WinHoriz	2377	Tile Windows Horizontally	
WinCascade	2375	Cascade Windows	
WinArrange	2378	Arrange Icons	
CloseAll		2379	Close All Windows
SYSPath	1210		
VCDPath	1211		
VCSPath	1212		
VCFPath	1214		
DWGPath	1214		
DXFPath	1215		
HelpIndex	2372	Help Index	
HelpOnCmd	2381	Help	
HelpAbout	2361	About Corel Visual CADD	
RegEdit		3375	Registry Editor

Entity Types

UNKNOWN	0
LINE2D	1
ARC2D	2
CIRCLE2D	3
ELLIPSE2D	4
BEZIER2D	5
POINT2D	6
SYMBOL2D	7
TEXT2D	8
DIMLINEAR2D	9
DIMANGULAR2D	10
DIMRADIAL2D	11
DIMDIAMETER2D	12
FILL2D	13
HATCH2D	14
ATTRIBUTE2D	15
INTERPCURV2D	16
MINTERPCURV2D	17
ELLIPTICALARC2D	18
CONTINUOUSLINE2D	19
CONTINUOUSBEZIER2D	20
LEADER2D	21
REFFRAME2D	22
ORDDIM2D	23
POLYGON3D	101
POINT3D	102
LINE3D	103
SYMBOL3D	104
CONTINUOUSLINE3D	105

User Defined Types

Visual Basic	C/C++	Delphi
TypeDefs		
N/A	typedef long ENTITYHANDLE;	N/A
N/A	typedef long WORLDHANDLE;	N/A
N/A	typedef long GRAPHICHANDLE;	N/A
N/A	typedef unsigned long UID;	N/A
N/A	typedef short VBOOL;	N/A
iPoint2D		

Type iPoint2D
  x As Integer
  y As Integer
End Type
	typedef struct iPoint2D {
	  int x,y;
	} iPoint2D;
		Type
		iPoint2D=record
		  x: Integer;
		  y: Integer;
		end;

lPoint2D		

Type lPoint2D
  x As Long
  y As Long
End Type
	typedef struct lPoint2D {
	  long x,y;
	} lPoint2D;
		Type
		lPoint2D=record
		  x: Longint;
		  y: Longint;
		end;

Point2D		

Type Point2D
  x As Double
  y As Double
End Type
	typedef struct Point2D {
	  double x,y;
	} Point2D;
		Type
		Point2D = record
		  x: Double;
		  y: Double;
		end;

Point3D		

Type Point3D
  x As Double
  y As Double
  z As Double
End Type
	typedef struct Point3D {
	  double x,y,z;
	} Point2D;
		Type
		Point3D=record
		  x: Double;
		  y: Double;
		  z: Double;
		end;

EAttr		

Type EAttr
  color As Integer
  layer As Integer
  linetype As Integer
  linewidth As Integer
End Type
	typedef struct EAttr {
	  short iColor, iLayer, iLinetype, iLinewidth;
	} EAttr;
		Type
		EAttr=record
		  color: Integer;
		  layer: Integer;
		  linetype: Integer;
		  linewidth: Integer;
		end;

Rect		

Type Rect
  top As Double
  bottom As Double
  left As Double
  right As Double
End Type
	typedef struct Rect{
	  double top,bottom,left,right;
	} Rect
		Type
		Rect=record
		  top: Double;
		  bottom: Double;
		  left: Double;
		  right: Double;
		end;

PrintStruct		

Type PrintStruct
  PrintMode As Integer
  PageSize As Point2D
  Margins As Rect
  dScale As Double
  Origin As Point2D
  ScaleMode As Integer
  Orientation As Integer
  dRotation As Double
  PrintToFile As Integer
  SelectionOnly As Integer
	typedef struct PrintStruct{
	  BOOL PrintMode ;
	  Point2D PageSize;
	  Rect Margins;
	  double dScale;
	  Point2D Origin;
	  short ScaleMode;
	  short Orientation;
	  double dRotation;
	  BOOL PrintToFile;
	  BOOL SelectionOnly;
		Type
		PrintStruct=Rect
		  PrintMode: Integer;
		  PageSize: Point2D;
		  Margins: Rect;
		  dScale: Double;
		  Origin: Point2D;
		  ScaleMode: Integer;
		  Orientation: Integer;
		  dRotation: Double;
		  PrintToFile: Integer;

```
    DateStamp As Integer
    FastPreview As Integer
    AllColorsToBlack As
Integer
    PaperUnit As Integer
End Type
```

```
    BOOL DateStamp;
    BOOL FastPreview;
    BOOL AllColorsToBlack;
      short PaperUnit;
}PrintStruct;
```

```
    SelectionOnly: Integer;
    DateStamp: Integer;
    FastPreview: Integer;
    AllColorsToBlack: Integer;
    PaperUnit: Integer;
end;
```

PlotStruct

```
Type PlotStruct
    Port As Integer
    BaudRate As Integer
    DataBits As Integer
    Parity As Integer
    StopBits As Integer
    NumPens As Integer
    NumCarousels As Integer
    DPI As Integer
    UseCarousels As Integer
    UseLLOrigin As Integer
    SortColors As Integer
    Optimize As Integer
    BufferOutput As Integer
End Type
```

```
typedef struct PlotStruct{
    short Port;
    short BaudRate;
    short Databits;
    short Parity;
    short StopBits;
    short NumPens;
    short NumCarousels;
    short DPI;
    BOOL UseCarousels;
    BOOL UseLLOrigin;
    BOOL SortColors;
    BOOL Optimize;
    BOOL BufferOutput;
}PlotStruct;
```

```
Type
PlotStruct=Rect
    Port: Integer;
    BaudRate: Integer;
    DataBits: Integer;
    Parity: Integer;
    StopBits: Integer;
    NumPens: Integer;
    NumCarousels: Integer;
    DPI: Integer;
    UseCarousels: Boolean;
    UseLLOrigin: Boolean;
    SortColors: Boolean;
    Optimize: Boolean;
    BufferOutput: Boolean;
end;
```

Entity Types

```
Global Const LINE2D = 1
Global Const ARC2D = 2
Global Const CIRCLE2D = 3
Global Const ELLIPSE2D = 4
Global Const BEZIER2D = 5
Global Const Point2D = 6
Global Const SYMBOL2D = 7
Global Const TEXT2D = 8
Global Const DIMLINEAR2D = 9
Global Const DIMANGULAR2D = 10
Global Const DIMRADIAL2D = 11
Global Const DIMDIAMETER2D = 12
Global Const FILL2D = 13
Global Const HATCH2D = 14
Global Const ATTRIBUTE2D = 15
Global Const INTERPCURV2D = 16
Global Const MINTERPCURV2D = 17
Global Const ELLIPTICALARC2D = 18
Global Const CONTINUOUSLINE2D = 19
Global Const CONTINUOUSBEZIER2D = 20
Global Const LEADER2D = 21
Global Const REFFRAME2D = 22
Global Const ORDDIM2D = 23
Global Const POLYGON3D = 101
Global Const Point3D = 102
Global Const LINE3D = 103
Global Const SYMBOL3D = 104
Global Const CONTINUOUSLINE3D = 105
```

VC Supported File Types

```
Global Const FILE_VCD = 0
Global Const FILE_VCS = 1
Global Const FILE_VCA = 2
Global Const FILE_GCD = 3
```

Global Const FILE_CMP = 4
Global Const FILE_DWG = 5
Global Const FILE_DXF = 6
Global Const FILE_STY = 7
Global Const FILE_VCF = 8
Global Const FILE_EMF = 9
Global Const FILE_WMF = 10

Conversion Unit Global Constants
Global Const UNIT_INCH = 0
Global Const UNIT_FEET = 1
Global Const UNIT_MM = 2
Global Const UNIT_CM = 3
Global Const UNIT_M = 4

Display Unit
Global Const IN_DEC = 0
Global Const FT_IN_DEC = 1
Global Const FT_DEC = 2
Global Const IN_FRAC = 3
Global Const FT_IN_FRAC = 4
Global Const FT_FRAC = 5
Global Const MIL = 6
Global Const CEN = 7
Global Const MET = 8
Global Const ANG_DEG = 9
Global Const ANG_DMS = 10

Dimension Mode
Global Const DIMMODESINGLE = 0
Global Const DIMMODECUMULATIVE = 1
Global Const DIMMODEPARTITIONED = 2

Dimension Direction
Global Const DIMALIGNED = 1
Global Const DIMHORIZONTAL = 2
Global Const DIMVERTICAL = 3
Global Const DIMATANANGLE = 4

Dimension Tolerance Type
Global Const DIMNOTOLERANCE = 0
Global Const DIMSTACKEDMINMAX = 1
Global Const DIMSTACKEDVARIANCE = 2
Global Const DIMFIXEDVARIANCE = 3

Dimension Extension Line
Global Const DIMEXTNOSTRETCH = 0
Global Const DIMEXTSTRETCH = 1

Dimension Unit Type (Linear, Radial, or Diameter Dimensions)
Global Const DIMINCHES = 1
Global Const DIMFEET = 2
Global Const DIMMETERS = 3
Global Const DIMMILLIMETERS = 4
Global Const DIMCENTIMETERS = 5
Global Const DIMFTIN = 6

Dimension Unit Type (Angular Dimensions)
Global Const DIMANGLEFORMATDECIMAL = 0
Global Const DIMANGLEFORMATMINUTES = 1

Dimension Arrow Types
Global Const DIMARROWREGNOFILL = 0
Global Const DIMARROWREGFILLED = 1
Global Const DIMARROWREGOPEN = 2
Global Const DIMARROWNOTCHED = 3
Global Const DIMARROWSLASH = 4
Global Const DIMARROWCIRCLENOFILL = 5
Global Const DIMARROWCIRCLEFILL = 6

Dimension Text
Global Const DIMTEXTINLINE = 0
Global Const DIMTEXTABOVELINE = 1
Global Const DIMTEXTFREEFLOAT = 2

Dimension Text Rotation
Global Const DIMTEXTROTATIONALIGNED = 0
Global Const DIMTEXTROTATIONHORIZONTAL = 1

Linear Dimension Line Position
Global Const DIMLINELEFT = 0
Global Const DIMLINERIGHT = 1

Dimension Proximity Mode
Global Const DIMLINENOPROXFIX = 0
Global Const DIMLINEPROXFIX = 1

Dimension Arrow Mode
Global Const DIMARROWNOFLIP = 0
Global Const DIMARROWFLIP = 1

Messaging Codes
Global Const ALERT_APP_ALL = 0
Global Const ALERT_APP_UTOOL_MOUSEDOWN = 1
Global Const ALERT_APP_UTOOL_MOUSEMOVE = 2
Global Const ALERT_APP_UTOOL_ABORT = 4
Global Const ALERT_APP_CMDLINE_CHAR = 8
Global Const ALERT_APP_CLOSE = 16
Global Const ALERT_APP_UTOOL_PENUP = 32
Global Const ALERT_PENUP_CHARCODE = 252
Global Const ALERT_APP_WORLD_CLOSE = 64
Global Const ALERT_WORLD_CLOSE_CHARCODE = 253
Global Const ALERT_APP_UTOOL_ERASERUBBER = 128
Global Const ALERT_UTOOL_ERASERUBBER_CHARCODE = 255
Global Const ALERT_APP_TOOL_COMPLETE = 256
Global Const ALERT_TOOL_COMPLETE_CHARCODE = 254
Global Const ALERT_APP_UTOOL_INIT = 512
Global Const ALERT_UTOOL_INIT_CHARCODE = 251
Global Const ALERT_APP_UTOOL_TERMINATE = 1024
Global Const ALERT_UTOOL_TERMINATE_CHARCODE = 250
Global Const ALERT_APP_FRAME_CLOSE = 2048
Global Const ALERT_FRAME_CLOSE_CHARCODE = 249

Global Const ALERT_APP_FRAME_RESIZE = 4096
Global Const ALERT_FRAME_RESIZE_CHARCODE = 248
Global Const ALERT_APP_ENTITY_ERASED = 8192
Global Const ALERT_APP_ENTITY_ERASED_CHARCODE = 247
Global Const ALERT_APP_ENTITY_SELECT_CHANGE = 16384
Global Const ALERT_APP_ENTITY_SELECT_CHANGE_CHARCODE = 246
Global Const ALERT_APP_ACTIVATE = 32768
Global Const ALERT_APP_ACTIVATE_CHARCODE = 245
Global Const ALERT_APP_DEACTIVATE = 65536
Global Const ALERT_APP_DEACTIVATE_CHARCODE = 244

API calls which use "SymbolIndex", use this parameter for entity section additions
Global Const NONDEFENTITY = -1
Global Const HATCHFILLENTITY = -2

3D Projection Codes
Global Const VIEW3D_FLAT = 0
Global Const VIEW3D_PARALLEL = 1
Global Const VIEW3D_PERSPECTIVE = 2

3D Display Codes
Global Const VIEW3D_WIREFRAME = 0
Global Const VIEW3D_QSHADE = 1

3D View Codes
Global Const CHANGE_VIEW3D_LEFT = 0
Global Const CHANGE_VIEW3D_RIGHT = 1
Global Const CHANGE_VIEW3D_UP = 2
Global Const CHANGE_VIEW3D_DOWN = 3

Import Unit Types
Global Const ACAD_UNIT_INCH = 0
Global Const ACAD_UNIT_FEET = 1
Global Const ACAD_UNIT_MILL = 2
Global Const ACAD_UNIT_CENT = 3
Global Const ACAD_UNIT_METER = 4

Plot/Print Mode
Global Const PRINTMODE = 0
Global Const PLOTMODE = 1

Plot/Print Scale
Global Const FITTOPAPER = 0
Global Const CURRENTVIEW = 1
Global Const USERSCALE = 2

Plot/Print Orientation
Global Const PORTRAITMODE = 0
Global Const LANDSCAPEMODE = 1

Plot/Print Units
Global Const METRICUNITS = 0
Global Const ENGLISHUNITS = 1

Plot COM Settings
Global Const PORTCOM1 = 0

Global Const PORTCOM2 = 1
Global Const PORTCOM3 = 2
Global Const PORTCOM4 = 3
Global Const PORTLPT1 = 4
Global Const PORTLPT2 = 5
Global Const PORTLPT3 = 6
Global Const PORTFILE = 7

Plot BAUD Settings
Global Const BAUD110 = 0
Global Const BAUD300 = 1
Global Const BAUD1200 = 2
Global Const BAUD2400 = 3
Global Const BAUD4800 = 4
Global Const BAUD9600 = 5
Global Const BAUD19200 = 6
Global Const BAUD38400 = 7
Global Const BAUD57600 = 8

Plot DataBit Settings
Global Const DATABITS7 = 0
Global Const DATABITS8 = 1

Plot Parity Settings
Global Const PARITYODD = 0
Global Const PARITYEVEN = 1
Global Const PARITYNONE = 2
Global Const STOPBITS1 = 0
Global Const STOPBITS2 = 1