

Coordinate API

Overview

We have created a Lisp file, CrdAPI.FAS, that contains some simple and easy-to-use functions for accessing, modifying, drawing and selecting any type of coordinate file that Carlson Software supports. The CrdAPI.FAS file is located in the Carlson LSP folder. This collection of functions can help you write your own tools. We call this toolkit the Coordinate File Application Programming Interface, or CRD API.

This document describes the Lisp interface of the CRD API.

Supported Types of Coordinate Files

The following enumerated variables are available for specifying the different types of coordinate files that are supported by the CRD API.

;;; Coordinate File format number

```
(setq
  enCrdTypeUnrecognized -2 ;file exists but is not a recognized type.
  enCrdInvalid          -1 ;not a valid type - used in creating new files.
  enCrdSurvCADDNumeric  0 ;Carlson original; point name is numeric. Extension "crd".
  enCrdSurvCADDAlpha    1 ;SurvCADD sequel; point names can be alphanumeric. Extension
"crd".
  enCrdMemory           2 ;NOT YET IMPLEMENTED.
  enCrdCgAlpha          3 ;C&G alphanumeric. Extension "cgc".
  enCrdCgNumeric        4 ;C&G numeric. Extension "crd".
  enCrdSimplicity       5 ;Simplicity "Sight Survey". Extension is "zak".
  enCrdLDD              6 ;Land Desktop Development. Filename is "points.mdb".
  enCrdCivil3D          7 ;Civil 3D; point name is numeric.
)
```

Working with Points in Coordinate Files

The following functions are for working with Crd Point structures.

Internally, the Crd Point looks like the following list.

```
(name desc x y z)
```

To allow for expansions and changes of the Crd Point in the future, please use only the following Crd Point functions for setting and accessing Crd Points.

CrdPointInit

A Crd Point structure will be returned that can be passed into CrdAddPoint, etc. *name* and *desc* must both be strings.

```
(defun CrdPointInit( x y z name desc)
  (list name desc x y z)
)
```

CrdPointSetName

Sets the name of the Crd Point structure. The Crd Point is returned.

```
(defun CrdPointSetName( point name)
  (cons name (cdr point))
)
```

CrdPointGetName

Gets the name of the Crd Point structure.

```
(defun CrdPointGetName( point)
  (car point)
)
```

CrdPointSetDesc

Sets the description of the Crd Point structure. The Crd Point is returned.

```
(defun CrdPointSetDesc( point desc)
  (cons (car point) (cons desc (cddr point)))
)
```

CrdPointGetDesc

Gets the description of the Crd Point structure.

```
(defun CrdPointGetDesc( point)
  (cadr point)
)
```

CrdPointSetXYZ

Sets the x, y, z values of the Crd Point structure given a list of (x y z). The Crd Point is returned.

```
(defun CrdPointSetXYZ( point xyz)
  (append (list (car point) (cadr point)) xyz)
)
```

CrdPointGetXYZ

Gets the x, y, z values of the Crd Point structure as a list of (x y z).

```
(defun CrdPointGetXYZ( point)
  (cddr point)
)
```

Working with the Coordinate File

The following functions are for working with coordinate files.

CrdOpen

Opens a coordinate file with the given filename. Only one coordinate file can be open at a time through the Lisp interface. If CrdOpen is called twice, the first crd file is closed automatically.

filename -- The file name of the coordinate file with required extension (.crd, .cgc, .mdb).

bOverwrite -- If 1 (integer), then any existing file of the same name will be destroyed. Use nil to select default value which is 0.

crdType -- See enCrdType above. This specifies the type of CRD file to create if the given name does not already exist. Use nil to select the default value which is enCrdSurvCADDAlpha.

Returns 1 (integer) if successful and 0 if there was some kind of failure. Call CrdGetLastError() to get the text message of the error.

```
(defun CrdOpen( filename bOverwrite crdType)
```

```
(cf:crdapi "CrdOpen" filename bOverwrite crdType)
)
```

CrdClose

Closes the current Crd file, if it is open.

```
(defun CrdClose()
  (cf:crdapi "CrdClose")
)
```

CrdAddPoint

Adds the given Crd Point to the current open Crd File, overwriting any point with the same name that may already exist. Returns 1 (integer) if successful and 0 otherwise. Call CrdGetLastError() to get text message of error.

```
(defun CrdAddPoint( point)
  (cf:crdapi "CrdAddPoint" point)
)
```

CrdBegin

Starts a traversal of all the points of the Crd File by resetting internal state so that CrdNext will start from beginning of file.

range -- if not nil, then it must be a Carlson-style string of point names suggested by the following examples: "1-5,6,12-19" or "all". If nil, that is the same as all. The points will be returned on successive calls to CrdNext() in the order listed in the range.

```
(defun CrdBegin( range)
  (cf:crdapi "CrdBegin" range)
)
```

CrdNext

Gets the next point in the Crd File. Returns a Crd Point if successful and nil if there are no more points or some other failure.

```
(defun CrdNext()
  (cf:crdapi "CrdNext")
)
```

CrdGetPoint

Gets the named point from the current open Crd File. A Crd Point is returned if successful and nil is returned if the point could not be found. Call CrdGetLastError() to get text message of error.

```
(defun CrdGetPoint( name)
  (cf:crdapi "CrdGetPoint" name)
)
```

CrdRemovePoint

Removes the named point from the current open Crd File. Returns 1 (integer) if successful and 0 otherwise. Call CrdGetLastError() to get text message of error.

```
(defun CrdRemovePoint( name)
  (cf:crdapi "CrdRemovePoint" name)
)
```

CrdRemovePoints

Removes the given range of points (a string) from the current open Crd File. For example, the range could be "1-5,7,10-12".

```
(defun CrdRemovePoints( range)
  (cf:crdapi "CrdRemovePoints" range)
)
```

```
)
```

CrdTestPointName

Returns nil if the point name is valid for the current Crd type. Otherwise, returns an error string describing the error.

```
(defun CrdTestPointName( name)
  (cf:crdapi "CrdTestPointName" name)
)
```

CrdTestPoint

Returns nil if the Crd Point is valid for the current Crd type. Otherwise, returns an error string describing the error.

```
(defun CrdTestPoint( point)
  (cf:crdapi "CrdTestPoint" point)
)
```

CrdGetLastError

Returns the error string of the result of the last operation.

```
(defun CrdGetLastError()
  (cf:crdapi "CrdGetLastError")
)
```

CrdDeleteFile

Deletes the data file(s) underlying the Crd File and closes the Crd File. May not work if the original filename was not a full path including drive letter. Returns 1 (integer) if successful and 0 otherwise. Call CrdGetLastError() to get text message of error.

```
(defun CrdDeleteFile()
  (cf:crdapi "CrdDeleteFile")
)
```

GetCrdType

Returns the crd type of the current open Crd File.

```
(defun GetCrdType()
  (cf:crdapi "GetCrdType")
)
```

GetCrdTypeName

Returns the official name of the given crd type.

```
(defun GetCrdTypeName( enCRDType)
  (cf:crdapi "GetCrdTypeName" enCRDType)
)
```

CrdDraw

Accepts an argument list to indicate a set of points to be drawn, which at a minimum takes the form '("Points" "<points designator>").

```
(defun CrdDraw(arglist)
  (eval (append '(cf:crdapi "CrdDraw") arglist)))
)
```

The syntax for additional optional parameters which may be included in arglist (indicated by "[]") is as follows:

```
(CrdDraw '("Points" "<points designator>"
  ["SymbolName" "<symbol block name>"]
  ["SymbolXScale" <X scale>])
```

```

["SymbolYScale" <Y scale>]
["SymbolRotation" "<symbol rotation in AUNITS>"]
["SymbolLayer" "<symbol layer name>"]
["BlockName" "<attribute block name>"]
["BlockXScale" <X scale>]
["BlockYScale" <Y scale>]
["BlockRotation" "<attribute block rotation in AUNITS>"]
["BlockNameLayer" "<layer for point name>"]
["BlockElevationLayer" "<layer for point elevation>"]
["BlockDescriptionLayer" "<layer for point description>"]
["PointLayer" "<layer for POINT>"]
)
)

```

CrdSelect

Accepts an argument list to indicate a set of points to be selected, which at a minimum takes the form nil to select all points.

```

(defun CrdSelect(arglist)
  (eval (append '(cf:crdapi "CrdSelect") arglist)))
)

```

The syntax for additional optional parameters which may be included in arglist (indicated by "[]") is as follows:

```

(CrdSelect '(
in drawing                                ; Select all points
      ["Points" "<points designator>"]    ; Select all points
in <point designator>
      ["Prompt" "<prompt for points>"]    ; Interactive
(spatial) select points in drawing
      ["P0" point "P1" point]            ; Select all points
in window having corners P0,P1
)
)

```

Note that all optional parameters within "[]" are mutually exclusive. The function returns the selected points as a list of points, each of which is itself a sublist indicating the point identifier and its associated drawing entity name.

Example Code

```

;;; Examples on how to use.
(CrdOpen "junk2.crd" nil nil)
(princ (strcat "Created a file of type: " (GetCrdTypeName (GetCrdType)) ".\n"))

; create a couple of points in memory.
; This first point is at coordinate 100,101,10.5 and has the point name
; of 1012 and the description of IP.
(setq point1 (CrdPointInit 100 101 10.5 "1012" "IP"))
(setq point2 point1)
(setq point2 (CrdPointSetXYZ point2 '(105 101 10.5)))
(setq point2 (CrdPointSetName point2 "3"))
(setq point3 (CrdPointSetName point2 "7"))

; adding a point and checking for errors.
(setq bSuccess (CrdAddPoint point1))
(if (= bSuccess 0)
  (princ (strcat "Error on point '" (CrdPointGetName point1) "' : " (CrdGetLastError))))
".\n")

```

```

; adding points without checking for errors.
(CrdAddPoint point2)
(CrdAddPoint point3)

; now print out the points 1 through 5
(princ "Listing the point names.\n")
(CrdBegin "1-5") ; nil or "all" would select all the points.
(while (setq point (CrdNext))
  (princ (strcat (CrdPointGetName point) " " (CrdPointGetDesc point) "\n")))
)
;
; Drawing points.
; Draw all points in the drawing using defaults. Argument list '("Points" "<points
designator>") is required as a minimum.
(CrdDraw '("Points" "ALL"))
;
; Syntax for additional optional parameters indicated by "[" is as follows:
; (CrdDraw '("Points" "<points designator>"
;           ["SymbolName" "<symbol block name>"]
;           ["SymbolXScale" <X scale>]
;           ["SymbolYScale" <Y scale>]
;           ["SymbolRotation" "<symbol rotation in AUNITS>"]
;           ["SymbolLayer" "<symbol layer name>"]
;           ["BlockName" "<attribute block name>"]
;           ["BlockXScale" <X scale>]
;           ["BlockYScale" <Y scale>]
;           ["BlockRotation" "<attribute block rotation in AUNITS>"]
;           ["BlockNameLayer" "<layer for point name>"]
;           ["BlockElevationLayer" "<layer for point elevation>"]
;           ["BlockDescriptionLayer" "<layer for point description>"]
;           ["PointLayer" "<layer for POINT>"]]))
;
; Selecting points.
; Select all points in the drawing
(CrdSelect nil)
;
; Select all points in window
(CrdSelect '("P0" (setq p0 (getpoint "First Corner: ")) "P1" (getcorner p0 "Opposite
corner: ")))
;
; Syntax for additional optional parameters indicated by "[" is as follows.
; (CrdSelect '(["Points" "<points designator>"] ; Select all points in
<point designator>
["Prompt" "<prompt for points>"] ; Interactive (spatial)
select points in drawing
["P0" point "P1" point] ; Select all points in
window
; All parameters within [] are optional and are mutually exclusive.
;
;
(CrdDeleteFile) ;the only proper way to delete the file(s). C&G uses two files, for
example.
(CrdClose) ;not necessary because of CrdDeleteFile, but doesn't hurt.
(princ "---the end---\n")

```