Reference Point

Volume 3, Number 2 Technical Resources for DataCAD ® Spring, 1993

3-D Capabilities in DataCAD: An Overview

DataCAD provides a powerful set of 3-D tools that you can use for a variety of drawing tasks. In the form of an overview, this article describes the fundamental 3-D capabilities of DataCAD. It assumes your general facility with 2-D drawing. Where appropriate, it discusses menu selections in some depth. The article describes the process by which an elevation drawing is generated from a plan drawing utilizing the 3-D (Z) attributes of 2-D entities. It describes basic 3-D entities and 3-D viewing controls. It illustrates a process for creating a 3-D model and some techniques for efficient 3-D modeling. It explores modeling considerations for generating hidden-line images and Velocity renderings. It presents a discussion of system configuration issues for running Velocity and explores the use of the Velocity macro and the Velocity rendering software.

From Plan to Elevation

By carefully controlling the structure of a plan drawing, viewing it in the 3-D portion of DataCAD, and saving an image of the elevation view, you can quickly construct an elevation drawing. Figure 1 illustrates ground floor and second floor plan drawings of the exterior walls of a building:

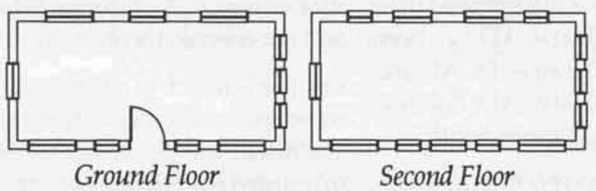

Figure 1

This example shows the two plans, drawn in four separate layers, "on top of" each other. Two layers are used for the walls, one for each floor. The door and windows are drawn in their own layers, one for each floor; the door and windows are cut into the walls of the appropriate floors with the LyrSrch toggle *on*. In constructing the ground floor walls, the Z Base has been set to 0, Z Height to 8'-0".

The door and windows of the ground floor plan are drawn with Z Base and Z Height settings (Z,z) at 0 and 8'-0" and their Sill and Head Height settings are controlled in the DoorSwng and Windows menus. Similarly, the second floor plan is constructed with Z Base set to 8'-0" and Z Height set to 16'-0". The windows cut into the second floor walls are drawn with Z settings left at 8'-0" and 16"-0".

The Sill Height and Head Height settings in the Windows menu are set to values *relative to the currently set Z Base*. Thus, when the Z Base is set to 8'-0" and a window is drawn with a Sill Height of 3'-4" and a Head Height of 6'-8", the window is drawn with *absolute* Z values of 11'-4" and 14'-8".

Since careful control of these settings is important to the success of the operation, it is critical that you understand where you are located in Z space at any moment. For this reason, it is highly recommended that you keep the display of Z information *on* at all times (Settings, Show Z).

With all four of the wall and door/window layers turned on and a fifth, empty, layer active, you enter the 3-D Views menu (V). When you select FrntElev from the Elevtion menu, a front elevation of the drawing is displayed. Walls, windows, and the door are all displayed in their correct orientation [it may be necessary to invoke WindowIn (/), Extents (F1) to display the full elevation].

A 3-D extrusion of 2-D elements (sometimes referred to as 2½D) comprises the displayed elevation. Since these elements are wire-frame in nature, having no surface characteristics, hidden-line removal on them will not produce an effective elevation image. The simplest means of "capturing" the elevation view is by using SaveImag, restricting the number of lines "captured" by using Clip Cube.

With the front elevation still displayed, exit to the 3D Views menu and select Clip Cube (S7). Then you choose New Cube (F2). The display changes to plan view (Ortho, in the parlance of the 3-D Views menu). You establish Z-Min (F4) and Z-Max (F5) settings for the Clip Cube describing a range to be *included* in the Clip Cube. In this example, values of -1'-0" and 17'-0" are used. You respond to the

continued on page 3

CONTENTS

Guidelines for Graphics Card Purchasers

Many DataCAD users are currently considering the purchase of a new graphics card, either as a part of a new system or as an upgrade to an existing one. The wealth of new cards on the market is largely driven by the proliferation of Microsoft Windows. Many of you are faced with questions about these cards' compatibility with DataCAD.

In considering a specific card, you should inquire about the software drivers supplied with the card. If the manufacturer provides a driver for DataCAD, then the card should work properly. If they do *not* supply a DataCAD driver, you should inquire about **VESA emulation or 8514A compatibility**. You can configure DataCAD to either of these standards.

An article in the Spring, 1992 issue of *Reference Point* details the process of configuring a system for use with a VESA-compliant graphics card. Basically, the card must first be put into VESA mode through the use of a vendor-supplied software driver. You then configure DataCAD with its own VESA driver: either VESA256.EXE or VESA16.EXE. You should place both the vendor-supplied VESA emulation driver and the DataCAD VESA driver in the High DOS memory area for optimum performance in DataCAD. See *Reference Point*, Summer, 1992 for a discussion of system optimization; the configuration detailed in that article utilizes the VESA drivers. An article in this issue shows a typical system configuration for Velocity, using VESA drivers.

The process for utilizing a card that is 8514A compatible is more straightforward. This type of card should be *register level compatible* with 8514 emulation and can be driven directly with the Cadkey supplied driver: ATC8514.EXE, which should be loaded to the High DOS memory area.

Cadkey technical support personnel are familiar with a number of the new graphics cards on the market and the means by which you can configure them for DataCAD. An inquiry to them might prove helpful in the process of selecting an appropriate graphics card. Further, in purchasing a card that you anticipate configuring with either of these emulation modes, your purchase should be made conditional upon successful configuration for DataCAD.

From the Editor

According to Cadkey, the recent price promotion for DataCAD (detailed in the flyer enclosed with the Winter, 1993 issue of *Reference Point*) has met with great success. Consequently, at the request of dealers, the promotion has been extended through the end of June. Many existing users have renewed their maintenance agreements, and DataCAD has been adopted at many new sites. Because DCAL and Velocity are now bundled with DataCAD, *Reference Point* will be covering their use in more depth.

A new column on DCAL, written by Bill D'Amico, begins in this issue. Long-time DataCAD users will be familiar with Bill's contributions to the DataCAD community. He works for a DataCAD dealer, Corporate Network Systems of Yarmouth, Maine, and is the author of a number of commercial DCAL macros: **DC Sprint, Blocker**, and **TouchUp**. Under the guise of "Dr. DataCAD," he was a regular contributor to *WindowIn on DataCAD*. He brings to these pages a wealth of experience with DataCAD, its users, and DCAL programming.

Bill's column will provide to those of you who are unfamiliar with software programming the information necessary to begin the process of creating DCAL macros. The intent is to help you to understand the concepts and procedures involved in this process and to assist you in understanding the documentation provided with the product.

The Cadkey bulletin board now has a new section for DCAL. Source code and compiled macros that Bill discusses in his column will be posted there, available for downloading. Cadkey will provide source code for routines, as well. Also, it is hoped that other DCAL programmers might share code and/or macros through the bulletin board.

On the subject of electronic communication, the recent provision of modems and CompuServe mail addresses for DataCAD personnel at Cadkey has prompted me to finally join that service. The process of initiating a membership is extremely simple (contact CompuServe at 800-848-8199). The services provided are wide ranging. My CompuServe mail address and those for various Cadkey folks are provided on page 12 of this issue.

I can recommend very highly the software "front-end" packages that CompuServe has developed. CIM (CompuServe Information Manager) is a menu-driven DOS program that facilitates the process of logging on to and navigating through CompuServe services. A Windows version, WIN CIM, has recently been released; you get the same basic tools as the DOS version, but the GUI interface makes it *much* easier to use. Depending upon your preference for DOS or Windows, either one of these is well worth the $49.95 price, which includes a $25 credit to the new member's account.

continued from page 1

message line prompts by identifying two points that define the diagonal on the CLIP CUBE. Toggle *on* CLIPON (F1). When the screen refreshes, only those entities contained *within* the defined Clip Cube are drawn; see Figure 2. Note that the extents of the Clip Cube are displayed in dashed lines. A portion of the ground floor door and the second floor window above it are seen.

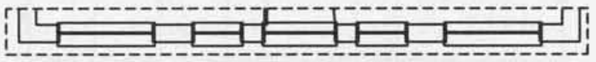
Figure 2

Exit the CLIP CUBE menu. You return to the 3D VIEWS menu in FRONT ELEVATION view, with only those entities contained within the CLIP CUBE displayed. You select SAVEIMAG (S5) to "capture" the elevation. It is saved to the active layer by choosing ACTVLYR (F1) as the destination (remember that a fifth, empty, layer is active). Figure 3 illustrates the saved image in ORTHO (plan) view, with only that layer on.

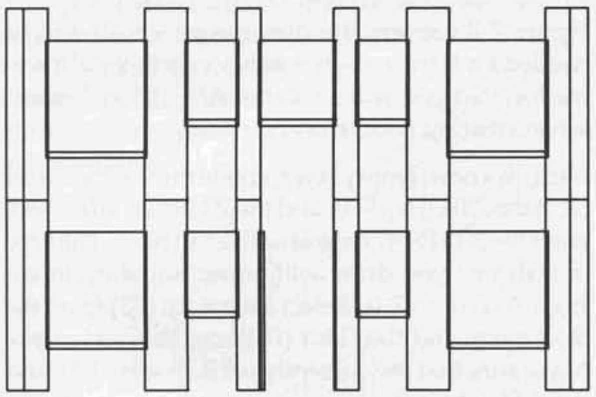
Figure 3

Note the varying window sill heights on both floors; this was accomplished at the point of creating the windows, in plan, by changing the SILLHGT setting in WINDOWS. The HEADHGT setting for all windows and the door was left at 6'-8". IDENTIFY (I) and MEASURES (Alt+X) operations performed on the saved image indicate that the elevation is dimensionally accurate and that all entities have been drawn with a Z BASE of 0 and a Z HEIGHT of 0.

Note also the effect of having created the plan with CUTOUT (F4 in both DOORSWNG and WINDOWS) turned on. At each cutout for a door or window, the wall has been cut from top to bottom. Because of the way that SAVEIMAG works, at every occurrence of a line in this image, there are, in fact, at least two lines, one overlaid on top of another. The process of cleaning this up could be time consuming and tedious. It is far simpler to trace (with LYRSNAP *on*), in another layer, a "clean" version of this image.

Figure 4 illustrates the "clean" version of the elevation base generated from the saved image. The amount of information traced from the SAVEIMAG layer is a function of the means by which it will be later detailed. In the example illustrated, the window sills have not been traced; only the "rough opening" has been copied. Because this elevation base will be used in later steps involving the use of 3-D entities, it has been drawn with lines having a Z BASE of 0 and a Z HEIGHT of 0.

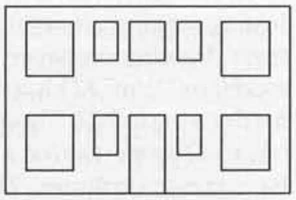
Figure 4

One of the more tedious operations in the development of an elevation drawing is the construction of accurate windows, particularly double-hung windows with divided lites. John Hitch has written a DCAL macro, DHW40, that provides parametric controls to facilitate this process.

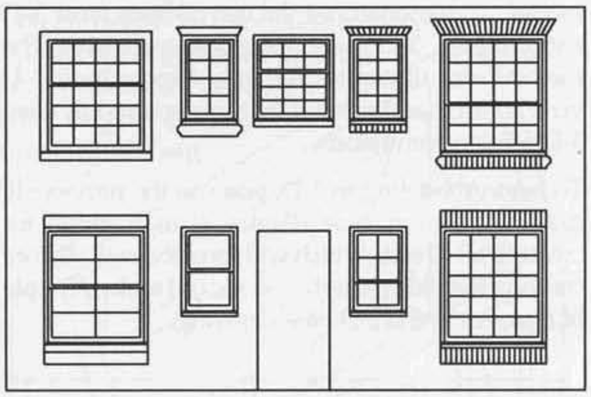
Figure 5

Figure 5 illustrates the elevation base drawing after you have added 2-D windows to it by using DHW40. Each window has been drawn with differing menu settings to illustrate the range of settings available. Again, these windows have been drawn on their own layer to maintain the integrity of the elevation base for later 3-D modeling procedures.

To use the macro, you define X and Y sash dimensions, select either Casement or Double Hung as the window type, and specify the number of lites in terms of X and Y divisions (options change depending upon window type selected). You can set dimensions for: casing, backband, sill, top rail, meeting rail (double hung only), bottom rail. You can set the color in which the window is drawn. You can specify Subsill and Lintel types (including "none") and set their sizes. Optionally, you can place descriptive text (not shown) with each window; menus allow you to control text size, color and aspect.

DHW40
John Hitch
Hitch & Associates
3309 Childers St.
Raleigh, NC 27612
(919) 782-4373

Building a 3-D Model

Just as making an architectural model is different from drawing on paper, the construction of a 3-D model in DataCAD is a different process than that for drawing in 2-D. Though DataCAD's 2-D entities have 3-D properties (i.e. a Z component), they do not have *planar* attributes. Therefore, the usefulness of 2-D entities in 3-D modeling is limited. Ultimately, the "output" from a 3-D model is either an image generated by the hidden-line process or a rendered Velocity image. While the hidden-line routine can process 2-D entities, its results are far more satisfactory when it operates on true 3-D entities. Velocity *requires* 3-D entities for its operation.

A wide range of tools for the creation of a 3-D model are available in the 3-D portion of DataCAD. They include three basic types: 3-D view controls, tools for creating 3-D entities, and 3-D editing tools. All of these are well documented in the DataCAD manual and will not be dwelt upon here. Rather, this article presents a simple method for the creation of a 3-D model. It demonstrates the use of three basic 3-D entity types: 3-D lines, polygons, and slabs. The method also illustrates the use of some basic 3-D view controls and some of the more commonly used 3-D editing commands.

To begin modeling in 3-D, you use the previously drawn elevation base (Figure 4) as a guide for creating a 3-D slab, which will form the wall. Before creating the slab, though, you should make a couple of changes to the 2-D base drawing.

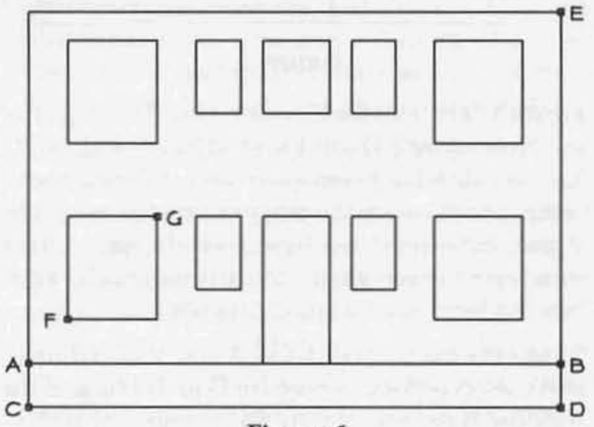

Figure 6

Figure 6 shows the modified elevation base drawing. You create a new line (C-D) by copying the original base line (A-B) some distance (2'-0" in this example) and then trim the vertical end lines to this new line. You determine the distance to copy based on the grade condition around the building which you are modeling. Since very few buildings sit on absolutely flat sites, and even then, the elevation of the first floor will be above grade, you want to extend the walls of the model below grade, which will be modeled separately in a later step.

Another thing that you may want to do at this point is to make careful note of the critical dimensions of the window penetrations that you will be creating.

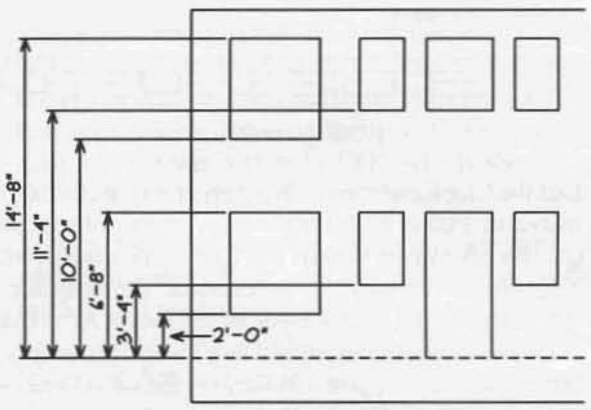

Figure 7

Figure 7 illustrates the dimensions which will be needed for later steps; it corresponds to the Z information that you will set in the AEC_Model macro when creating windows and doors.

Next, in a new, empty layer, create the wall. First set (Z,z) the Z Base to -1'-0" and the Z Height to 0. Then, enter the 3-D Entity menu and select Slab. The type of slab that you draw will be rectangular, drawn from Z -1'-0" to Z 0. Select Rectngle (F3) from the Slab menu and Bas/Hgt (F3) from the next menu. Make sure that the currently set Z Base is -1'-0" and the Z Height is 0.

At the prompt: "Enter first point of rectangular slab," snap (LYR snap should be *on*) to a corner point on the 2-D base drawing (point C in Figure 6). At the prompt: "Enter second point of rectangular slab," snap to point E, Figure 6. Repeat the process for each of the windows and the door; snapping to points F and G for one of the windows, for instance.

Next, the window slabs and the door slab need to be processed as voids in the wall slab. To do so, in the Slab menu, select Voids (F5). At the prompt: "Select master polygon or slab to process voids," pick the wall slab by clicking on it with the left mouse button. The slab will be displayed in light grey dashed lines. With Area (F3) toggled as the selection method, at the prompts: "Select first corner of area to make into a void" and "Select second point of area to make into a void," draw an area box around the window and door slabs. They will all be displayed in light grey, dashed lines. Make sure that AddVoid (F8) is toggled on and exit the menu. An Identify performed on the slab will indicate that there is one slab containing ten voids.

Next, you need to draw a 3-D Line at Z 0 on the face of the slab, indicating on the outside of the model the first floor grade elevation. To do so, you enter the 3-D Line menu (F1) from 3-D ENTITY, toggle Z-HGHT (F4) to indicate that both ends of the line will be located at the currently set Z Height (0). Draw the 3-D Line by snapping to points on the 2-D base drawing (A and B, Figure 6). For purposes of clarity in later steps, you should make the color of this line different from the color of the slab or a different line type; it is illustrated in this example as a dashed line.

Next, **Rotate** the slab into position, using the ROTATE menu in the 3-D section of DataCAD. In this example, the slab is rotated 90° about the X axis to bring it into position. Figure 8 illustrates this; it shows a hidden-line image of the slab, with voids, rotated up from the 2-D base drawing (at Z 0,0). The center point about which the rotation is performed is selected by snapping to either end of the 3-D "base" line. 3-D Rotate defaults to a Z location of 0, which is appropriate for this example.

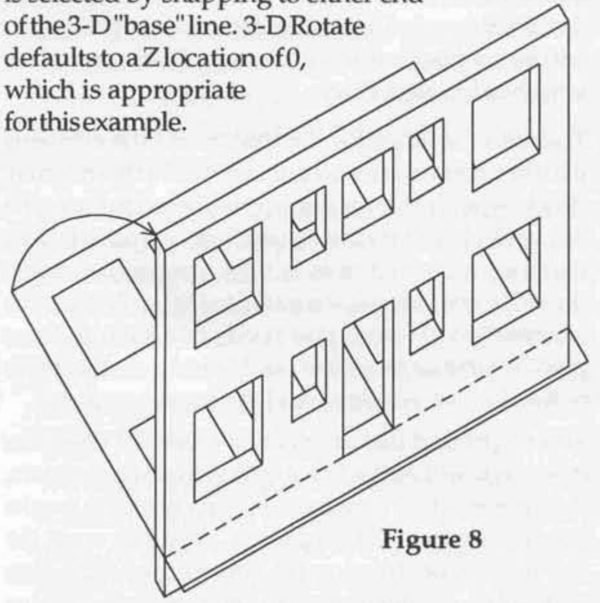
Figure 8

The Right Hand Rule governs 3-D ROTATE (and 2-D ROTATE, as well). This is an important concept to understand. Imagine that you are holding a pencil in the palm of your right hand with your fingers wrapped around it. The point of the pencil is aimed down, toward the heel of your hand and your thumb is aimed up the shaft, toward the eraser. With the point of the pencil at the "center of rotation" and the axis of the pencil tilted to the "axis of rotation," rotate the pencil by moving your thumb toward the knuckles of your hand. This is rotation in a positive direction for whatever axis you are representing. In the example illustrated here, you can see that the rotation selected is +90° along the X axis.

Why not use Vertical Slabs for the wall and the voids, setting the Z Base and Z Height as appropriate and avoid the necessity of rotating horizontal slabs into place? There are two reasons: First, the horizontal technique allows you to trace from a developed elevation created in 2-D. 2-D editing tools (notably CLEANUP) provide to you the ability to draw complex elements, like the exact location of a roof pitch, and to accurately place window penetrations. You may first work out complex elevations in 2-D and then trace that work in another layer with 3-D entities.

Second, accurate processing of voids is dependent upon the coordination of *void* slabs with the *master* slab. DataCAD's rule is that ***void and master slabs must be coplanar*** (in a horizontal configuration, all of the slabs must have the same Z Base and Z Height) and all of ***the slabs must have their reference faces in the same plane.*** When drawing a wall with voids using vertical slabs, it can prove to be difficult to coordinate these two important factors. Especially if you are new to 3-D modeling in DataCAD, it is recommended that you use the methods described above as you develop your modeling skills.

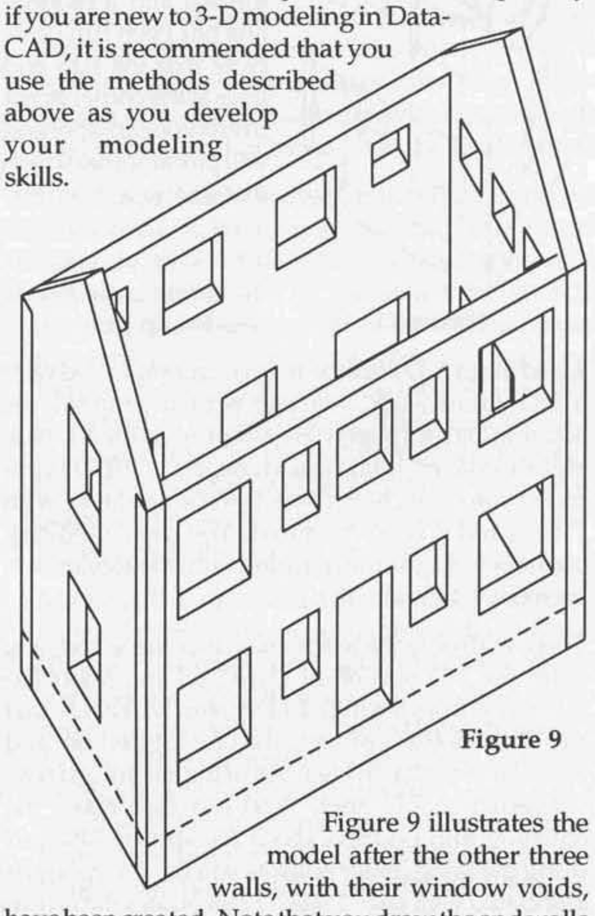
Figure 9

Figure 9 illustrates the model after the other three walls, with their window voids, have been created. Note that you draw the end walls (at the gable ends) with horizontal slabs, rather than with rectangular slabs so that they slope with the roof pitch (a slab may be comprised of as many as 36 vertices). In all other respects, you draw this type of wall slab in the same manner as described above.

Figure 9 also illustrates a problem that commonly occurs with 3-D models; because the wall slabs overlap at the corners, the hidden-line process has drawn lines from the thickness of the slabs on the face of the intersecting slabs. Cleaning these up can be tedious. There is a way to avoid the problem, at the cost of a minor inaccuracy in the model.

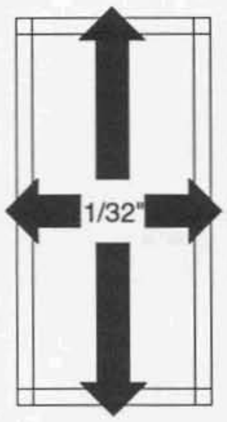

Figure 10

Once you get the model to the point illustrated in Figure 9 (i.e. all of the wall slabs are properly constructed), move each of the slabs (and the 3-D Lines associated with each of them) 1/32" away from the center of the model as illustrated in Figure 10. You use 1/32" as it is the smallest distance that you may specify in DataCAD. In a rectangular model, the footprint has grown 1/16" in the X and Y directions as a result.

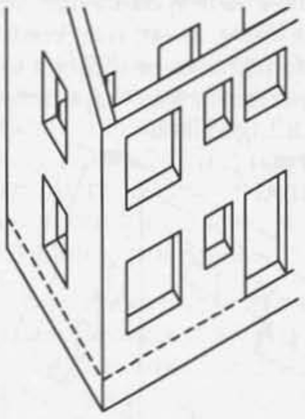

Figure 11

Figure 11 shows the corner of the model after the move has been performed and a hidden-line has been run on it. Note that the slab end lines at the corner *do not* overlay one another and will plot at some (small) distance away from one another. This may serve a positive purpose, as the corner indication is heavied up a bit.

Hatching a 3-D surface may be accomplished with a little bit of work. Starting with an empty layer active in ORTHO (plan) view, you turn on the 2-D base elevation layer. Using it as a base, in the (2-D) HATCH menu you create hatch lines in the active layer, with Z BASE and Z HEIGHT set to 0. You use 2-D editing controls to adjust the hatching until the elevation is correctly hatched.

Then, with only the hatch lines layer turned on, you enter the 3-D portion of DataCAD and select EXPLODE (F9) from the initial 3-D menu. Make sure that the TOLINES (F8) option is the one toggled on, and select all of the hatch lines. All of the hatching is now converted to 3-D lines. You can then rotate the hatching into position about the appropriate axis using the same center point as you used to rotate its associated wall slab. Figure 12 illustrates the resulting "hatched" slab with voids.

Note that **TouchUp**, a third-party macro, incorporates parametric tools that perform a similar function; see the Summer, 1992 issue for a more complete description of the use of this macro.

TouchUp
Bill D'Amico
Corporate Network Systems
P.O. Box 965
30A Rte. 1 Suite 1
Yarmouth, ME 04096
(207) 846-0772

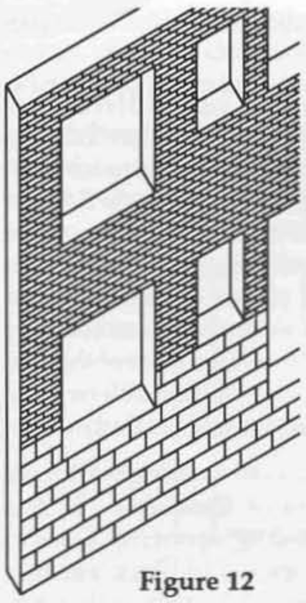

Figure 12

Using AEC_Model: Next, you use the AEC_Model macro to construct window and door elements in the 3-D model. As in earlier steps, you will find that it is generally easier to work in plan (ORTHO) view to create elements in the model and to then verify the results by viewing in a PARALLEL, OBLIQUE, or PERSPECTIVE view. To begin, work with a new, empty layer as the active one. Turn on the layer containing the wall slab to which you will be adding doors or windows and, if you are not already there, set your 3-D View to ORTHO. In both the Door and Window sections of AEC_Model, the first thing that you tell the macro is whether you will be creating entities in PLAN or ELEVATION view; you should select PLAN.

The menu settings for the macro and the elements that they control are well documented in the manual. Working with PLAN entry, you will need to know the SILL and HEAD HEIGHT for each door and window that you construct; this is why you earlier noted these dimensions, as illustrated in Figure 7. With this information at hand, you need only snap to three points at the appropriate void to indicate the limits of the door or window, and the macro creates it.

You might find that an efficient means of using the macro is to utilize the FORM option to set parameters, then place a door or window, and check the results by viewing in a 3-D projection. You can erase the last-placed construction (<), correct the parameters selected by either entering the appropriate menu or by opening the FORM again, and re-place the door or window. Once you have established settings that you will utilize over again, you can save them to file so that they can be loaded to the macro easily.

Use RoofIt to construct a roof for the model. The use of this macro is detailed in the Summer, 1992 issue of *Reference Point*.

Grade elements around the model can be constructed in a variety of ways. Most simply, INCLINED POLYGONS provide a means of creating pathways and sloping grade conditions at the edge of the building. They serve to mask the portion of the model that extends below grade.

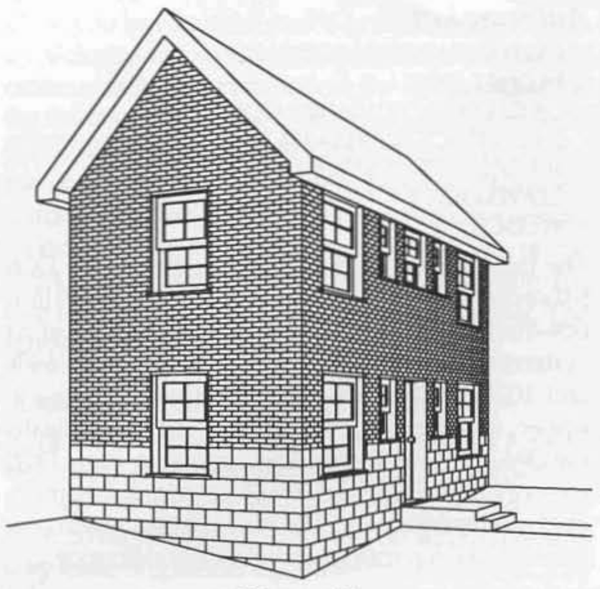
Figure 13

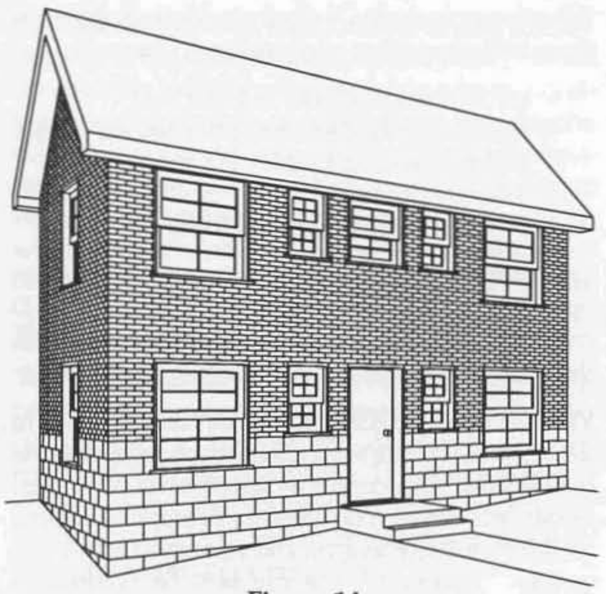
Figure 14

Figures 13 and 14 illustrate perspective views of the model constructed in the previous steps.

3-D Views are an important component in the process of creating a 3-D model. You should experiment with the controls that are available and read the sections in the manual that cover them. A couple of important notes:

While the "globe" that is displayed when PARALLEL is selected can be very useful in establishing a view for checking the accuracy of modelled elements, it distorts the height of the model (ELEVATION views are a sub-set of PARALLEL which do not distort the Z component, however). You should use OBLIQUE views to generate axonometrics of your model, as Z values remain "true."

The SAVE VIEW function accesses the 3-D GOTOVIEW function, which is similar to the 2-D GOTOVIEW menu. It can save up to 999 views, each of which can have a (different) CLIPCUBE associated with it. Saved 3-D views are useful in the process of constructing a model and in processing hidden-lines; they are essential to the process of rendering with Velocity.

The CLIP CUBE can be used to limit the amount of information displayed to the screen at any given time. It is useful for limiting the amount of visual clutter that you have to deal with while modeling.

The **HIDE** function, however, does not support CLIP CUBE. The only imaging function that supports it is SAVIMAG, as described earlier. Two important notes about HIDE: Before running a HIDE, make sure that SAVIMAG (F3) is turned *on*. There is nothing worse than tying up your computer for an extended period of time while it processes an image, only to find that you cannot save that work. Second, toggle *on* CROPIMAG (S2) any time that lines extend beyond the drawing window in a perspective view. When CROPIMAG is off, you may be left with lots of lines that need to be erased or trimmed in the final image.

Use AutoHide in ViewMaster to batch process a series of hidden-line images overnight. You can select the saved 3-D VIEWS that you wish to process and save the images to layers within the active drawing file or to external layer files. When you save to layer files, make sure that when you specify the PATH to which they are sent, you include a backslash at the end of the path name: C:\MTEC\LYR\ If the path name does not include the final backslash, the layer files written by the AUTOHIDE process will overwrite one another, and, in the end, you will be left with only the last one processed.

Manage the drawing file carefully. From the material presented here, you can see that 3-D modeling involves the construction of many discreet components. You should organize these components in separate layers, grouped by each face of the model. In the example illustrated here, each face of the building has separate layers for: the wall slab, the window and door assemblies, and the hatching. A (thirteenth) layer contains the roof and a separate (fourteenth) one is used for the site elements.

Because Hidden-line processing time and Velocity rendering time are largely a function of the total number of entities to be processed, you should be able to turn off layers containing elements that are not visible in the view to be processed.

Rendering with Velocity

When you produce a Velocity rendering, there are three basic steps that you have to go through:

- constructing a model in DataCAD
- translating the data that describes the model and its associated views to a format that Velocity can read
- processing that data in Velocity

The components of the model on which Velocity can operate are 3-D entities; it will not recognize 2-D entities. You build a model for Velocity rendering in the manner described in earlier sections of this issue.

VELOCITY.DCX, a DCAL macro normally installed to the C:\MTEC\DCX directory, is used to accomplish the conversion. You access it by selecting VELOCITY from the MACROS menu. You receive a prompt for a name for the file to be generated, and you have a NEWPATH option. Normally, the file will be written to C:\VELOCITY\DRN, a subdirectory that is created by the INSTALL routine for Velocity. While the macro is running, a message appears: "Writing rendering file C:\VELOCITY\DRN\FILENAME.DRN."

Note that the macro writes to the .DRN file *only those layers that are turned on* at the point of executing the macro. It writes to the file all 3-D GOTOVIEW saved views. However, Velocity does not support layer switching by the views; for views of the model that require different combinations of layers to be turned on, you must create separate .DRN files.

After running the macro, exit DataCAD and enter Velocity. For optimal performance in Velocity, you should cold boot your computer in a system configuration different from the one used for DataCAD.

Configuring for Velocity

Articles in the Winter and Summer, 1992 issues of *Reference Point* discuss system requirements for DataCAD. You should refer to them for a detailed examination of how DataCAD works and how you should configure your system to maximize its performance. Similarly, Velocity has specialized requirements for its running. You need to create versions of CONFIG.SYS and AUTOEXEC.BAT that are particular to running Velocity. The following are typical versions of CONFIG.SYS and AUTOEXEC.BAT for Velocity using MS DOS 5.0 drivers:

Config.sys

```
DEVICE=C:\DOS\HIMEM.SYS
DOS=HIGH,UMB
DEVICE=C:\DOS\EMM386.EXE 6960 RAM
FILES=30
BUFFERS=15
STACKS=0,0
```

Autoexec.bat

```
PATH C:;C:\DOS;C:\COMFILES
PROMPT $P$G
LH C:\STAR\UTIL\VMODE.COM VESA
LH C:\MTEC\DRV\VESA256.EXE
SET DC_GDT=VESA256,60,3,0,0,1
CD\VELOCITY
VELOCITY
```

The first line in CONFIG.SYS installs HIMEM.SYS, DOS 5.0's extended memory manager. In the second line, DOS=HIGH loads a large portion of the DOS operating system to the high DOS memory area (between 640k and 1024k). The UMB statement enables access to upper memory blocks. The third line installs EMM386.EXE, which provides access to high DOS memory and configures 6960k of *extended* memory as ***expanded*** memory. The FILES, BUFFERS, and STACKS lines establish parameters for DOS's execution.

This example is for a system configured with 8 megs of physical RAM. The number that you enter on the EMM386.EXE line will depend upon the amount of RAM installed on your system. The goal for Velocity is to ***configure as much RAM as possible as expanded memory***. A RAM disk provides no speed enhancement to Velocity. A disk cache would improve some aspects of its performance, but it would do so at the expense of the amount of expanded memory directly available to Velocity, a much more critical factor in its overall performance.

In AUTOEXEC.BAT, note that the Velocity directory (normally C:\VELOCITY) is *not* included in the path statement. As in the examples presented in the DataCAD configuration articles, the illustrated system uses a Diamond SpeedStar graphics card, which is driven in DataCAD and Velocity with Cadkey's VESA driver: VESA256.EXE. As with the DataCAD configuration, the card must first be put into VESA mode; this is accomplished with Diamond's driver: VMODE.COM; it is installed to the high DOS memory area by including the LH (load high) statement.

Next Cadkey's VESA driver is installed, again to the high DOS memory area. For Velocity, you can use the same graphics driver that you use for DataCAD *if it is a 256 color driver.* The SET DC_GDT= line establishes a DOS environment variable that is read by Velocity to set the display resolution. The documentation for Velocity includes specific instructions for each of the available graphics drivers. Consult it for details about you own graphics card.

See the Fall, 1992 issue of *Reference Point* for two batch files that will help you save different versions of CONFIG.SYS and AUTOEXEC.BAT, swap them into the root directory, and reboot your system.

Running Velocity

Once you have established a system configuration for Velocity and can run the program, it is strongly recommended that you follow the steps outlined in the third section of the Velocity documentation, *A Quick Tour.* This will provide to you an overview of the modeling and rendering process. You will quickly see that, of the steps involved in running Velocity, most are straightforward. Two of the steps, though, are relatively complex: assigning surface attributes and specifying the image format. The following sections discuss them.

Assigning surface attributes

In the main menu of Velocity, the ATTRIBUTE ASSIGNMENT option allows you to designate surface attributes for entities in the model that you have selected for rendering. The way you accomplish this is by assigning rendering attributes on a (DataCAD) color by color basis. F2 (OPTIONS) opens a lower level menu which allows you a number of controls over the assignment process.

If, for instance, you want all blue entities in all layers to be rendered as MARBLE, you may assign this attribute globally by selecting ALL LAYERS. If you want blue entities on one layer to be MARBLE and blue entities on another layer to be TIN, you may enter each of the layers separately and assign the values to blue entities in each layer according to your requirements. A DataCAD model can include as many as 1,000 layers, and each layer can contain entities drawn in as many as 15 colors. Hence, in Velocity you can specify attributes for as many as 15,000 discreet entity identifiers.

You quickly understand the need to manage the original DataCAD model in a manner that facilitates attribute assignment in Velocity. The more consistently that you use the same color to draw entities of the same ultimate material designation, the less work you will have to do in Velocity. For example, you might use Red for all slabs that will be rendered as BRICK and limit the location of those slabs to only a few layers. Then, finding them in Velocity and assigning BRICK to Red entities will be simplified. Just as color assignment and the correlation of color to pen assignment is an important office standard for 2-D production drawings, standard layer/color assignment for frequently used materials in 3-D models serves to facilitate their rendering.

Cadkey's 3-D macros, AEC_Model, RoofIt, and 3-DStairs, all provide the ability to set the color for components of the assemblies that they model. They also allow you to save settings to file. For frequently used window types, you can establish a color standard for the window components, saving the settings to a file created from within AEC_Model. Similarly, in Velocity, you can create a corresponding grouping of attribute assignments, keyed to the color standard, and save it to file. Once you have saved this file, you can load it into Velocity to establish the attribute assignments for the model on which you are working. This provides to you some considerable time savings in the rendering process.

Image format

You can specify the output of Velocity's rendering process to a number of file formats. Typically, you can produce a .2RN file and display it at *screen resolution*. You can display any .2RN image to screen at the resolution that you specified in the set DC_GDT= line of AUTOEXEC.BAT. For many purposes, this is an adequate means of presentation.

You may wish, however, to export a true color (16.7 million colors) version of the rendering to a file format that can be read by a film recorder. You should refer to the Velocity documentation for information about converting .6RN files to .SCD format.

Another means of processing Velocity output is to export a rendering file to "paint" software, in which you can touch-up and enhance the rendered Velocity image. Do not confuse this with the PAINT option under IMAGE RESOLUTION in the VIEW SPECIFICATION menu, in which you generate a file that can be read by the Hewlett-Packard PaintJet printer.

Most paint programs today run under MS Windows and support a variety of file formats. Unfortunately, .6RN files cannot be directly read by these packages. To export a Velocity rendering to a paint package, you need to convert a .6rn file to a format that can be read by other software.

One way to do this is to use VEL2TGA.EXE, a conversion utility provided with Velocity, to convert a .6RN file to .TGA (Targa) format (see the manual pp. 5-19 to 5-21). Many paint packages can directly read Targa files. If your paint software does not read .TGA files, do not despair; there are a number of shareware conversion utilities that can convert graphics files to a number of commonly used formats.

Upgraded versions of two conversion files are available on the Cadkey Bulletin Board. Look for:
VEL2TGA.EXE
TGA2VEL.EXE

Cheapware lists a couple of these utilities. It also has a number of other items which can be very useful to you for 3-D modeling and rendering in Velocity. 3-D symbols for modeling in DataCAD are available. Alternate Velocity brick texture files and "sky" backgrounds are listed, as are a couple of 3-D "color wheels" which enhance the process of color selection in Velocity.

CHEAPWARE

Evan Shu
Shu Associates
10 Thacher St.
Suite 114
Boston, MA 02113
(617) 367-9622

DCAL WORKSHOP

What is DCAL? DCAL (DataCAD Applications Language) is a programming language which allows you to write 'macros' which users can access and run from within DataCAD. DCAL macros can be programmed to perform nearly any drawing manipulation task and greatly extend DataCAD's capabilities.

Who Can Use DCAL? Programming DataCAD effectively with DCAL requires both a thorough knowledge of DataCAD from the user's point of view, as well as at least a beginner's level of programming principles using a language such as BASIC or, even better, PASCAL. DCAL programming is very PASCAL-like in structure and flow, so elementary concepts such as functions, procedures, and variable types need to be firmly grasped in order to write even the simplest of DCAL macros.

It is *not* the case that anyone who uses DataCAD - even a DataCAD 'guru' - can write DCAL macros. It is also *not* the case that any programmer can write good, easy-to-use DCAL macros. DCAL is a powerful tool; it is also a dangerous one. It is powerful because it can directly read and write data in a DataCAD drawing file; it is dangerous because a DCAL macro can corrupt a DataCAD drawing file if it is not carefully written. If I haven't scared you off yet, let's get to it...

Setting Up to Program in DCAL You must do a few things before you can program in DCAL. First, you must install the DCAL language programs themselves from the disks supplied by Cadkey. After that, I recommend that you include the directory into which you installed DCAL (usually C:\DCAL) in the PATH statement of AUTOEXEC.BAT. You can then write and edit programs in any directory, and compile and link them by just typing the DCAL compile and link commands. This way you can more easily separate your work into a number of directories. To program in DCAL, you must be able to edit a text file. Use DOS 5.0's EDIT, or use your word processor to edit and save your program source files as plain ASCII text files.

Our First Example: 'Hello World' Many language tutorial manuals start by showing the simplest program anyone would want to write, typically, the famous 'Hello World' example. The following macro displays the words 'Hello World' on the message line below DataCAD's drawing window:

```
PROGRAM hello;
BEGIN
wrterr('Hello World');
END hello.
```

HELLO.DCS

This tiny DCAL program shows the basic structure of a program file. The program name is '**hello**'; program execution begins at the word '**BEGIN**' and ends at the word '**END**'. The only actual program code is the line '**wrterr('Hello World');**'. The **wrterr** statement invokes the procedure 'wrterr', a built-in DCAL subprogram that takes the string that you give it in the parentheses and single quotation marks ('Hello World') and displays it on what DCAL calls the error line of the DataCAD display. Note: *program code lines must always end in a semicolon.*

To write this program, open a new file with a text editor, type in the four command lines listed, then save the file with a .DCS (for DCAL Source) extension. Call it HELLO.DCS. You must process a source file through two steps to convert it to an executable DCAL macro file (.DCX) for DataCAD to run. The steps are: *compiling* and *linking*. Our first step, compiling HELLO.DCS, is accomplished from the DOS command line by typing: **DCC HELLO** DCC is the DCAL Compiler. If DCC reports no errors in the compile step, it creates the file: HELLO.DCO. Then, HELLO.DCO is *linked* to form HELLO.DCX by typing: **DCL HELLO;**

DCL is the DCAL Linker. Note that you must include the semicolon. DCL can link many source files into one large macro file, so the semicolon on the command line tells the linker that this is the last file (and in this simple case the only one) that is needed to build the DCX macro executable file.

If DCL does not report any link errors, you now have a HELLO.DCX file that can be run from DataCAD's MACROS menu. Just copy HELLO.DCX to your 'macros' subdirectory (usually C:\MTEC\DCX), run DataCAD, and pick HELLO from the MACROS menu. 'Hello World' displays on the 'error line' in the prompt area at the bottom of the screen.

```
PROGRAM example1;
VAR
a : integer;
b : integer;
c : integer;
d : integer;
e : integer;
f : real;
g : real;
h : real;
s : string(80);
m : boolean;
BEGIN
a := 10;
b := 3;
c := a + b;
d := a * b;
e := a / b;
f := 10.0;
g := 3.0;
h := f / g;
s := 'Hello World';
m := true;
END example1.
```

EXAMPLE1.DCS

Variables and Variable Types: The HELLO macro is so simple that it really shows us nothing of the elementary concepts of DCAL: procedures, functions, and variable types. Let us now consider the small macro program called EXAMPLE1, which introduces variable types.

The 'VAR' section of EXAMPLE1 declares the variables that the program is going to use *before* they are ever used. This is a requirement of DCAL which some other languages, notably BASIC, do not have. Any and all variables (or procedures or functions) that will be used in a program must have their 'type' declared in a 'VAR' section before the program's 'BEGIN' statement. Declaring a variable means indicating what type of information that variable will hold. Variable names can be up to 20 characters in length, and must start with a letter (a-z).

Note: Upper and lower case is ignored in all DCAL statements, so you can use them interchangeably. I prefer to keep DCAL statements that define program sections and program flow, such as PROGRAM, IF, THEN, BEGIN, END, PROCEDURE, etc. as all capitals, as it makes programs easier to read. Program sections may be separated by extra carriage returns or tabbed indents to make those areas easier to find. DCAL ignores this white space when the program is compiled.

The EXAMPLE1 macro shows how variables are declared and used in DCAL. Variables **a** through **e** are declared as integers; they can be assigned any value from -32767 to 32767, but only integer values (no decimal point). We assign to the variable **a** the value of 10 and to **b** the value of 3 with the first two program statements. Then we illustrate addition, multiplication, and division of integers with the next three. ':=' is called the *assignment operator*; it is used to assign the value of the expression to its right to the variable on its left. So **c** will take on the value 13, and **d** will be 30.

The value of **e**, however, will be 3, *not* 3.3333333 as you would expect if you did 10 / 3 on your calculator. This is because we declared that **e** can only hold an *integer* value, so 10 / 3 will only yield the integer portion of the result, which is, of course, 3. So **e** will have a value of 3. Doing the same division operation using the variables **f**, **g**, and **h** will result in **h**'s value being set to 3.33333333. Why? Because **f**, **g**, and **h** have been declared as **real** variables, which can have decimal points. i.e. fractional values.

Variable **s** is declared as a string variable that can be up to 80 characters long. We assign **s** equal to the value 'Hello World'. Integers, reals, and strings are three of the more commonly used types, as is a 'boolean', which can have the value of 'true' or 'false'. The EXAMPLE1 macro assigns the value of *true* to the variable **m**. Chapter 8 - 'DCAL Built In Types' in the DCAL manual goes into much more detail about these and other variable types. You can also define your own types in a TYPE section of your program, but let's not get ahead of ourselves.

DataCAD also has many variables already declared and built into DCAL, all of which have to do with various drawing settings. These are outlined in Chapter 9 - 'DCAL Built In Variables' in the DCAL manual. As the user works in DataCAD and changes various drawing settings, the values of these built-in variables are what are actually being changed.

Variable types are one of the cornerstones of any programming language. Familiarize yourself with DCAL's built in **types** by looking over Chapter 8 in the DCAL manual, then the built in **variables** in Chapter 9. If you are very familiar with DataCAD as a user, you should recognize most or all of the built in variables in Chapter 9.

Procedures and Functions: Since good programming practice dictates that any program of any significant size must be broken down into subprograms that interact with one another, a serious programming language provides the ability to do so. DCAL has two mechanisms for creating such subprograms - calling them, like PASCAL, **'procedures'** and **'functions'**.

A procedure is like a little program all itself. Like the built in DCAL **'wrterr'** procedure used in the HELLO macro, you can write procedures that can be called upon to perform a specific task and/or produce a result based on data that is passed to it. In the HELLO macro, we knew that we wanted to display a message on the screen, and we knew what the message was. Since **wrterr** knows how to do that, we simply 'called' the wrterr procedure and gave it the message string that we wanted the user to see. **A function** is identical to a procedure except that functions *return a value* of a particular type. Hence, you must have a variable ready to receive the result of a function, for example:

```
r := 4.0;
s := sqrt(r);
```

Here **r** and **s** are both variables of type **real**. The variable **r** is assigned the value 4.0, then the square root function is asked to operate on the value of **r**, and **s** is assigned that value: 2.0. DCAL has some 500 built-in procedures and functions which will do very sophisticated things for you that you would otherwise have to program yourself. We will get into writing our own procedures and functions as we get further into DCAL; for now, our next example will just call a few of DCAL's many built in procedures.

```
PROGRAM datetime;
#include '/dcal/inc_misc.inc'
VAR
year, month, day, hours : integer;
minutes, seconds, hundreths : integer;
tmp_str : str8;
outstr  : str80;
am_or_pm : string(2);
BEGIN
readclock(year, month, day, hours, minutes, seconds, hundreths);
cvintst(month,outstr);
strcat(outstr,'/');
cvintst(day,tmp_str);
strcat(outstr,tmp_str);
strcat(outstr,'/');
cvintst(year,tmp_str);
strcat(outstr,tmp_str);
strcat(outstr,' — ');
IF hours > 12 THEN
  cvintst(hours-12,tmp_str);
  am_or_pm := 'PM';
ELSE
  cvintst(hours,tmp_str);
  am_or_pm := 'AM';
END;
strcat(outstr,tmp_str);
strcat(outstr,':');
cvintst(minutes,tmp_str);
IF minutes < 10 THEN
  strcat(outstr,'0');
END;
strcat(outstr,tmp_str);
strcat(outstr,am_or_pm);
wrterr(outstr);
END datetime.
```

DATETIME.DCS

'DataCAD, what time is it?' Have you ever been working along in DataCAD, not wearing your watch, and wished you could ask DataCAD to tell you the time? Or have you wanted to automatically place the current time and/or date on your drawing? In either case, you probably concluded that DataCAD simply can't tell time even though you know that your computer can. Actually, DataCAD *can* tell time, it simply has no command on any menu for you to ask it to. This is a perfect example of why a DCAL macro is useful. We can use DCAL to perform a task (displaying date and time) that no standard DataCAD command will do.

DCAL contains a procedure called **'readclock'**; it reads the PC's clock and assigns values to integer variables. It allows us to derive the date and time and display them to the user. As well as the 'readclock' procedure, the DATETIME macro introduces two other procedures: **'strcat'** and **'cvintst'**.

The 'readclock' procedure in DCAL does the actual work of getting the current date and time information from the computer. It stores that information in the variables enclosed in the parentheses (hours, minutes, etc..). Each of these is just an integer value which would look something like: 2 19 1993 18 3. To pretty it up, we use further program logic to read and convert these numbers into a nice text string, in a form like: 2/19/1993 — 6:03 PM

The **wrterr** procedure will only accept a *single* string or string variable for display. We must convert the integers into strings and put the strings together to make one string before we ask **wrterr** to display that string. This is done by combining sequences of the **cvintst** procedure, which converts each integer value into a string, and the **strcat** procedure, which takes a string and adds it to the end of another string. Both procedures are documented in the DCAL manual, Chapter 11; look there for more information.

I use the 8 character string variable **tmp_str** over and over as I convert each integer of the date and time to a string. I am building the 80 character string variable **outstr** as I go along and I only need **tmp_str** as a temporary holding place for the string value of each converted integer. I also add nice punctuation as I build the **outstr** variable, because I want it to be nicely formatted for the user to read.

I throw in an IF..THEN to decide if it is 'AM' or 'PM' to get our feet wet with conditional logic in DCAL. I also set the value of the 2 character string variable **am_or_pm** appropriately; it will get added, by the last program statement, to the string just before it is displayed to the user, using the **wrterr** procedure.

The only other new concept in this example is the line: **#include '/dcal/inc/_misc.inc'** The definition of the **readclock** procedure is actually external to DCAL's compiler; we tell the compiler to include the definitions that it finds in the file '_misc.inc' (in DCAL's include file directory) to use the **readclock** procedure that is defined there. I use forward slashes instead of DOS's usual backslashes here, because backslashes are used for another purpose in DCAL.

Summary: At its simplest, you can use DCAL to teach DataCAD a few small tricks that it will not already do for you, like this issue's DATETIME.DCX. Like any powerful language, DCAL can do more. In fact, there is very little you can imagine that you would like DataCAD to do that can't be done with a DCAL macro. However, programming DataCAD through DCAL is different from and more complex than simply using DataCAD. My goal in this column is to instruct you in the basics of DCAL, illustrated by sample macros that are both instructional and functional.

Bill D'Amico is the author of Blocker, DC Sprint, and TouchUp. He can be reached on CompuServe at: 70033,3072

CONTACT

Cadkey:	(203) 298-8888
Technical Support:	ext. 8060
Tech Support FAX:	(203) 298-6404
Cadkey Bulletin Board:	(203) 298-6405 *(8 bits, no parity, 1 stop bit)*
Lou Bodnar: *A/E/C Marketing*	ext. 6425
Malcolm Davies: *President, CEO*	ext. 8060 cs: 73417,2373
Clay Rogers: *Quality Assurance*	ext. 8060
Frank Simpson: *Marketing Publications*	ext. 6443 cs: 72640,3560
Mark White: *A/E/C Product Management*	ext. 6455 cs: 72640,3570

PUBLICATION INFORMATION

Reference Point is published quarterly by:
Cadkey, Inc. 4 Griffin Road North Windsor, CT 06095-1511

Additional copies, changes in address, and other business communications:
Tel: (203) 298-8888; Fax: (203) 298-6401

Managing Editor:	Frank Simpson
Editor:	Philip Hart
Editorial Review Committee:	Lou Bodnar Clay Rogers Eric Smith Mark White
Editorial Communication:	Philip Hart 9 High Street Brunswick, ME 04011 (207) 729-0907 cs: 71563,1330