P-CAD DBX Programmer's Interface
User Guide and Reference

This is the User Guide and Reference Manual for the P-CAD Database Exchange (DBX) Programmer's Interface.

The P-CAD DBX interface allows you to create custom reports, CAM output, and utility functions by writing user programs that retrieve design data from an active P-CAD PCB or Schematic design session. The DBX interface also allows you to modify an P-CAD P-CAD PCB or Schematic design by placing new design objects or modifying properties of existing objects. Similarly, DBX provides the capability to read and write specific data from/to P-CAD Library Manager and P-CAD Library Executive. This User Guide and Reference includes general user information, programming hints, definition of the functional interface and its data structures, and some basic examples.

This document is formatted so that it is suitable for both viewing on your screen and printing hardcopies. To print this document, choose the File Print command while this document is loaded in Microsoft Word or WordPad.

The first-time user should read Section 1 for an overview of P-CAD DBX and its capabilities. Then, when you are ready to write your first user program, skim Sections 2 through 7 to learn the key aspects and features of the interface, what data is available, and how to manipulate it. Section 8 provides an overview of project development, with Section 8.4 and 8.5 including Visual Basic and C++ specifics. Examples are provided in Section 8.6, and online. Referring to these examples, and copying the examples to use as a starting point is highly recommended.

Important Information about P-CAD 2004 Version 18.00

The supplied DBX32.H and DBX32.BAS source files have been updated. For your existing applications you need to recompile and relink your source programs using the supplied DBX32.H (for C or C++ programs) or DBX32.BAS (for Visual Basic programs).

Layer Stackup Support (PCB): Support has been added for the new layer stackup data. This includes one new data structure TLayerStackup. Contained within are four char arrays; layerName, layerMaterial, layerThickness, layerDialectricConstant. With the data structure are two functions TGetFirstLayerStackup and TGetNextLayerStackup for fetching the data.

Layer and Sheet Suppor (PCB/Schematic): Two new functions, TAddLayer and TAddSheet have been added to allow creation of layers/sheets in a design.

Pad and Via Style Support: Four new functions were added to support retrieving pad and via style data. They are: TGetFirstPadStyle, TGetNextPadStyle, TGetFirstViaStyle, and TGetNextViaStyle.

Copyright © 2006 Altium Limited

Contents

1 Introduction to DBX

1.1 What is P-CAD DBX?

P-CAD DBX is a programmatic interface that allows you to retrieve or modify design data from an active P-CAD PCB or Schematic design using your own Visual Basic, C, or C++ programs. Data is retrieved from the design by making P-CAD DBX library function calls. Design data is modified by making DBX library function calls with PCB design data items as input. Similarly, there is a DBX interface to the Component Library Manager and Library Executive.

By using the P-CAD DBX interface, you can create interactive design query functions, custom report generators, or custom design file output. You can also create custom applications that make modifications to the active design based on your own specific design needs. Since the interface communicates directly with an active P-CAD design session, your DBX user programs can perform custom database queries and modifications as you design, without the need to generate intermediate files that your program must then parse, interpret, and modify. Your DBX user program can extract or modify as much, or as little, detailed information as is needed.

The P-CAD DBX Programmer's Interface includes a functional interface (a "function library"), pre-defined data structures, support files and utilities, examples, and a user guide.

This interface may be called by any Windows application that can reference Windows Dynamic Link Libraries (DLLs). Developing DBX applications using either the Visual Basic or Visual C++ development environment is described in this manual. It is recommended that programmers less familiar with developing Windows applications use either Visual Basic or the Windows application development environment with which you are already familar.

By using one of the Windows development environments available and the supplied code samples provided with P-CAD DBX, novice and experienced programmers alike will find this a convenient, yet extremely flexible tool to enhance your development process. This guide provides an overview that describes the use of two prevalent Windows development environments: Visual Basic and Visual C++.

1.2 Document Conventions

P-CAD supplied functions and datatype names are **bolded.** Function parameters, when referred to within the body of the text of this manual, are shown in *italics.* Filenames are shown in CAPITAL letters. User variables and structure member data are shown in lower case, with all words capitalized except the first word.

1.3 Data Overview

The P-CAD DBX Programmer's Interface provides direct access to all key PCB, Schematic, and Library data. DBX programs may retrieve and modify PCB and Schematic design information about the current design: layers, sheets, nets, components, pads, vias, styles, symbols, component patterns, connections, routed

traces, polygons, pours, text items, netclasses, class to class rules, rooms, grids and attributes. DBX programs may retrieve and modify Library Manager or Library Executive library objects, including components, symbols, and patterns.

Predefined data structures and data types are provided for easy retrieval and access to the specific data defining these items. These data structures are provided as Visual Basic User Defined Types for Visual Basic user programs, and as C structures for C and C++ user programs. An example is the **TNet** structure defined in Visual Basic as the User Defined Structure TNet:

```
Type TNet
    netId     as Long
    netName   as String*DBX_MAX_NAME_LEN
    nodeCount as Long;
    length    as Long;
    isPlane   as Long;
TNet;
```

and in C as a structure Typedef TNet:

```
typedef struct
{
    long      netId;
    char      netName[DBX_MAX_NAME_LEN];
    long      nodeCount;
    long      length;
    long      isPlane;
} TNet;
```

TArc, TAttribute, TClassToClass, TComponent, TDesign, Tgrid, TLine, TLayer, TNetClass, TPad, TPadViaShape, TPadViaStyle, TPin, TPoly, TPour, TRoom, TSymbol, TText, TTextStyle, and TVia are defined similarly. An overview of the data and functions to retrieve the data returned in these structures is provided in Sections 3 through 7 of this document. Note that many data items and function calls are shared between PCB, Schematic and Library Manager, with Library Manager and Library Executive being identical. This helps simplify the interface, and makes it possible for some DBX programs to be written such that they access multiple P-CAD products without any changes or a recompile. The actual structure definitions, types, and sizes are provided for reference in Appendix B.

User programs reference the data in the predefined structures by declaring a DBX item and examining or assigning values to the individual data elements within the structure. An example of declaring a TNet item and accessing TNet data is given in Section 1.5.

1.4 Functional Overview

The P-CAD DBX interface provides the ability to extract or modify complete design information, yet is still easy to use and flexible enough to satisfy a variety of user needs. Function input arguments are simple, with consistent, predefined data output formats returned by each function. Each function call returns a status value indicating success or failure, and if failure, what type of failure.

The P-CAD DBX interface retrieves data in a top-down fashion. In other words, high-level functions provide high level data, with subsequent calls available to access more

detailed and in-depth data. The interface additionally uses a "get first", "get next" approach to return design data with a variable number of items. This approach eliminates the need to define large arrays and data structures within your program to hold *potentially* large amounts of data, where the size of the array would be dependent on the size of the design being accessed.

For example, functions returning P-CAD PCB net data, net items, and net nodes are:

```
TGetFirstNet
TGetNextNet
TGetNetById
TGetNetByName

TGetFirstNetItem
TGetNextNetItem

TGetFirstNetNode
TGetNextNetNode
```

Design modification and updates are performed by function calls that have a DBX item as input to the function. Input to these functions are the same DBX items that are returned by the data extraction function calls.

For example, functions to modify an existing PCB component are:

```
TFlipComponent
TMoveComponent
TRotateComponent
THighlightComponent
TUnHighlightComponent
TModifyComponent
TDeleteComponent
```

and to place or delete a component:

```
TPlaceComponent
TDeleteComponent
```

The next sections provide an example of how these functions and DBX data types may be used.

1.5 Retrieving Net Data - A Brief Example

Retrieving data about an P-CAD PCB net is an example demonstrating this type of interface and how the interface is used.

Function **TGetFirstNet** and **TGetNextNet** return general net data for a P-CAD PCB net including net length, number of nodes, and the net name. The output data is returned in a P-CAD DBX data structure called **TNet**. If you were interested in the length of the net, for example, your program would declare a DBX item *aNet* as a **TNet** structure, and examine *aNet.length*, a member of the DBX item *aNet*.

In addition, if your program needed to retrieve data on the nodes which define the net, a call maybe made to **TGetFirstNetNode** to retrieve the first node in the net, followed by

calls to **TGetNextNetNode** for each subsequent node in the net, until either all nodes have been retrieved or you have located a particular node of interest. In this way a user program may be written to access all nodes in a net (or all nets, for that matter) without knowing in advance what the largest net to be handled is expected.

The following examples find the length of the longest net in the current P-CAD PCB design.

Using Visual Basic:

```
Dim myNet as TNet
Dim maxNetLength as Long

status = TGetFirstNet(tContext,myNet)
maxNetLength = myNet.length

Do While (tStatus == DBX_OK)
    tStatus = TGetNextNet(tContext,myNet)
    if (myNet.length > maxNetLength) then
        maxNetLength = myNet.length
    endif
Loop
```

Using C:

```
TNet myNet;
long maxNetLength;

status = TGetFirstNet(tContext,myNet);
maxNetLength = myNet.length;

while (tStatus == DBX_OK)
{
    tStatus = TGetNextNet(tContext,myNet);
    if (myNet.length > maxNetLength)
    {
        maxNetLength = myNet.length;
    }
}
```

Variables *tStatus* and *tContext* are explained in Section 2.2, 2.9 and 2.10.

Complete examples are provided in Section 8.6, and in the online sample programs included with your P-CAD DBX installation files. The online sample programs are described in Section 10 of this manual.

1.6 Modifying Components - A Brief Example

Retrieving all of the components in a PCB design and flipping each component that meets a certain criteria is an example of modifying active design data.

Functions **TGetFirstComponet** and **TGetNextComponent** return general component data for an P-CAD PCB component including component type, refdes, pattern name, number of pins, library, location and orientation. The output data is returned in a P-CAD DBX data structure called **TComponent.** If you were interested in the component type,

for example, your program would declare a DBX item *aComponent* as a **TComponent** structure, and examine *aComponent.compType*, a member of the DBX item *aComponent*.

In addition, if your program needed to retrieve and modify the component pads, calls to **GetFirstCompPad** and **GetNextCompPad**, followed by the appropriate function call to modify each pad would update the component's pads.

The following example retrieves all of the design components, and flips the component if the component is of type 7400.

Using Visual Basic:

```
Dim myComponent as TComponent

status = TGetFirstComponent(tContext,myComponent)

Do While (tStatus == DBX_OK)

    if (trim(myComponent.compType) = "7400") then
      status = TFlipComponent(tContext,myComponent)
      if (status <> DBX_OK) then
         go to ErrorHandler
      endif

  tStatus = TGetNextComponent(tContext,myComponent)
Loop
```

Using C or C++:

```
TComponent myComponent;

status = TGetFirstComponent(tContext,myComponent);

while (tStatus == DBX_OK)
{
    if (stricmp(myComponent.compType,"7400")==0)
    {
    status = TFlipComponent(tContext,myComponent);
    if (status != DBX_OK)
    {
       break;
    }
  tStatus = TGetNextComponent(tContext,myComponent)
    }
}
```

Variables *tStatus* and *tContext* are explained in Sections 2.2, 2.9, and 2.10.

Complete examples are provided in Section 8.6, and in the online sample programs included with your P-CAD DBX installation files. The online sample programs are described in Section 10 of this manual.

2 Interface Summary

This section provides an overview of some of the fundamental aspects and features of the P-CAD DBX interface. Specific details on item definitions, sizes, and values are listed in Appendix B. Function syntax and parameters are described in Appendix C. Mnemonic constants mentioned in this section are listed in Appendix A. All of the information presented in Appendices A-C may be found online in the file DBX32.H.

2.1 Function Naming Conventions

All P-CAD DBX functions are prefaced with the capital letter 'T'. This convention helps to distinguish P-CAD-supplied DBX functions from your user function and program names. In addition, the first letter of each word in the function name is capitalized. Visual Basic is not case sensitive, so this capitalization convention may be followed or ignored based on your own preference. For 'C' programs, however, the case must be observed or the functions will be listed as "undeclared" at compile time.

2.2 Status and Error Returns

All P-CAD DBX functions return a completion status value. This status is an integer value. For ease of use, each value is represented by a mnemonic constant declared in DBX32.H and DBX32.BAS. All status constants are prefaced by "DBX_", (for example, DBX_ITEM_NOT_FOUND) and are grouped according to completion status and severity of the error.

The value of DBX_OK is zero. All other return values are non-zero, beginning at 32001. This convention will allow non-zero status checking by 'C' programs, and direct use of the Visual Basic error handling tools without conflicting with pre-defined Visual Basic error return values.

Note that many of these non-zero status values returned are not necessarily errors. They may indicate normal operating conditions, like DBX_NO_MORE_ITEMS, for example.

A complete list of all status values may be found in DBX32.H or DBX32.BAS. A global variable *tStatus* has also been declared in both DBX32.H and DBX32.BAS. This variable may be used directly by your user program, as is shown in the following examples.

2.3 Stateful Functions

Any function prefaced by "GetNext" indicates that it is a stateful function, retrieving the next sequential data item from the list of related data items being retrieved. Layer and Net functions return items in order of layer number and Net ID. Other functions return items in the order the items are stored in the P-CAD database. Except for layers and nets, this order is design dependent and should not be assumed to be consistent from DBX program session to session.

Function calls which set the internal program state to a particular item in the sequence are those prefaced by "GetFirst". A GetFirst function must be called prior to calling any GetNext function or the error status DBX_GETFIRST_NOT_CALLED will be returned.

GetNext functions can be called after a GetFirst function call and will return the next sequential item after the GetFirst or GetBy item returned.

2.4 Declaring User Data Using P-CAD DBX-Supplied Structures

There are two P-CAD supplied files providing structure definitions you may use directly within your user program to define local variables and structures. The files are DBX32.H and DBX32.BAS. DBX32.H includes C structure *typedefs*, and is used by user programs written in C or C++. DBX32.BAS includes Visual Basic *Declare Type* definitions, and is used by user programs written in Visual Basic. For example, to declare a local DBX item *myNetData* to be used with the **TGetFirstNet** function, you would use:

```
Dim myNetData as TNet
```

in the declarations section of a Visual Basic program, or

```
TNet myNetData;
```

in the declarations section of a C program.

In either language, the local DBX item and its member data are accessed and named identically. As in the previous example, to access the net length of a net after a Get*Net call, you would use the syntax:

```
myNetLength = myNetData.length;   (no semi-colon in VB)
```

As with other supported datatypes, you can declare arrays of DBX items. Continuing with the net data example, to declare an array of 100 **TNet** items, you would use the following syntax:

```
Dim mynetarray as TNet(100)       (from Visual Basic)

TNet mynetarray[100];             (from C)
```

2.5 The P-CAD DBX TItem

The P-CAD DBX structure **TItem** is used to create a special type of DBX item. It is used to declare a DBX item that can receive data from functions that may return one of many types of design items. For example, **TGetFirstLayerItem** and **TGetNextLayerItem** functions accessing a PCB design may return an arc, component, line, pad, point, polygon, copper pour, text, or a via on a single call. The P-CAD DBX **TItem** structure eliminates the need for your program to know in advance what type of item is about to be returned from a function by defining a structure which can represent each of these different item types. It is also defined so that the structure elements have easy to reference data elements with consistent meanings regardless of the type of item it contains.

```
Dim myItem As TItem               (for Visual Basic)
TItem myItem;                     (for C)
```

After extracting a DBX item, you can access the **TItem** data directly. To examine the

radius of an arc returned by **TGetNextLayerItem**, for example, you could use the following statement:

```
status = TGetNextLayerItem(tContext, myItem)
arcRadius = myItem.arc.radius
```

Alternatively, the item returned can be copied directly to a local DBX item, in this case an arc item, and the arc specific information retrieved from the local DBX item *myArc* by:

```
Dim myItem As TItem         (for Visual Basic)
Dim myArc as TArc
status = TGetNextLayerItem(tContext, myItem)
myArc = myItem.arc
arcRadius = myArc.radius
```

Either approach will generate identical results. The former is preferred if only a select amount of information is needed, while the latter is generally more convenient if you will be using the data from this particular arc in many places or referencing the data many times, since it requires less typing.

The items returned by these functions are:

- arc
- attribute
- bus
- ClassToClass
- CompPin
- component
- Design
- detail
- diagram
- field
- grid
- layer
- line
- metafile
- net
- NetClass
- pad
- PadViaShape
- PadViaStyle
- Pattern
- pin
- point
- Poly
- port
- pour
- PrintJob
- Room
- symbol
- table
- text
- TextStyle
- via
- wire

The **TItem** structure includes a field which is present for all item types returned: *itemType.* This field is an integer value indicating what type of item has been returned in the **TItem** structure. By using this value after a get Item function call (e.g. **GetFirstLayerItem**), you can determine what type of item has been returned and process the data appropriately. These item types are enumerated in both DBX32.H and DBX32.BAS. For ease of use, readability, and future program compatibility, we strongly recommend that you use the mnemonic constants within your program and *not* the integer values they represent. For example, to determine if the item returned is an arc item:

```
if myItem.itemType = DBX_ARC            ' Right

if myItem.itemType = 3                  ' Wrong
```

You can declare arrays of **TItem** items, but due to the size requirements for **TItem**, you should not declare large arrays of **TItem**s, but declare arrays of the specific DBX item type you intend to keep instead, an array of **TLine** items, for example.

2.6 Using a TItem to Modify Design Data

Several functions are available to modify PCB design data using a **TItem** directly as an input argument. This provides a mechanism to generically retrieve and modify a set of items without specifically writing code to make different DBX Modify calls based on the type of item retrieved. This is especially useful when using calls that return a TItem like **GetFirstNetItem** and **GetNextNetItem**, followed by a DBX Modify call to update those items regardless of the item type. For example, to retrieve all of the items in a specific net and highlight those items, you would use the **GetFirstNetItem/GetNextNetItem** calls followed by a call to **THighlightItem** with the extracted **TItem** as input to the highlight call.

The generic DBX Modify item functions are:

```
TFlipItem
TMoveItem
TModifyItem
TRotateItem
THighlightItem
TUnHighlightItem
TDeleteItem
```

2.7 Database Units vs. User Units

All functions returning or requiring size, location, length, width, or rotation values use values in database units. Using database units preserves database accuracy, and allows user programs to work in their unit of choice: database, mils, millimeter, or user defined. Database units are defined as 2540 database units per mil, and 10000 database units per mm. Rotation values are represented at the ratio of 10 database units per degree of rotation (in other words, 3600 database units is equivalent to 360 degrees).

As a convenience, DBXUTILS.H (for C and C++user programs) and DBXUTILS.BAS (for Visual Basic user programs) provide utility functions that convert database units to and from mils and mm, both as numbers and as strings.

2.8 Default Item Origins for Modify Operations

DBX Flip and Rotate functions have an optional parameter to specify the point about which to flip or rotate the item. If this point is not specified (by setting the x and y coordinate values of the rotate or flip point to -1) a default item origin is used for the operation. The default origin depends on the type of the item being flipped or rotated and is as follows in the form (x,y):

Arc: *(centerPt.x, centerPt.y)*
Attribute: *(refPoint.x, refPoint.y)*
Bus: *(startPt.x, startPt.y)*
Component: *(refPoint.x, refPoint.y)*
Field: *(refPoint.x, refPoint.y)*
Line: *(startPt.x, startPt.y)*
Free Pad or Via: *(center.x, center.y)*
Pin: *(refPoint.x, refPoint.y)*
Point: *(point.x, point.y)*
Port: *(point.x, point.y)*
Symbol: *(refPoint.x, refPoint.y)*
Table: *(refPoint.x refPoint.y)*
Text: *(refPoint.x, refPoint.y)*
Wire: *(startPt.x, starrtPt.y)*

The default flip or rotate point can be overridden by specifically setting the appropriate input argument to the function to have the coordinates of the flip or rotate point. See sections 5.1, 5.2, 6.1, and 6.2 for more information on how to do this.

2.9 DBX Conversation Structure tContext

P-CAD DBX user programs communicate with P-CAD applications using Windows Dynamic Data Exchange (DDE) protocols. Using this protocol allows DBX programs to communicate directly with an active P-CAD application during a design session. The DDE protocol requires certain conversation identification information each time one process communicates with another. This information is returned by **TOpenDesign** (**TOpenLibrary** in P-CAD Library Manger) in a P-CAD supplied structure called **TContext**, and must be passed to each DBX function call after the **TOpenDesign** call. For convenience, a global variable *tContext* is provided in DBX32.H and DBX32.BAS and may be used directly by your user program to receive and maintain this information. The global variable *tContext* is used throughout this manual and in the samples provided with your P-CAD DBX installation. Opening and closing a library is detailed in section 7.1.

2.10 Opening, Closing, and Saving a Design

You establish a connection between your user program and the PCB or Schematic design session by "opening" a design. To open the design, you use the function **TOpenDesign,** providing some basic information: the version of the P-CAD DBX software you are using, and the type of language you are using to write your user program, and the P-CAD application name, either "pcb" or "sch". **TOpenDesign** returns the P-CAD-supplied **TContext** structure *tContext,* which describes the connection you just established. This **TContext** structure must be used in all subsequent DBX function

calls to identify the DBX conversation. For example, to open the current P-CAD PCB design:

```
TOpenDesign(DBX_LANGUAGE, DBX_VERSION, "pcb",tContext)
```

DBX_VERSION and DBX_LANGUAGE are constants provided by P-CAD DBX and may be used exactly as written here. Also defined in DBX32.H and DBX32.BAS is *tContext,* a global variable which may also be used exactly as written here, and in subsequent function calls, to accept and maintain the conversation context information.

The 3rd argument to TOpenDesign, the P-CAD "Application:Design" name, should have a value of either "pcb" or "sch" to signify which P-CAD application to open. To provide easier migration of TangoPRO DBX programs to P-CAD, using an empty string ("") for the application name will open P-CAD PCB by default. Optionally, a colon followed by the full path and design file name may be specified. For example, "pcb:c:\P-CAD\files\demo2.pcb" ,may be used to open the "c:\P-CAD\files\demo2.pcb" design file in P-CAD PCB. If no design file name is specified, then the current design in the P-CAD application is used. If the design file name matches an existing open design, then that design will become the current design. If the design file name does not specify an existing file or there are any errors while loading the design then DBX_FILE_OPEN_FAILURE will be returned.

To save the current design, use **TSaveDesign**. This will write the current design to the file it was loaded from. If the current design is "Untitled", you will be prompted by the P-CAD application for the file name to save to. The design will always be saved even if no changes have been made. To determine if the current design has been modified, see **TGetDesignInfo**'s **isModified** flag.

To end your DBX session, use **TCloseDesign.** Note that *tContext* is the global structure returned by **TOpenDesign**.

```
TCloseDesign(tContext,"")
```

If the empty string ("") is specified for the second argument then the connection is closed but the design is left open in the P-CAD application. Optionally you may specify "FileClose" as the second argument to close the connection to the P-CAD application and close the design also.

Important: **TCloseDesign** will not save the design even if changes have been made.

A DBX conversation with P-CAD PCB or Schematic may also be terminated by clicking Cancel on the dialog which is presented in the design window when your DBX program begins executing. Using **TCloseDesign** is the preferred method, but the Cancel dialog is useful should your DBX program terminate abnormally or find itself in an infinite loop from which you cannot exit. The Cancel dialog will return P-CAD PCB or Schematic to a state from which you can safely continue your PCB design session.

For information about Library DBX see sections 7.1 on opening and closing a component library.

3 Retrieving Data - PCB

This section provides a summary of the interface functions and data returned by P-CAD DBX PCB functions when retrieving P-CAD PCB design information. The functions are grouped by the type of data they retrieve. For a detailed description of each function, its arguments, or P-CAD DBX structure contents, see DBX32.H, or the appendices at the end of this manual.

To enable code sharing between DBX applications used for PCB, Schematic, and Library Manager or Library Executive, function calls and DBX items are intentionally similar where possible.

3.1 Extracting General Design Data

General design information is retrieved using the function

```
TGetDesignInfo(tContext,designInfo)
```

where *tContext* is the structure returned by **TOpenDesign.** Design data is returned the structure *designInfo,* which you declare as type **TDesign.** The **TGetDesignInfo** function returns general board information including the board size, workspace extents, relative grid origin, design name, title, author, version, date and modified flag.

3.2 Extracting Layers and Layer Data

Four functions return layer data. Layer data is returned in the DBX item *myLayerData,* which you declare as type **TLayer.**

```
TGetFirstLayer(tContext,myLayerData)
TGetNextLayer(tContext,myLayerData)
TGetLayerById(tContext,layerId,myLayerData)
TGetLayerByName(tContext,layerName,myLayerData)
```

Layer functions **GetFirstLayer, GetNextLayer, GetLayerById**, and **GetLayerByName** return layer name, layerID, type and bias, clearances, and plane status.

3.3 Extracting Layer Items

Two functions return the items belonging to a P-CAD PCB layer. Item data is returned in the DBX item *myItem,* which you declare as type **TItem.** See section 2.5 for a discussion of the **TItem** structure and its usage.

```
TGetFirstLayerItem(tContext,layerId,myItem)
TGetNextLayerItem(tContext,myItem)
```

Layer Item functions **GetFirstLayerItem** and **GetNextLayerItem** return PCB design items located on a layer in a **TItem** data item. The items returned include arcs, attributes, components, connections, cutouts, details, diagrams, keepouts, lines, metafiles, pads, polygons, points, copper pours, tables, text, and vias.

3.4 Extracting Nets and Net Data

Four functions return net data. Net data is returned in the DBX item *myNetData,* which you declare as type **TNet.**

```
TGetFirstNet(tContext,myNetData)
TGetNextNet(tContext,myNetData)

TGetNetById(tContext,netId,myNetData)
TGetNetByName(tContext,netName,myNetData)
```

Net functions **GetFirstNet, GetNextNet, GetNetById,** and **GetNetByName** return net name, netID, net length, number of nodes, and plane status.

3.5 Extracting NetClasses and NetClass Data

Four functions return NetClass data. NetClass data is returned in the DBX item *myNetClassData,* which you declare as type **TNetClass.**

```
TGetFirstNetClass(tContext,myNetClassData)
TGetNextNetClass(tContext,myNetClassData)

TGetNetClassById(tContext,netClassId,myNetClassData)
TGetNetClassByName(tContext,netClassName,myNetClassData)
```

NetClass functions **GetFirstNetClass, GetNextNetClass, GetNetClassById,** and **GetNetClassByName** return netClassName, netClassID, number of nets.

3.6 Extracting ClassToClass Data

Three functions return ClassToClass data. ClassToClass data is returned in the DBX item *myClassToClassData,* which you declare as type **TClassToClass.**

```
TGetFirstClassToClass(tContext,myClassToClassData)
TGetNextClassToClass(tContext,myClassToClassData)

TGetClassToClassById(tContext,netClassId1,netClassID2,
                     myClassToClassData)
```

ClassToClass functions **GetFirstClassToClass, GetNextClassToClass** and **GetClassToClassById** return NetClassName1, NetClassName2, NetClassID1 and NetClassID2. Note that NetClassID1 must be less than or equal to NetClassID2 in function TGetClassToClassById.

3.7 Extracting Net Nodes and Net Items

Four functions return the items and nodes defining a P-CAD PCB Net. Item data is returned in the DBX item *myitem,* which you declare as type **TItem**. See Section 2.5 for

a discussion of the **TItem** structure and its usage.

```
TGetFirstNetItem(tContext,netId,myItem)
TGetNextNetItem(tContext,myItem)

TGetFirstNetNode(tContext,netId,myItem)
TGetNextNetNode(tContext,myItem)
```

Net Item functions **GetFirstNetItem** and **GetNextNetItem** return PCB design items defining a net in a **TItem** data item. The items returned include arcs, connections, lines, and vias.

Net Node functions **GetFirstNetNode** and **GetNextNetNode** return PCB design items which are the nodes in the net in a **TItem** data item. These items include pads.

3.8 Extracting NetClassNets and Net Data

Two functions return NetClassNet data. NetClassNet data is returned in the DBX item *myNetData,* which you declare as type **TNet.**

```
TGetFirstNetClassNet(tContext,netClassId,myNetData)
TGetNextNetClassNet(tContext,myNetData)
```

NetClassNet functions **GetFirstNetClassNet** and **GetNextNetClassNet** return net name, netID*,* net length, number of nodes, and plane status.

3.9 Extracting Components and Component Data

Three functions return component data. Component data is returned in the DBX item *myCompData*, which you declare as type **TComponent.**

```
TGetFirstComponent(tContext,myCompData)
TGetNextComponent(tContext,myCompData)

TGetCompByRefDes(tContext,compRefDes,myCompData)
```

Component functions **GetFirstComponent**, **GetNextComponent,** and **GetCompByRefDes** return Component ID, refdes, component type, value, pattern name, location, number of pads, bounding box, and orientation information in a **TComponent** data item.

3.10 Extracting Component Pad and Pattern Data

Four functions return the pads and pattern items defining a PCB Component. Pad items are returned in the DBX item *myPad,* which you declare as type **TPad**. Pattern item data is returned in the DBX item *myItem,* which you declare as type **TItem.** See section 2.5 for a discussion of the **TItem** structure and its usage.

```
TGetFirstCompPad(tContext,refDes,myPad )
TGetNextCompPad(tContext,myPad)
```

```
TGetFirstCompItem(tContext,refDes,myItem)
TGetNextCompItem(tContext,myItem)
```

Pad and Via data returned by Layer Item, Net Item, and Component Pad functions includes location, pad or via style ID, the net ID, rotation, and bounding box information. Data for pads and vias is returned in **TPad** and **TVia** data items, respectively. Pad data additionally includes the pad pin number, pin designator, pin type, and component refdes.

Component Item functions **GetFirstCompItem** and **GetNextNetCompItem** return the PCB design items defining the component pattern in a **TItem** data item. The items returned include arcs, lines, points, polygons and text.

3.11 Extracting Pad, Via, and Text Style Data

Pads, Vias, and Text items each include a StyleId as part of their definition. This StyleId value may be used to get information about a style in general, or more specific information about the current pad/via/text item being examined. There are three functions available to retrieve style information for pads, via, and text. Style information is returned in *myPadStyle (*or *myTextStyle, myViaStyle*), which you declare as **TPadStyle** (or **TTextStyle, TViaStyle**).

```
TGetPadStyle(tContext,padStyleId,myPadStyle)
TGetTextStyle(tContext,textStyleId,myTextStyle)
TGetViaStyle(tContext,viaStyleId,myViaStyle)
```

Given a StyleId, **GetPadStyle** and **GetViaStyle** functions return the style type, style name, hole diameter, and x and y offsets in a **TPadViaStyle** data item.

Note that valid style ids range from 0 to 99 each for text, pad, and via styles. The styles used may not be consecutive, however, so looping through the style ids from 0 until a DBX_STYLE_NOT_FOUND to find all valid styles may stop too soon if there are "holes" in the used style ids. The correct approach is to loop from 0 to 99, recording which styles are valid based on the status returned from the GetStyle function. Another way to loop over pad or via styles is to use TGetFirstPadStyle and TGetNextPadStyle, or TGetFirstViaStyle and TGetNextViaStyle. These functions will skip any holes in the used style ids.

Text Style function **TGetTextStyle** returns the style name, font type, pen width, and text height in a **TTextStyle** data item.

3.12 Extracting Pad and Via Shape Data

There is a significant amount of design data which may be different for each pad, or more accurately, for each pad style on each of the active layers. This is also true for vias. There are two functions available to retrieve layer specific shape data from a pad or via style. Data is returned in *myPadShape* for pad shapes, and *myViaShape* for vias, which you declare as **TPadShape** and **TViaShape,** respectively.

```
TGetPadShapeByLayer(tContext,padStyleId,layerId,myPadShape)
TGetViaShapeByLayer(tContext,viaStyleId,layerId,myViaShape)
```

TGetPadShapeByLayer or T**GetViaShapeByLayer** functions return pad or via style,

layer type, hole diameter, shape, width, and height (or for thermals: outer diameter, inner diameter and spoke width) in a **TPadViaShape** data item.

3.13 Extracting Rooms and Room Data

Two functions return room data. Room data is returned in the DBX item *myRoomData,* which you declare as type **TRoom.**

```
TGetFirstRoom(tContext,myRoomData)
TGetNextRoom(tContext,myRoomData)
```

Room functions **TGetFirstRoom** and **TGetNextRoom** return room name, room ID*,* number of included components, boundary rectangle, placement side, is fixed, is flipped, is highlighted, room fill pattern, reference point and rotation angle.

3.14 Extracting Room Points

Two functions return room point data. Room point data is returned in the DBX item *myPointData,* which you declare as type **TPoint.**

```
TGetFirstRoomPoint(tContext,roomId,myPointData)
TGetNextRoomPoint(tContext,myPointData)
```

Room Point functions **TGetFirstRoomPoint** and **TGetNextRoomPoint** return item ID*,* x, y, point type, number, text info string, layer Id, is flipped, is visible and is highlighted.

3.15 Extracting Room Components

Two functions return room component data. Room component data is returned in the DBX item *myCompData,* which you declare as type **TComponent.**

```
TGetFirstIncludedRoomComponent(tContext,roomId,myCompData)
TGetNextIncludedRoomComponent(tContext,myCompData)
```

Room component functions **TGetFirstincludedRoomComponent** and **TGetNextIncludedRoomComponent** return component Id, reference designator string, component type string, value string, pattern name, library name, reference point, boundary rectangle, rotation angle, number of pads, number of pins, number of parts, is alpha, is flipped, is highlighted, is heterogeneous, connection type and is fixed.

3.16 Extracting Grids and Grid Data

Two functions return grid data. Grid data is returned in the DBX item *myGridData,* which you declare as type **TGrid.**

```
TGetFirstGrid(tContext,myGridData)
TGetNextGrid(tContext,myGridData)
```

Grid functions **TGetFirstGrid** and **TGetNextGrid** return grid ID and grid spacing which is

a string containing the spacing values separated by a comma.

3.17 Extracting Attributes

Fourteen functions return the attributes which are associated with design, layer, net, net class, class to class, room or component. Attribute data is returned in *myAttr,* which you declare as **TAttribute.** For components, you specify the component *refDes* using a string variable in Visual Basic, or a Char* in C or C++. For all others specify the ID declared as a long.

```
TGetFirstDesignAttribute(tContext, myAttr)
TGetNextDesignAttribute(tContext, myAttr)
TGetFirstLayerAttribute(tContext,layerId, myAttr)
TGetNextLayerAttribute(tContext, myAttr)
TGetFirstNetAttribute(tContext,netId, myAttr)
TGetNextNetAttribute(tContext, myAttr)
TGetFirstNetClassAttribute(tContext, NetClassId,myAttr)
TGetNextNetClassAttribute(tContext, myAttr)
TGetFirstClassToClassAttribute(tContext, NetClassId1,
                               NetClassId2, myAttr)
TGetNextClassToClassAttribute(tContext, myAttr)
TGetFirstRoomAttribute(tContext, RoomId,myAttr)
TGetNextRoomAttribute(tContext, myAttr)
TGetFirstCompAttribute(tContext,refDes,myAttr)
TGetNextCompAttribute(tContext, myAttr)
```

The above functions return type and value strings for each attribute associated with the class in a **TAttribute** data item. These attributes also include location, style, and orientation information. Additionally, the layer functions return pre-defined clearance information.

3.18 Extracting Polygon Points

Two functions return the points defining a polygon. Point data is returned in *myPoint,* which you declare as type **TPoint.**

```
TGetFirstPolyPoint(tContext,polyId,myPoint)
TGetNextPolyPoint(tContext,myPoint)
```

TGetFirstPolyPoint and **TGetNextPolyPoint** return the point location and layer id **TPoint** data item.

3.19 Extracting Items from the Current Selection Set

Two functions return the items in the active design selection set (items you have selected by Block Select or selecting individual items). Item data is returned in selectItem, which you declare as type **TItem.**

```
TGetFirstSelectedItem(tContext,selectItem)
TGetNextSelectedItem(tContext,selectItem)
```

Get Selection functions return selected items in a **TItem** data item. The item types returned in the **TItem** include arc, attribute, connection, component, cutout, detail, diagram, keepout, line, metafile, pad, point, polygon, pour, table, text, and via.

3.20 Extracting Print Jobs

Two functions return the print jobs in the active design. Print job information is returned in *aJob*, which you declare as type **TPrintJob**.

```
TGetFirstPrintJob(tContext, aJob)
TGetNextPrintJob(tContext, aJob)
```

Print Job functions **TGetFirstPrintJob** and **TGetNextPrintJob** return the name of the print job and if it is selected for output during a print operation.

3.21 Extracting Layer Stackup

Two functions retrieve the layer stackup information in the active design. Layer stackup information is returned in pStackup, which you declare as type **TLayerStackup.**.

```
TGetFirstLayerStackup(tContext, pStackup)
TGetNextLayerStackup(tContext, pStackup)
```

Layer Stackup functions **TGetFirstLayerStackup** and **TGetNextLayerStackup** return a TLayerStackup struct that contains the layerName, layerMaterial, layerThickness, and layerDialectricConstant.

3.22 Extracting Variants

These methods are used for retrieving all the existing Variants in a design.

```
long TGetFirstVariant(DbxContext* pContext, TVariant* pVariant)
long TGetNextVariant(DbxContext* pContext, TVariant* pVariant)
```

Variant functions **TGetFirstVariant** and **TGetNextVariant** return TVariant structure that contains the Variant name, description, and id.

4 Retrieving Data - Schematic

This section provides a summary of the interface functions and data returned by P-CAD DBX functions when retrieving P-CAD Schematic data. The functions are grouped by the type of data they retrieve. For a detailed description of each function, its arguments, or P-CAD DBX structure contents, see DBX32.H, or the appendices at the end of this manual.

To enable code sharing between DBX applications used for PCB, Schematic, and Library Manager or Library Executive, function calls and DBX items are intentionally similar where possible.

4.1 Extracting General Design Data

General design information is retrieved using the function

```
TGetDesignInfo(tContext,designInfo)
```

where *tContext* is the structure returned by **TOpenDesign.** Design data is returned the structure *designInfo*, which you declare as type **TDesign.** The **TGetDesignInfo** function returns general design information including the design size, workspace extents, relative grid origin, design name, title, author, version, and date.

4.2 Extracting Sheets and Sheet Data

Four functions return sheet data. To provide consistency between the PCB and Schematic interfaces, sheet and sheet data are referenced as Layers. Sheet data is returned in the DBX item *myLayerData*, which you declare as type **TLayer.**

```
TGetFirstLayer(tContext,myLayerData)
TGetNextLayer(tContext,myLayerData)
TGetLayerById(tContext,layerId,myLayerData)
TGetLayerByName(tContext,layerName,myLayerData)
```

Layer functions **TGetFirstLayer, TGetNextLayer, TGetLayerById**, and **TGetLayerByName** return the sheet name (sheet description) as layer name and the sheet number as layerId. *layerId* zero is returned, but not useable. Schematic DBX applications should skip to *layerId* 1.

4.3 Extracting Sheet Items

Two functions return the items on a Schematic sheet. To provide consistency between the PCB and Schematic interfaces, sheet and sheet data are referenced as Layers. Item data is returned in the DBX item *myItem*, which you declare as type **TItem.** See section 2.5 for a discussion of the **TItem** structure and its usage.

```
TGetFirstLayerItem(tContext,layerId,myItem)
TGetNextLayerItem(tContext,myItem)
```

Layer Item functions **TGetFirstLayerItem** and T**GetNextLayerItem** return Schematic

design items located on a sheet in a **TItem** data item. The items returned include arcs, attributes, buses, fields, info points, lines, pins, ports, ref points, symbols, table, text, and wires.

4.4 Extracting Nets and Net Data

Four functions return net data. Net data is returned in the DBX item *myNetData,* which you declare as type **TNet.**

```
TGetFirstNet(tContext,myNetData)
TGetNextNet(tContext,myNetData)

TGetNetById(tContext,netId,myNetData)
TGetNetByName(tContext,netName,myNetData)
```

Net functions **TGetFirstNet, TGetNextNet, TGetNetById,** and **TGetNetByName** return net name, netID, and number of nodes.

4.5 Extracting NetClasses and NetClass Data

Four functions return NetClass data. NetClass data is returned in the DBX item *myNetClassData,* which you declare as type **TNetClass.**

```
TGetFirstNetClass(tContext,myNetClassData)
TGetNextNetClass(tContext,myNetClassData)

TGetNetClassById(tContext,netClassId,myNetClassData)
TGetNetClassByName(tContext,netClassName,myNetClassData)
```

Net functions **TGetFirstNetClass, TGetNextNetClass, TGetNetClassById,** and T**GetNetClassByName** return netClassName, netClassId, number of nets.

4.6 Extracting ClassToClass Data

Three functions return ClassToClass data. ClassToclass data is returned in the DBX item *myClassToClassData,* which you declare as type **TClassToClass.**

```
TGetFirstClassToClass(tContext,myClassToClassData)
TGetNextClassToClass(tContext,myClassToClassData)

TGetClassToClassById(tContext,netClassId1,netClassID2,myClassTo
                     ClassData)
```

Net functions **TGetFirstClassToClass, TGetNextClassToClass** and **TGetClassToClassById** return netClassName1, netClassName2, netClassId1 and netClassId2. Note that in function **TGetClassToClassById** NetClassID1 must be less than or equal to NetClassID2.

4.7 Extracting Net Nodes and Net Items

Two functions return the items defining a P-CAD Schematic Net. Item data is returned in the DBX item *myitem,* which you declare as type **TItem.** See Section 2.5 for a discussion of the **TItem** structure and its usage.

```
TGetFirstNetNode(tContext,netId,myItem)
TGetNextNetNode(tContext,myItem)
```

Net Node functions **TGetFirstNetNode** and T**GetNextNetNode** return Schematic design items which are the nodes in the net in a **TItem** data item. These items are pins.

4.8 Extracting NetClassNets and Net Data

Two functions return NetClassNet data. NetClassNet data is returned in the DBX item *myNetData,* which you declare as type **TNet.**

```
TGetFirstNetClassNet(tContext,netClassId,myNetData)
TGetNextNetClassNet(tContext,myNetData)
```

NetClassNet functions **GetFirstNetClassNet** and **GetNextNetClassNet** return net name, netID, net length, number of nodes, and plane status.

4.9 Extracting Components and Component Data

Three functions return component data. Component data is returned in the DBX item *myCompData,* which you declare as type **TComponent.**

```
TGetFirstComponent(tContext,myCompData)
TGetNextComponent(tContext,myCompData)

TGetCompByRefDes(tContext,compRefDes,myCompData)
```

Component functions **TGetFirstComponent, TGetNextComponent,** and T**GetCompByRefDes** return Component ID, refdes, component type, value, pattern name, number of pins, number of parts, isAlpha, isHetero, and connection type information in a **TComponent** data item.

4.10 Extracting Component Pin and Symbol Data

Six functions return pin and symbol data. Symbol items are returned in the DBX item *mySymbol,* which you declare as type **TSymbol.** Pin data is returned in the DBX item *myPin,* which you declare as type **TPin.** The component is specified by a character string, *refDes*. This is the component, not the symbol, refDes (e.g. U1).

```
TGetFirstCompPin(tContext,refDes,myPin)
TGetNextCompPin(tContext,myPin)

TGetFirstCompSymbol(tContext,refDes,mySymbol)
TGetNextCompSymbol(tContext,mySymbol)
```

```
TGetCompSymbolByPartNumber(tContext,refDes,PartNo,mySymbol)
TGetCompSymbolByRefDes (tContext,refDes,mySymbol)
```

Symbol data returned includes the symbol Id, name, symbol refdes (e.g. U1:A), number of pins, part number and alternate type. Pin data returned in myPin includes the pin item Id, pin number, symbol part number, component refDes (e.g. U1), pin pinDes (e.g. 1), pin type, outside and inside style and edge styles, net Id, gate equivalence value, and pin equivalence value.

4.11 Extracting Symbol Pin Data

Two functions return symbol pin data. Pin data is returned in the DBX item *myPin*, which you declare as type **TPin.** The symbol is specified by a character string, *refDes*. This is the full symbol refDes (e.g. U1:A).

```
TGetFirstSymbolPin(tContext,refDes,myPin)
TGetNextSymbolPin(tContext,myPin)
```

Pin data returned in myPin includes the pin item Id, pin number, symbol part number, component refDes (e.g. U1), pinDes (e.g. 1), pin type, outside and inside style and edge styles, net Id, gate equivalence value, and pin equivalence value.

4.12 Extracting Style Data for Text Items

Text items include a StyleId as part of their definition. This StyleId value may be used to get information about a style in general, or more specific information about the current text item being examined. There are three functions available to retrieve style information. Style information is returned in *myTextStyle,* which you declare as **TTextStyle.**

```
TGetTextStyle(tContext,textStyleId,myTextStyle)
```

Note that valid style ids range from 0 to 99 for text styles. The styles used may not be consecutive, however, so looping through the style ids from 0 until a DBX_STYLE_NOT_FOUND to find all valid styles may stop too soon if there are "holes" in the used style ids. The correct approach is to loop from 0 to 99, recording which styles are valid based on the status returned from the GetStyle function.

Text Style function **TGetTextStyle** returns the style name, font type, pen width, and text height in a **TTextStyle** data item.

4.13 Extracting Grids and Grid Data

Two functions return grid data. Grid data is returned in the DBX item *myGridData,* which you declare as type **TGrid.**

```
TGetFirstGrid(tContext,myGridData)
TGetNextGrid(tContext,myGridData)
```

Grid functions **TGetFirstGrid** and **TGetNextGrid** return grid ID and grid spacing which is

a string containing the spacing values separated by a comma.

4.14 Extracting Attributes

Ten functions return the attributes which are associated with design, layer, net, net class, class to class, room or component. Attribute data is returned in *myAttr,* which you declare as **TAttribute.** For components, you specify the component *refDes* using a string variable in Visual Basic, or a char* in C or C++. For all others specify the ID declared as a long.

```
TGetFirstDesignAttribute(tContext, myAttr)
TGetNextDesignAttribute(tContext, myAttr)
TGetFirstNetAttribute(tContext,netId, myAttr)
TGetNextNetAttribute(tContext, myAttr)
TGetFirstNetClassAttribute(tContext,NetClassId, myAttr)
TGetNextNetClassAttribute(tContext, myAttr)
TGetFirstClassToClassAttribute(tContext, NetClassId1,
                               NetClassId2, myAttr)
TGetNextClassToClassAttribute(tContext, myAttr)
TGetFirstCompAttribute(tContext,refDes,myAttr)
TGetNextCompAttribute(tContext, myAttr)
```

The above functions return type and value strings for each attribute associated with the class in a **TAttribute** data item. These attributes also include location, style, and orientation information.

4.15 Extracting Symbol Attributes

Two functions return the attributes which are associated with a symbol. Attribute data is returned in *myAttr,* which you declare as **TAttribute**. For symbols, you specify the symbol, *refDes* (i.e. U1:A) using a string variable, in Visual Basic, or a Char*, in C or C++.

```
TGetFirstSymAttribute (tContext, refDes, myAttr)
TGetNextSymAttribute (tContext, myAttr)
```

Symbol Attribute functions **TGetFirstCompAttribute** and T**GetNextCompAttribute** return type and value strings for each attribute associated to a symbol in a **TAttribute** data item. These attributes also include location, style, and orientation information.

4.16 Extracting Items from the Current Selection Set

Two functions return the items in the active design selection set (items you have selected by Block Select or selecting individual items). Item data is returned in *selectItem,* which you declare as type **TItem.** See section 2.5 for a discussion of the **TItem** structure and its usage.

```
TGetFirstSelectedItem(tContext,selectItem)
TGetNextSelectedItem(tContext,selectItem)
```

Get Selection functions return selected items in a **TItem** data item. The item types returned in the**TItem** include arcs, attributes, buses, fields, info points, lines, pins, ports,

ref points, symbols, tables, text, and wires.

4.17 Extracting Print Job

Two functions return the print jobs in the active design. Print job information is returned in *aJob*, which you declare as type **TPrintJob**.

```
TGetFirstPrintJob(tContext, aJob)
TGetNextPrintJob(tContext, aJob)
```

Print Job functions **TGetFirstPrintJob** and **TGetNextPrintJob** return the name of the print job and if it is selected for output during a print operation.

4.18 Extracting Variants

These methods are used for retrieving all the existing Variants in a design.

```
long TGetFirstVariant(DbxContext* pContext, TVariant* pVariant)
long TGetNextVariant(DbxContext* pContext, TVariant* pVariant)
```

Variant functions **TGetFirstVariant** and **TGetNextVariant** return TVariant structure that contains the Variant name, description, and id.

5 Modifying Design Data - PCB

This section provides a summary of the interface functions that add, modify, or delete PCB design data. Note that these functions are available only for P-CAD P-CAD PCB. For a detailed description of each function, its arguments, or P-CAD DBX structure contents, see DBX32.H, or the appendices at the end of this manual.

Note that DBX items returned by a successful Modify operation will have a new dbId or compId database identifier. The old dbId or compId value will no longer be a valid identifier; the modified DBX item returned by the function call should be used for subsequent DBX function calls.

To enable code sharing between DBX applications used for PCB, Schematic, and Library Manager or Library Executive, function calls and DBX items are intentionally similar where possible.

5.1 Flipping Objects

Nine functions flip PCB design objects. Input to the function is the item to be flipped, and the point about which the item is to be flipped. Items may be flipped using item type specific function calls, or by using a DBX **TItem** as input to **TFlipItem.**

The item to be flipped is declared as a **TArc, TAttribute, TComponent, TField, TLine, TPad, TPoint, Ttable, TText, TVia** or **TItem,** respectively. The point about which to flip, *pPoint,* is declared as a **TCoord.** If the **TCoord** coordinates are specified as (-1,-1), the item is flipped about its default origin. See Section 2.8, Default Item Origins for a listing of what item property is used as a default origin.

```
TFlipArc(tContext, pPoint, pArc)
TFlipAttribute(tContext, pPoint, pAttribute)
TFlipComponent(tContext, pPoint, pComponent)
TFlipField(tContext, pPoint, pField)
TFlipLine(tContext, pPoint, pLine)
TFlipPad(tContext, pPoint, pPad)
TFlipPoint(tContext, pPoint, pPoint2)
TFlipTable(tContext, pPoint, pTable)
TFlipText(tContext, pPoint, pText)
TFlipVia(tContext, pPoint, pVia)

TFlipItem(tContext, pPoint, TItem)
```

DBX Flip functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **TFlipItem.** The item is updated to have its *isFlipped* field set appropriately.

5.2 Rotating Objects

Ten functions rotate PCB design objects. Input to the function is the item to be rotated, the rotation angle, and the point about which the item is to be rotated. Items may be rotated using item type specific function calls, or by using a DBX **TItem** as input to **TRotateItem.**

The item to be rotated is declared as a **TArc, TAttribute, TComponent, Tfield, TLine, TPad, Tpoint, TTable**, **TText, TVia** or **TItem**, respectively. The rotation angle angle, is declared as a long and may range from -3600 to 3600 (recall that degree values are specified in 1/10 degrees). The point about which to rotate, *pPoint,* is declared as a **TCoord.** If the **TCoord** coordinates are specified as (-1,-1), the item is rotated about its default origin. See Section 2.8, Default Item Origins for a listing of what item property is used as a default origin.

```
TRotateArc(tContext, angle, pPoint, pArc)
TRotateAttribute(tContext, angle, pPoint, pAttribute)
TRotateComponent(tContext, angle, pPoint, pComponent)
TRotateField(tContext, angle, pPoint, pField)
TRotateLine(tContext, angle, pPoint, pLine)
TRotatePad(tContext, angle, pPoint, pPad)
TRotatePoint(tContext, angle, pPoint, pPoint2)
TRotateTable(tContext, angle, pPoint, pTable)
TRotateText(tContext, angle, pPoint, pText)
TRotateVia(tContext, angle, pPoint, pVia)

TRotateItem(tContext, angle, pPoint, pItem)
```

DBX Rotate functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **TRotateItem**. The item *rotateAngle* or *rotation* fields are updated to reflect the item's new orientation.

5.3 Moving Objects

Thirteen functions move PCB design objects. Input to the function is the item to be moved, and the distance in the x and y directions to move the item. Items may be moved using item type specific function calls, or by using a DBX **TItem** as input to **TMoveItem**.

The item to be moved is declared as a **TArc, TAttribute, TComponent, TDetail, TDiagram, TLine, TMetafile, TPad, TPoint, TTable, TText, TVia** or **TItem**, respectively. The distances to move, *dx* and *dy,* are declared as a longs and are in database units.

```
TMoveArc(tContext, dx, dy, pArc)
TMoveAttribute(tContext, dx, dy, pAttribute)
TMoveComponent(tContext, dx, dy, pComponent)
TMoveDetail(tContext, dx, dy, pDetail)
TMoveDiagram(tContext, dx, dy, pDiagram)
TMoveLine(tContext, dx, dy, pLine)
TMoveMetafile(tContext, dx, dy, pMetafile)
TMovePad(tContext, dx, dy, pPad)
TMovePoint(tContext, dx, dy, pPoint)
TMoveTable(tContext, dx, dy, pTable)
TMoveText(tContext, dx, dy, pText)
TMoveVia(tContext, dx, dy, pVia)

TMoveItem(tContext, dx, dy, pItem)
```

DBX Move functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **TMoveItem.** The item location information is updated to reflect the item's new location after the Move operation.

5.4 Deleting Objects

Seventeen functions delete PCB design objects. Input to the function is the item to be deleted. Items may be deleted using item type specific function calls, or by using a DBX **TItem** as input to **TDeleteItem**.

The item to be deleted is declared as a **TArc, TAttribute, TClassToClass, TComponent, TDetail, TDiagram, TLine, TMetafile, TPad, TPoint, TText, TVia** or **TItem** respectively.

```
TDeleteArc(tContext, pArc)
TDeleteAttribute(tContext, pAttribute)
TDeleteClassToClass(tContext, pClassToClass)
TDeleteComponent(tContext, pComponent)
TDeleteDetail(tContext, pDetail)
TDeleteDiagram(tContext, pDiagram)
TDeleteIncludedRoomComponent(tContext, roomId, pComponent)
TDeleteLine(tContext, pLine)
TDeleteNetClass(tContext, pNetClass)
TDeleteNetClassNet(tContext,NetClassId, pNet)
TDeleteMetafile(tContext, pMetafile)
TDeletePad(tContext, pPad)
TDeletePoint(tContext, pPoint)
TDeleteTable(tContext, pTable)
TDeleteText(tContext, pText)
TDeleteVia(tContext, pVia)
TDeleteItem(tContext,pItem)
```

DBX Delete functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **TDeleteItem**. The item is indicated as no longer being a valid PCB design item by having its DBID property set equal to zero.

5.5 Highlighting Objects

Twenty-five functions highlight and unhighlight PCB design objects. Input to each function is the item to be highlighted or unhighlighted, and a highlight color for highlight functions. Items may be highlighted or unhighligted using item type specific function calls, or by using a DBX **TItem** as input to **THighlightItem** and **TUnHighlightItem**.

All items in the active design may be unhighlighted using the **TUnHighlightAll** function.

All items in a net may be highlighted or unhighlighted using the **THighlightNet** and **TUnHighlightNet** functions, respectively.

The item to be highlighted is declared as a **TArc, TAttribute, TComponent, TLine, TNet, TPad, TPoint, TRoom, TTable, TText, TVia** or **TItem**, respectively. The highlight color color is declared as a long value and specifies highlight colors 1 through 20. The 20 colors associated with the highlight color constants are listed in Appendix A, P-CAD DBX Data Constants under DBX Color Index Types.

```
THighlightArc(tContext, color, pArc)
THighlightAttribute(tContext, color, pAttribute)
THighlightComponent(tContext, color, pComponent)
```

```
THighlightLine(tContext, color, pLine)
THighlightPad(tContext, color, pPad)
THighlightPoint(tContext, color, pPoint)
THighlightTable(tContext, color, pTable)
THighlightText(tContext, color, pText)
THighlightVia(tContext, color, pVia)

TUnHighlightArc(tContext, pArc)
TUnHighlightAttribute(tContext, pAttribute)
TUnHighlightComponent(tContext, pComponent)
TUnHighlightLine(tContext, pLine)
TUnHighlightPad(tContext, pPad)
TUnHighlightPoint(tContext, pPoint)
TUnHighlightTable(tContext, pTable)
TUnHighlightText(tContext, pText)
TUnHighlightVia(tContext, pVia)

THighlightNet(tContext, color, pNet)
TUnHighlightNet(tContext, pNet)

THighlightRoom(tContext, color, pRoom)
TUnHighlightRoom(tContext, pRoom)

THighlightItem(tContext, color, pItem)
TUnHighlightItem(tContext, pItem)
TUnHighlightAll(tContext)
```

DBX Highlight functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **THighlightItem** or **TUnHighlightItem.** Individual items highlighted or unhighlighted are returned having their *isHighlighted* field updated to indicate its highlight color. (*isHighlighted* is 0 to indicate that the item is not highlighted and ranges 1 through 20 to specify which highlight color is in use.)
In addition, Details, Diagrams, and Metafiles can be highlighted or unhighlighted by using the **THighlightItem** or **TUnHighlightItem** with the apporiprate fields of the **TItem** structure filled in.

5.6 Modifying Object Properties

Twelve functions modify PCB design objects properties. Input to the function is the item to be modified. Items may be modified using item type specific function calls, or by using a DBX **TItem** as input to **TModifyItem.**

The modification to be performed is specified by setting one or more of the DBX item properties prior to calling the Modify function. The properties that may be modified by a DBX program are specific to the type of item being modified and are listed below. All modifications must conform to P-CAD PCB design rules. For example, component *refDes* values must be unique and contain no invalid characters; a location or rotation which would move the item outside the design workspace will return a DBX error value of DBX_ITEM_OUTSIDE_WORKSPACE; moving an item to an invalid layer will return an error value of DBX_INVALID_LAYER.

The item to be modified is declared as a **TArc, TAttribute, TComponent, TLine, TNet, TNetClass, TPad, Tpoint, TRoom, TText, TVia** or **TItem,** respectively.

The modifiable properties for each DBX item are listed below:

Arc: *width, radius, centerPt, startAng, sweepAng, layerId*
Attribute: *value, refPoint, textStyleId, justPoint, layerId, formula, comment, units*
Component: *refDes, value, refPoint* (component location)
Component Pad: *styleId*
Free Pad: *styleId, location, padNum*
Line: *width, startPt, endPt, layerId*
Net: *netName*
NetClass: *netClassName*
Point: *point* (the point location)
Room: *roomName, fillPattern, placementSide, isFixed*
Text: *text, refPoint, textStyleId, justPoint, isVisible, layerId*
Via: *styleId, location*

```
TModifyArc(tContext, pArc)
TModifyAttribute(tContext, pAttribute)
TModifyComponent(tContext, pComponent)
TModifyLine(tContext, pLine)
TModifyNet(tContext, pNet)
TModifyNetClass(tContext, pNetClass)
TModifyPad(tContext, pPad)
TModifyPoint(tContext, pPoint)
TModifyRoom(tContext, pRoom)
TModifyText(tContext, pText)
TModifyVia(tContext, pVia)

TModifyItem(tContext, pItem)
```

DBX Modify functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **TModifyItem.** The item properties are updated to reflect the item status after the Modify call. Item properties not legal for modification that were modified in the DBX item prior to the function call are overwritten with valid data representing the item's current state after the Modify call.

Note: For information on modifying component and net attributes, see sections 5.13 and 5.14, respectively.

5.7 Placing Objects

Eight functions place PCB design objects in an active PCB design. Input to the function is the item to be placed. There is no generic **TPlaceItem** form.

The properties for the item to be placed is specified by setting one or more of the DBX item properties prior to calling the Place function. The properties that must be specified during placement are specific to the type of item being placed and are listed below. All items to be placed, and the properties of those items, must conform to P-CAD PCB design rules. For example, component *refDes* values must be unique and contain no invalid characters; a location or rotation which would place the item outside the design workspace will return a DBX error value of DBX_ITEM_OUTSIDE_WORKSPACE; placing an item on an invalid layer will return an error value of DBX_INVALID_LAYER.

The item to be placed is declared as a **TArc, TAttribute, TComponent, TLine, TPad, TPoint, Ttext** and **TVia,** respectively.

The specifiable properties for each DBX item are listed below. Note that except for *component.libraryName, component.typeName,* and *point.infoText,* these are exactly the properties valid for DBX Modify operations. Note also that all properties listed must be specified to place a new design item.

Arc: *width, radius, centerPt, startAng, sweepAng, layerId*
Attribute: *type, value, refPoint, textStyleId, justPoint, isVisible, layerId*
Component: *refDes, value, refPoint, libraryName, compType*
Line: *width, startPt, endPt, layerId*
Net: *netName*
Free Pad: *padStyleId, center, pinNumber*
Via: *viaStyleId, center*
Point: *pointType x, y (location), layerId; InfoText, ruleCategory, ruleType (InfoPoint only)*
Text: *text, refPoint, textStyleId, justPoint, isVisible, layerId*

Note: When placing a component *component.libraryName*, if specified, must include the complete library path and file name (e.g. c:\P-CAD\demo.lib). If not specified by leaving the input property field blank, the open libraries will be searched in the current open library order for the input component type.

```
TPlaceArc(tContext, pArc)
TPlaceAttribute(tContext, pAttribute)
TPlaceComponent(tContext, pComponent)
TPlaceLine(tContext, pLine)
TPlacePad(tContext, pPad)
TPlacePoint(tContext, pPoint)
TPlaceText(tContext, pText)
TPlaceVia(tContext, pVia)
```

DBX Place functions return a DBX error return status, and an updated DBX item. The item properties are updated to reflect the item status after the Place call. Item properties not legal to be specified during placement that were specified in the DBX item prior to the function call are overwritten with valid data representing the item's current state after the Place call.

5.8 Creating and Deleting Nets

Two functions create and delete a PCB design net. Input to the **TCreateNet** function is a DBX **TNet** item. The *net.netName* field must specify the name of the new net. The name must be a valid netname, and not be a duplicate of an existing net. All other **TNet** properties are ignored.

A PCB net must contain no design items (lines, arcs, vias) and have no net nodes to be deleted. See Sections 5.4, Deleting Objects, and 5.9, Adding and Deleting Net Nodes for information on how to delete net items and net nodes.

Note: When deleting a net, net objects (using **TGetFirstNetItem** and **TGetNextNetItem**) should be deleted first, then delete the net nodes (using **TGetFirstNetNode** and **TGetNextNetNode**). It is not necessary to delete all net attributes before deleting a net.

```
TCreateNet(tContext, pNet)
TDeleteNet(tContext, pNet)
```

DBX **TCreateNet** and **TDeleteNet** functions return a DBX error return status, and an updated DBX net item. If successful, a created **TNet** will be returned and indicate a valid net number in the *net.netNumber* field. A deleted net will be indicated by a *net.netNumber* field value of zero.

5.9 Adding and Deleting Net Nodes

Two functions add and delete PCB net nodes. Input to these functions is the net number, and the pad to be added or deleted. *netId* is declared as a long, and must be a valid net number. *pItem* is the node to be added or deleted from the net, and is declared as a **TItem** with the pad structure filled in.

```
TAddNetNode(tContext, netId, pItem)
TDeleteNetNode(tContext, netId, pItem)
```

DBX **TAddNetNode** and **TDeleteNet** functions return a DBX error return status, and an updated DBX **TItem** with the pin structure filled in. If a pad has been successfully added to a net, the pad *netId* property will be updated to reflect the net number of the net to which it has been added. If a pad has been successfully deleted from a net, the pad *netId* property will be set to zero, indicating it belongs to no nets.

5.10 Creating and Deleting NetClasses

Two functions create and delete a PCB design NetClass. Input to the **TCreateNetClass** function is a DBX **TNetClass** item. The *netClass.netClassName* field must specify the name of the new NetClass. The name must be a valid netClassName, and not be a duplicate of an existing NetClass. All other **TNet** properties are ignored.

```
TCreateNetClass(tContext, pNetClass)
TDeleteNetClass(tContext, pNetClass)
```

DBX **TCreateNetClass** and **TDeleteNetClass** functions return a DBX error return status, and an updated DBX net class item. If successful, a created **TNetClass** will be returned and indicate a valid net class number in the *netClass.numberOfNets* field.

5.11 Adding and Deleting NetClassNets

Two functions add and delete a PCB design net class net. Input to the **TCreateNetClassNet** function is a DBX **netClassId** and a **TNet** item. The **netClassId** is declared as a long and must be a valid NetClass number. **pNet** is the net to be added or deleted from the net, and is declared as a **TNet** with the net structure filled in.

```
TAddNetClassNet(tContext, netClassID, pNet)
TDeleteNetClassNet(tContext, netClassId, pNet)
```

DBX **TAddNetClassNet** and **TDeleteNetClassNet** functions return a DBX error return status and the number of nets in the net class is incremented or decremented.

5.12 Creating and Deleting ClassToClass Rules

Two functions create and delete a PCB design ClassToClass rules. Input to the **TCreateClassToClass** function is a DBX **TClassToClass** item.

```
TCreateClassToClass(tContext, pClassToClass)
TDeleteClassToClass(tContext, pClassToClass)
```

DBX **TCreateClassToClass** and **TDeleteClassToClass** functions return a DBX error return status.

5.13 Modifying Component Attributes

Three functions modify component attributes. Input to these functions is the component *refdes,* which you declare as a string variable in Visual Basic, and as a char* in C or C++, and the attribute to be added, modified, or deleted, *pAttribute*, declared as a **TAttribute.**

```
TAddCompAttribute(tContext, refDes, pAttribute)
TModifyCompAttribute(tContext, refDes, pAttribute)
TDeleteCompAttribute(tContext, refDes, pAttribute)
```

The following **TAttribute** properties are used when adding an attribute to a component: *type, value, refPoint, textStyleId, justPoint, isVisible*, and *layerId*. All properties must be fully specified unless *isVisible=0,* where the following defaults are used:

refPoint: will be placed at the component reference point
textStyleId: default
justPoint: lower left justification
layerId: Top Silk layer

The following **TAttribute** properties may be modified using the **TModifyCompAttribute** function: *value, refPoint, textStyleId, justPoint, isVisible,* and *layerId*. Note that a component refDes is modifiable only by using **TModifyComponent** with the component *refDes* property updated.

DBX Modify Component Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified attributes will be returned with all attribute properties updated. A deleted attribute will be returned with its DB ID set to zero.

5.14 Modifying Net Attributes

Three functions modify net attributes. Input to these functions is the net number, *netId,* which you declare as a long, and the attribute to be added, modified, or deleted, *pAttribute,* declared as a **TAttribute.**

```
TAddNetAttribute(tContext, netId, pAttribute)
TModifyNetAttribute(tContext, netId, pAttribute)
TDeleteNetAttribute(tContext, netId, pAttribute)
```

The following **TAttribute** properties are used when adding an attribute to a net or modifying an existing net attribute: *type, value, formula, comment and units.* Net

attributes are not visible. All other properties are ignored when adding or modifying a net attribute.

DBX Modify Net Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified attributes will be returned with all attribute properties updated. A deleted attribute will be returned with its DB ID set to zero.

5.15 Modifying Design Attributes

Three functions modify design attributes. The attribute to be added, modified, or deleted, *pAttribute,* is declared as a **TAttribute.**

```
TAddDesignAttribute(tContext, pAttribute)
TModifyDesignAttribute(tContext, pAttribute)
TDeleteDesignAttribute(tContext, pAttribute)
```

The following **TAttribute** properties are used when adding an attribute to a design or modifying an existing design attribute: *type, value, formula, comment and units.* All other properties are ignored when adding or modifying a design attribute.

DBX Modify Design Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified attributes will be returned with all attribute properties updated. A deleted attribute will be returned with its DB ID set to zero.

5.16 Modifying Layer Attributes

Three functions modify layer attributes. The attribute to be added, modified, or deleted, *pAttribute,* is declared as a **TAttribute.**

```
TAddLayerAttribute(tContext, pAttribute)
TModifyLayerAttribute(tContext, pAttribute)
TDeleteLayerAttribute(tContext, pAttribute)
```

The following **TAttribute** properties are used when adding a layer attribute to a design or modifying an existing layer attribute: *type, value, formula, comment and units.* All other properties are ignored when adding or modifying a layer attribute.

DBX Modify Layer Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified layer attributes will be returned with all attribute properties updated. A deleted layer attribute will be returned with its DB ID set to zero.

5.17 Modifying NetClass Attributes

Three functions modify NetClass attributes. The attribute to be added, modified, or deleted, *pAttribute,* is declared as a **TAttribute.**

```
TAddNetClassAttribute(tContext, netClassId, pAttribute)
TModifyNetClassAttribute(tContext, netClassId, pAttribute)
TDeleteNetClassAttribute(tContext, netClassId, pAttribute)
```

The following **TAttribute** properties are used when adding a netClass attribute to a

design or modifying an existing netClass attribute: *type, value, formula, comment and units.* All other properties are ignored when adding or modifying a netClass attribute.

DBX Modify NetClass Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified NetClass attributes will be returned with all attribute properties updated. A deleted NetClass attribute will be returned with its DB ID set to zero.

5.18 Modifying ClassToClass Attributes

Three functions modify ClassToClass attributes. The attribute to be added, modified, or deleted, *pAttribute,* is declared as a **TAttribute.**

```
TAddClassToClassAttribute(tContext, netClassId1, netClassId2,
                          pAttribute)
TModifyClassToClassAttribute(tContext, netClassId1,
                          netClassId2, pAttribute)
TDeleteClassToClassAttribute(tContext, netClassId1,
                          netClassId2, pAttribute)
```

The following **TAttribute** properties are used when adding a ClassToClass attribute to a design or modifying an existing ClassToClass attribute: *type, value, formula, comment and units.* All other properties are ignored when adding or modifying a ClassToClass attribute.

DBX Modify ClassToClass Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified ClassToClass attributes will be returned with all attribute properties updated. A deleted ClassToClass attribute will be returned with its DB ID set to zero.

5.19 Modifying Room Attributes

Three functions modify Room attributes. The attribute to be added, modified, or deleted, *pAttribute,* is declared as a **TAttribute.**

```
TAddRoomAttribute(tContext, roomID, pAttribute)
TModifyRoomAttribute(tContext, roomId, pAttribute)
TDeleteRoomttribute(tContext, roomId, pAttribute)
```

The following **TAttribute** properties are used when adding a Room attribute to a design or modifying an existing Room attribute: *type, value, formula, comment and units.* All other properties are ignored when adding or modifying a Room attribute.

DBX Modify Room Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified Room attributes will be returned with all attribute properties updated. A deleted Room attribute will be returned with its DB ID set to zero.

5.20 Saving a Design

The following function is used to save the active design.

```
TSaveDesign(tContext)
```

The active design will be written to disk in the same manner as if the File Save command was chosen from the P-CAD application menu.

5.21 Selecting Print Jobs

Three functions select or deselect the print jobs in the active design. Selected print jobs are those that are output during the next print operation. Print jobs are specified in the *jobName* string.

```
TSelectPrintJob(tContext, jobName)
TDeselectPrintJob(tContext, jobName) TSelectAllPrintJob(tContext)
```

the **TSelectPrintJob** and **TDeselectPrintJob** functions will select or deselect the print job specified by its name. The **TSelectAllPrintJobs** function will select all the print jobs in the active design.

5.22 Printing Print Jobs

Two functions print the print jobs of the active design to the current printer. Print jobs are specified in the *jobName* string.

```
TOutputPrintJobByName(tContext, jobName)
TOutputSelectedPrintJobs(tContext)
```

The **TOutputPrintJobByName** function will print the job specified by its name, regardless of whether or not it has been selected for output. The **TOutputSelectedPrintJobs** function will print jobs that are selected for output.

5.23 Add Layers

One function adds a layer to the active design

```
TAddLayer(tContext, pLayer)
```

The **TAddLayer** function will add a layer with the specific data of layer name, layerID, type and bias, clearances, and plane status. If the layerID is already used it TAddLayer will use the next available number. If the layer name already exists then an error is returned.

5.24 Modifying Variants

These three functions modify variants. The variant to be added, deleted or renamed, *pVariant*, is declared as a **TVariant**.

```
long TAddVariant(DbxContext* pContext, TVariant* pVariant)
long TDeleteVariant(DbxContext* pContext, TVariant* pVariant)
long TRenameVariant(DbxContext* pContext, TVariant* pVariant)
```

TAddVariant is used for adding a uniquely named Variant to a design. In the TVariant structure the name must be unique otherwise an error is returned of duplicate name. TVariant.description and TVariant.id are optional.

TDeleteVariant is used for removing a Variant from a design. The Variant must exist otherwise an error is returned. Description and id are optional.

TRenameVariant method is used to rename a Variant. To use, first get the variant data by **TGetFirstVariant** and **TGetNextVariant**. When you have the one of interest change its name and call **TRenameVariant**. Be sure to not change the id as this is used to identify the variant for renaming purposes.

5.25 Modifying Pad Styles

These three functions modify pad styles.

```
Long TCreatePadStyle(DbxContext* pContext,   // (i/o) dbx context info
          long       fromId                 // (i) existing style Id
                                                to copy from - Use 0
                                                for default
           const char*   pNewName,           // (i)   new style name
           TPadViaStyle* pStyle);            // (o)   new style return
```

The **TCreatePadStyle** function create a new pad style by copying from an existing style. The parameter fromId is the id of an existing pad style. You can use the **TGetFirstPadStyle** and **TGetNextPadStyle** functions to fetch the existing pad styles in the design. pNewName is the name for the newly create style. **TPadViaStyle*** is filled in if successful. It fills in the style id, name, hole diameter, etc. See **TPadViaStyle** type definition for more details.

```
long TDeletePadStyle(DbxContext* pContext,   // (i/o) dbx context info
                  long      padStyleId);    // (i)   pad style Id to
                                                delete
```

The **TDeletePadStyle** function is used to delete a pad style from the design. Default style or styles that are in-use in the design cannot be deleted. Use the padStyleId to specify the style to delete.

```
long TModifyPadStyle(DbxContext*   pContext, // (i/o) dbx context info
                     TPadViaStyle* pStyle);  // (i) style info to
                                                modify
```

The **TModifyPadStyle** function is used to modify a pad style. pStyle is the pad style info to modify. The following fields need to be filled in:

- styleId
- name
- holeDia
- xOffset
- yOffset
- holeStartLayer
- holeEndLayer

5.26 Modifying Pad Shapes

These two functions modify pad shapes.

```
long TAddPadShape(DbxContext*    pContext,   // (i/o) dbx context info
                  long           padStyleId, // (i) Pad style Id to
                                             add shape to
                  TPadViaShape*  pPadShape); // (i) Pad shape to add
```

The **TAddPadShape** function is used to add or modify the pad shape (for a layer) of a pad style. Note that because polygon shape uses different property attributes, this function will not add a polygon pad shape. To add/modify polygon pad shapes for a layer, use the **TAddPolyPadShape** function.

```
long TDeletePadShape(DbxContext* pContext,   // (i/o) dbx context info
                     long        padStyleId, // (i) pad style id
                     long        layerId);   // (i) layer id
```

The **TDeletePadShape** function is used to delete a pad style shape. Note that it cannot delete default shapes for the following layers: top, bottom, default non-signal, signal and plane.

5.27 Modifying Polygon Pad Shapes

These two functions are used to add regular and irregular polygon pad shape definitions.

```
long TAddPadRegularPolyShape(DbxContext*  pContext, // (i/o) dbx
                                                       context info
                long    padStyleId,                 // (i) pad style id
                TPadViaRegularPolyShape* pRegPolyShape);  // (i)
```

The **TAddPadRegularPolyShape** function is used to add or modify the regular polygon shape for a pad style. Use pRegPolyShape to pass in the number of sides, and rotation angle for the regular polygon.

```
Long TAddPadIrregularPolyShape(DbxContext* pContext, // (i/o) dbx
                                                       context info
              long    padStyleId,            // (i) pad style id
              long    layerId,               // (i) layer Id or 0 if
                                                add or modify default
                                                layer
              long    layerType,             // (i) use if layer Id is
                                                0
```

```
            long    isPourNoConn,          // (i) Prohibit Cu pour
                                               thermalizing
            long    numberOfPoints,        // (i) number of poly
                                               points
            const   TCoord* pPoints);      // (i) point array. The
                                               size of the array is
                                               numberOfPoints
```

The **TAddPadIrregularPolyShape** function is used to add or modify a Irregular polygon shape for a pad style. Note that the array of points, the pPoints parameter, defines a polygon shape. The X, Y coordinate of a point is relative to the "origin" of the polygon, not the PCB design. The coordinate is in database unit. The layerId is the layer number for the pad shape. If the layer number is 0, the layerType can be used to modify the default shape of a pad style:

```
DBX_LAYERTYPE_SIGNAL -shape for default Signal Layer
DBX_LAYERTYPE_PLANE - shape for default Plane Layer, and
DBX_LAYERTYPE_NON_SIGNAL - shape for default Non Signal layer
```

5.28 Modifying Via Styles and Shapes

The following functions are duplicates of the Pad style and shape functions except they are used for Vias.

```
long  TCreateViaStyle(DbxContext* pContext,   // (i/o) dbx context info
            long          fromId,             // (i) existing style Id
                                                 to copy from - Use 0
                                                 for default
            const char*   pNewName,           // (i) new style name
            TPadViaStyle* pStyle).            // (o) new style return

long TDeleteViaStyle(DbxContext* pContext,    // (i/o) dbx context info
            long          viaStyleId);        // (i)   via style Id to
                                                 delete

long TModifyViaStyle(DbxContext* pContext,    // (i/o) dbx context info
            TPadViaStyle* pStyle);            // (i)   style info to
                                                 modify

long TAddViaShape(DbxContext*    pContext,    // (i/o) dbx context info
            long           viaStyleId,        // (i)   Via style Id to
                                                 add shape to
            TPadViaShape*  pViaShape);        // (i)   via shape to add

Long TAddViaRegularPolyShape(DbxContext* pContext,    // (i/o) dbx
                                                        context info
            long viaStyleId,                          // (i) via style
                                                        id
            TPadViaRegularPolyShape*  pRegPolyShape); // (i) Regular
                                                        poly shape
                                                        definition

long TAddViaIrregularPolyShape(DbxContext* pContext,  // (i/o) dbx
                                                        context info
            long          viaStyleId,         // (i) via style id
            long          layerId,            // (i) layer Id or 0 if
                                                 add or modify default
```

```
                                                  layer
        long          layerType,          // (i) use if layer Id is
                                                  0
        long          isPourNoConn,       // (i) Prohibit Cu pour
                                                  thermalizing
        long          numberOfPoints,     // (i) number of poly
                                                  points
        const TCoord* pPoints);            // (i)  point array.
                                                  The size of the array
                                                  is numberOfPoints.
```

5.29 Modifying Text Styles

Below are Text style functions. These are similar to those for Pad styles.

```
long  TAddTextStyle(DbxContext*   pContext,  // (i/o) dbx context info
       long         fromId,                 // (i) existing style Id
                                                to copy from
       TTextStyle*  pStyle);                // (i/o) new text style
                                                return
```

Description: Add a new text style by copying it from an existing style.

```
long  TDeleteTextStyle(DbxContext* pContext,  // (i/o) dbx context info
         long          styleId);              // (i) style to delete
```

Description: Delete an existing text style. Use the styleId to specify the style to be removed. Note that Default style or styles that are in-used cannot be deleted.

```
long DLLX TModifyTextStyle(DbxContext* pContext,    // (i/o) dbx
                                                       context info
                           TTextStyle* pStyle);     // (i) style info
                                                       to modify
```

Description: Modify various fields of a text style. Use the styleId to specify the text style to modify. Valid fields that can be modified are:

- name
- strokeHeight
- strokePenWidth
- isTrueTypeAllowed
- isDisplayTrueType
- tTypeHeight

Note: This function returns DBX_BAD_INPUT when one of the following possible errors happened:

1) The name is a duplicate of another text style.

2) Height or width of the text are out of range

Note: This function returns DBX_ILLEGAL_OP when one of the following possible errors happened:

1) The style to be modified is a default style (i.e. (Default), (DefaultTTF))

2) The new font height and width might make the text fall out of the workspace.

6 Modifying Design Data - Schematic

This section provides a summary of the interface functions that add, modify, or delete Schematic design data. For a detailed description of each function, its arguments, or P-CAD DBX structure contents, see DBX32.H, or the appendices at the end of this manual.

Note that DBX items returned by a successful Modify operation will have a new dbId or compId database identifier. The old dbId or compId value will no longer be a valid identifier; the modified DBX item returned by the function call should be used for subsequent DBX function calls.

Before placing or modifying a symbol or wire, read section 6.22; the section discusses net connectivity issues.

To enable code sharing between DBX applications used for PCB, Schematic, and Library Manager or Library Executive, function calls and DBX items are intentionally similar where possible.

6.1 Flipping Objects

Twelve functions flip Schematic design objects. Input to the function is the item to be flipped, and the point about which the item is to be flipped. Items may be flipped using item type specific function calls, or by using a DBX **TItem** as input to **TFlipItem.**

The item to be flipped is declared as a **TArc, TAttribute, TBus, TField, TLine, TPin, TPort, TSymbol, TTable, TText, TWire** or **TItem,** respectively. The point about which to flip, *pPoint,* is declared as a **TCoord.** If the **TCoord** coordinates are specified as (-1,-1), the item is flipped about its default origin. See Section 2.8, Default Item Origins for a listing of what item property is used as a default origin.

```
TFlipArc(tContext, pPoint, pArc)
TFlipAttribute(tContext, pPoint, pAttribute)
TFlipBus(tContext, pPoint, pBus)
TFlipField(tContext, pPoint, pField)
TFlipLine(tContext, pPoint, pLine)
TFlipPin(tContext, pPoint, pPin)
TFlipPort(tContext, pPoint, pPort)
TFlipSymbol(tContext, pPoint, pSymbol)
TFlipTable(tContext, pPoint, pTable)
TFlipText(tContext, pPoint, pText)
TFlipWire(tContext, pPoint, pWire)

TFlipItem(tContext, pPoint, TItem)
```

DBX Flip functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **TFlipItem.** The item is updated to have its *isFlipped* field set appropriately.

6.2 Rotating Objects

Twelve functions rotate Schematic design objects. Input to the function is the item to be rotated, the rotation angle, and the point about which the item is to be rotated. Items may

be rotated using item type specific function calls, or by using a DBX **TItem** as input to **TRotateItem.**

The item to be rotated is declared as a **TArc, TAttribute, TBus, TField, TLine, TPin, TPort, TSymbol, TTable, TText, TWire** or **TItem**, respectively. The rotation angle angle, is declared as a long and may range from -3600 to 3600 (recall that degree values are specified in 1/10 degrees). The point about which to rotate, *pPoint,* is declared as a **TCoord.** If the **TCoord** coordinates are specified as (-1,-1), the item is rotated about its default origin. See section 2.8, Default Item Origins for a listing of what item property is used as a default origin.

```
TRotateArc(tContext, angle, pPoint, pArc)
TRotateAttribute(tContext, angle, pPoint, pAttribute)
TRotateBus(tContext, angle, pPoint, pBus)
TRotateField(tContext, angle, pPoint, pField)
TRotateLine(tContext, angle, pPoint, pLine)
TRotatePin(tContext, angle, pPoint, pPin)
TRotatePort(tContext, angle, pPoint, pPort)
TRotateSymbol(tContext, angle, pPoint, pSymbol)
TRotateTable(tContext, angle, pPoint, pTable)
TRotateText(tContext, angle, pPoint, pText)
TRotateWire(tContext, angle, pPoint, pWire)

TRotateItem(tContext, angle, pPoint, pItem)
```

DBX Rotate functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **TRotateItem**. The item *rotateAngle* or *rotation* fields are updated to reflect the item's new orientation.

6.3 Moving Objects

Thirteen functions move Schematic design objects. Input to the function is the item to be moved, and the distance in the x and y directions to move the item. Items may be moved using item type specific function calls, or by using a DBX **TItem** as input to **TMoveItem**.

The item to be moved is declared as a **TArc, TAttribute, TBus, TField, TLine, TPin, TPoint, TPort, TSymbol, TTable, TText, TWire** or **TItem**, respectively. The distances to move, *dx* and *dy,* are declared as a longs and are in database units.

```
TMoveArc(tContext, dx, dy, pArc)
TMoveAttribute(tContext, dx, dy, pAttribute)
TMoveBus(tContext, dx, dy, pBus)
TMoveField(tContext, dx, dy, pField)
TMoveLine(tContext, dx, dy, pLine)
TMovePin(tContext, dx, dy, pPin)
TMovePoint(tContext, dx, dy, pPoint)
TMovePort(tContext, dx, dy, pPort)
TMoveSymbol(tContext, dx, dy, pSymbol)
TMoveTable(tContext, dx, dy, pTable)
TMoveText(tContext, dx, dy, pText)
TMoveWire(tContext, dx, dy, pWire)

TMoveItem(tContext, dx, dy, pItem)
```

DBX Move functions return a DBX error return status, and an updated DBX item, either in

an item type specific DBX item structure, or as a **TItem** from **TMoveItem.** The item location information is updated to reflect the item's new location after the Move operation.

6.4 Deleting Objects

Seventeen functions delete Schematic design objects. Input to the function is the item to be deleted. Items may be deleted using item type specific function calls, or by using a DBX **TItem** as input to **TDeleteItem**.

The item to be deleted is declared as a **TArc, TAttribute, TBus, TClassToClass, TField, TLine, TNet, TNetClass, TPin, TPoint, TPort, TSymbol, TTable, TText, TWire** or **TItem**, respectively.

```
TDeleteArc(tContext, pArc)
TDeleteAttribute(tContext, pAttribute)
TDeleteBus(tContext, pBus)
TDeleteClassToClass(tContext, pClassToClass)
TDeleteField(tContext, pField)
TDeleteLine(tContext, pLine)
TDeleteNet(tContext, pNet)
TDeleteNetClass(tContext, pNetClass)
TDeleteNetClassNet(tContext,NetClassId, pNet)
TDeletePin(tContext, pPin)
TDeletePoint(tContext, pPoint)
TDeletePort(tContext, pPort)
TDeleteSymbol(tContext, pSymbol)
TDeleteTable(tContext, pTable)
TDeleteText(tContext, pText)
TDeleteWire(tContext, pWire)

TDeleteItem(tContext,pItem)
```

DBX Delete functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **TDeleteItem**. The item is indicated as no longer being a valid Schematic design item by having its DBID property set equal to zero.

6.5 Highlighting Objects

Twenty-nine functions highlight and unhighlight Schematic design objects. Input to each function is the item to be highlighted or unhighlighted, and a highlight color for highlight functions. Items may be highlighted or unhighligted using item type specific function calls, or by using a DBX **TItem** as input to **THighlightItem** and **TUnHighlightItem**.

All items in the active design may be unhighlighted using the **TUnHighlightAll** function.

All items in a net may be highlighted or unhighlighted using the **THighlightNet** and **TUnHighlightNet** functions, respectively.

The item to be highlighted or unhighlighted is declared as a **TArc, TAttribute, TBus, TField, TLine, TNet, TPin, TPoint, TPort, TSymbol, TTable, TText, TWire** or **TItem**, respectively. The highlight color color is declared as a long value and specifies highlight colors 1 through 20. The 20 colors associated with the highlight color constants are listed

in Appendix A, P-CAD DBX Data Constants under DBX Color Index Types.

```
THighlightArc(tContext, color, pArc)
THighlightAttribute(tContext, color, pAttribute)
THighlightBus(tContext, color, pBus)
THighlightField(tContext, color, pField)
THighlightLine(tContext, color, pLine)
THighlightNet(tContext, color, pNet)
THighlightPin(tContext, color, pPin)
THighlightPoint(tContext, color, pPoint)
THighlightPort(tContext, color, pPort)
THighlightSymbol(tContext, color, pSymbol)
THighlightTable(tContext, color, pTable)
THighlightText(tContext, color, pText)
THighlightWire(tContext, color, pWire)

TUnHighlightArc(tContext, pArc)
TUnHighlightAttribute(tContext, pAttribute)
TUnHighlightBus(tContext, pBus)
TUnHighlightField(tContext, pField)
TUnHighlightLine(tContext, pLine)
TUnHighlightNet(tContext, pNet)
TUnHighlightPin(tContext, pPin)
TUnHighlightPoint(tContext, pPoint)
TUnHighlightPort(tContext, pPort)
TUnHighlightSymbol(tContext, pSymbol)
TUnHighlightTable(tContext, pTable)
TUnHighlightText(tContext, pText)
TUnHighlightWire(tContext, pWire)

THighlightItem(tContext, color, pItem)
TUnHighlightItem(tContext, pItem)
TUnHighlightAll(tContext)
```

DBX Highlight functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **THighlightItem** or **TUnHighlightItem.** Individual items highlighted or unhighlighted are returned having their *isHighlighted* field updated to indicate its highlight color. (*isHighlighted* is 0 to indicate that the item is not highlighted and ranges 1 through 20 to specify which highlight color is in use.)

6.6 Modifying Object Properties

Fourteen functions modify Schematic design objects properties. Input to the function is the item to be modified. Items may be modified using item type specific function calls, or by using a DBX **TItem** as input to **TModifyItem.**

The modification to be performed is specified by setting one or more of the DBX item properties prior to calling the Modify function. The properties that may be modified by a DBX program are specific to the type of item being modified and are listed below. All modifications must conform to P-CAD Schematic design rules. For example, component *refDes* values must be unique and contain no invalid characters; a location or rotation which would move the item outside the design workspace will return a DBX error value of DBX_ITEM_OUTSIDE_WORKSPACE; moving an item to an invalid layer will return an error value of DBX_INVALID_LAYER.

The item to be modified is declared as a **TArc, TAttribute, TBus, TField, TLine, TNet, TNetClass, TPin, Tpoint, TPort, TSymbol, TText, TWire** or **TItem,** respectively.

The modifiable properties for each DBX item are listed below:

Arc: *width, radius, centerPt, startAng, sweepAng*
Attribute: *value, refPoint, style, justification, visiblity, formula, comment, units*
Bus: *startPt, endPt, busName, isNameVisible*
Field: *refPoint, justification, textStyleId, layerId*
Line: *width, startPt, endPt, lineType*
Net: *netName*
NetClass: *netClassName*
Component Pin: *outsideStyle, outsideEdgeStyle, insideStyle, insideEdgeStyle*
Free Pin: *outsideStyle, outsideEdgeStyle, insideStyle, insideEdgeStyle, sympinNumber, location*
Point: *location*
Port: *portType, pinLength, refPoint*
Symbol: *refDes, altType, location*
Text: *text, refPoint, textStyleId, justPoint*
Wire: *startPt, endPt, isNameVisible, width*

```
TModifyArc(tContext, pArc)
TModifyAttribute(tContext, pAttribute)
TModifyBus(tContext, pBus)
TModifyField(tContext, pField)
TModifyLine(tContext, pLine)
TModifyNet(tContext, pNet)
TModifyNetClass(tContext, pNetClass)
TModifyPin(tContext, pPin)
TModifyPoint(tContext, pPoint)
TModifyPort(tContext, pPort)
TModifySymbol(tContext, pSymbol)
TModifyText(tContext, pText)
TModifyWire(tContext, pWire)

TModifyItem(tContext, pItem)
```

DBX Modify functions return a DBX error return status, and an updated DBX item, either in an item type specific DBX item structure, or as a **TItem** from **TModifyItem.** The item properties are updated to reflect the item status after the Modify call. Item properties not legal for modification that were modified in the DBX item prior to the function call are overwritten with valid data representing the item's current state after the Modify call. The function **TModifyPin** is only applicable to free pins only. Modifing a symbol's refDes may result in components being created and deleted. If you change refDes to an existing component, it must be of the same type and the gate must be unused.

Note: For information on modifying component, symbol and net attributes, see sections 6.13, 6.14 and 6.15, respectively.

Note: For information on net conectivity issues, see section 6.22.

6.7 Placing Objects

Eleven functions place Schematic design objects in an active Schematic design. Input to the function is the item to be placed. There is no generic **TPlaceItem** form.

The properties for the item to be placed is specified by setting one or more of the DBX item properties prior to calling the Place function. The properties that must be specified during placement are specific to the type of item being placed and are listed below. All items to be placed, and the properties of those items, must conform to P-CAD Schematic design rules. For example, component *refDes* values must be unique and contain no invalid characters; a location or rotation which would place the item outside the design workspace will return a DBX error value of DBX_ITEM_OUTSIDE_WORKSPACE; placing an item on an invalid layer will return an error value of DBX_INVALID_LAYER.

The item to be placed is declared as a **TArc, TAttribute, TBus, TField, TLine, TPin, TPoint, TPort, TSymbol, Ttext** and **TWire,** respectively.

The specifiable properties for each DBX item are listed below. Note that except for *component.libraryName, component.typeName,* and *point.infoText,* these are exactly the properties valid for DBX Modify operations. Note also that all properties listed must be specified to place a new design item.

Arc: *width, radius, centerPt, startAng, sweepAng, layerId*
Attribute: *type, value, refPoint, textStyleId, justPoint, isVisible, layerId*
Bus: *startPt, endPt, busName, layerId*
Field: *fieldKeyType, refPoint, textStyleId, layerId, justPoint*
Line: *width, startPt, endPt, style, layerId*
Pin: *refPoint, style, pinNumber, layerId, pinLength*
Point: *pointType, x, y (location), layerId; textInfo, ruleCategory, ruleType (InfoPoint only)*
Port: *portType, pinLength, refPoint, layerId, netId*
Symbol: *compType, refDes, partNumber, altType, refPoint, libraryName, layerId*
Text: *text, refPoint, textStyleId, justPoint, layerId*
Wire: *startPt, endPt, netId, layerId, width*
DesignInfo*: isModified*

Note: When placing a symbol, *symbol.libraryName*, if specified, must include the complete library path and file name (e.g. c:\P-CAD\demo.lib). If not specified by leaving the input property field blank, the open libraries will be searched in the current open library order for the input component type.

```
TModifyDesignInfo(tContext, pDesignInfo)
TPlaceArc(tContext, pArc)
TPlaceAttribute(tContext, pAttribute)
TPlaceBus(tContext, pBus)
TPlaceField(tContext, pField)
TPlaceLine(tContext, pLine)
TPlacePin(tContext, pPin)
TPlacePoint(tContext, pPoint)
TPlacePort(tContext, pPort)
TPlaceSymbol(tContext,pSymbol)
TPlaceText(tContext, pText)
TPlaceWire(tContext, pWire)
```

DBX Place functions return a DBX error return status, and an updated DBX item. The item properties are updated to reflect the item status after the Place call. Item properties not legal to be specified during placement that were specified in the DBX item prior to the function call are overwritten with valid data representing the item's current state after the

Place call. When placing a symbol, if the component doesn't exist it will be created otherwise the compType field must be the same as the existing component.

Note: For information on net connectivity issues concerning placing a wire or symbol, see section 6.22.

6.8 Creating and Deleting Nets

Two functions create and delete a Schematic net. Input to the **TCreateNet** function is a DBX **TNet** item. The *net.netName* field must specify the name of the new net. The name must be a valid netname, and not be a duplicate of an existing net. All other **TNet** properties are ignored.

A Schematic net must contain no design items (lines, arcs, vias) and have no net nodes to be deleted. See Sections 6.4, Deleting Objects, and 6.9, Adding and Deleting Net Nodes for information on how to delete net items and net nodes.

Note: When deleting a net, net objects (using **GetFirstNetItem** and **GetNextNetItem**) should be deleted first, then delete the net nodes (using **GetFirstNetNode** and **GetNextNetNode**). It is not necessary to delete all net attributes before deleting a net.

```
TCreateNet(tContext, pNet)
TDeleteNet(tContext, pNet)
```

DBX **TCreateNet** and **TDeleteNet** functions return a DBX error return status, and an updated DBX net item. If successful, a created **TNet** will be returned and indicate a valid net number in the *net.netNumber* field. A deleted net will be indicated by a *net.netNumber* field value of zero.

6.9 Adding and Deleting Net Nodes

Two functions add and delete Schematic net nodes. Input to these functions is the net number, and the pin to be added or deleted. *netId* is declared as a long, and must be a valid net number. *pItem* is the node to be added or deleted from the net, and is declared as a **TItem** with the pin structure filled in.

```
TAddNetNode(tContext, netId, pItem)
TDeleteNetNode(tContext, netId, pItem)
```

DBX **TAddNetNode** and **TDeleteNetNode** functions return a DBX error return status, and an updated DBX **TItem** with the pin structure filled in. If a pin has been successfully added to a net, the pin *netId* property will be updated to reflect the net number of the net to which it has been added. If a pin has been successfully deleted from a net, the pin *netId* property will be set to zero, indicating it belongs to no nets.

6.10 Creating and Deleting NetClasses

Two functions create and delete a Schematic design net class. Input to the **TCreateNetClass** function is a DBX **TNetClass** item. The *netClass.netClassName* field must specify the name of the new net class. The name must be a valid netClassName, and not be a duplicate of an existing netClass. All other **TNet** properties are ignored.

```
TCreateNetClass(tContext, pNetClass)
TDeleteNetClass(tContext, pNetClass)
```

DBX **TCreateNetClass** and **TDeleteNetClass** functions return a DBX error return status, and an updated DBX net class item. If successful, a created **TNetClass** will be returned and indicate a valid net class number in the *netClass.numberOfNets* field.

6.11 Adding and Deleting NetClassNets

Two functions add and delete a Schematic design net class net. Input to the **TCreateNetClassNet** function is a DBX **netClassId** and a **TNet** item. The **netClassId** is declared as a long and must be a valid NetClass number. **pNet** is the net to be added or deleted from the net, and is declared as a **TNet** with the net structure filled in.

```
TAddNetClassNet(tContext, netClassID, pNet)
TDeleteNetClassNet(tContext, netClassId, pNet)
```

DBX **TAddNetClassNet** and **TDeleteNetClassNet** functions return a DBX error return status and the number of nets in the net class is incremented or decremented.

6.12 Creating and Deleting ClassToClass Rules

Two functions create and delete a Schematic design class to class rules. Input to the **TCreateClassToClass** function is a DBX **TClassToClass** item.

```
TCreateClassToClass(tContext, pClassToClass)
TDeleteClassToClass(tContext, pClassToClass)
```

DBX **TCreateClassToClass** and **TDeleteClassToClass** functions return a DBX error return status.

6.13 Modifying Component Attributes

Three functions modify component attributes. Input to these functions is the component *refdes,* which you declare as a string variable in Visual Basic, and as a Char* in C or C++, and the attribute to be added, modified, or deleted, *pAttribute*, declared as a **TAttribute.**

```
TAddCompAttribute(tContext, refDes, pAttribute)
TModifyCompAttribute(tContext, refDes, pAttribute)
TDeleteCompAttribute(tContext, refDes, pAttribute)
```

The following **TAttribute** properties are used when adding an attribute to a component: *type, value*. The component attribute is added, modified, or deleted to/from all symbols of that component instance. Use **TModifySymbolAttribute** to modify the graphical representation of each individual symbol.

The following **TAttribute** properties may be modified using the **TModifyCompAttribute** function: *value*.

DBX Modify Component Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified attributes will be returned with all attribute properties updated. A deleted attribute will be returned with its DB ID set to zero.

6.14 Modifying Symbol Attributes

One function is available to modify a symbol's attributes. Input to this function is the symbol object's reference designator (e.g. U1:A) and the attribute to be modified declared as a **TAttribute**.

```
status = TModifySymbolAttribute(tContext,SymbolRefDes, TAttribute)
```

Only the graphical representation of a symbol is modifiable by this function, including *refpoint, textstyle, justification, and visibility*. The user can not change its value. Use **TModifyCompAttribute** to change a symbol's attribute value (note that this value is shared by all symbols in the component).

DBX Modify Symbol Attribute function returns any of the following DBX error return status (DBX_ITEM_NOT_FOUND if the target symbol is not found and DBX_MODIFY_ERROR for others). If successful, the function returns an updated DBX **TAttribute** item. Modified attributes will be returned with all attribute properties updated.

6.15 Modifying Net Attributes

Three functions modify net attributes. Input to these functions is the net number, *netId,* which you declare as a long, and the attribute to be added, modified, or deleted, *pAttribute,* declared as a **TAttribute.**

```
TAddNetAttribute(tContext, netId, pAttribute)
TModifyNetAttribute(tContext, netId, pAttribute)
TDeleteNetAttribute(tContext, netId, pAttribute)
```

The following **TAttribute** properties are used when adding an attribute to a net or modifying an existing net attribute: *type,* and *value.* Net attributes are not visible. All other properties are ignored when adding or modifying a net attribute.

DBX Modify Net Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified attributes will be returned with all attribute properties updated. A deleted attribute will be returned with its DB ID set to zero.

6.16 Modifying Design Attributes

Three functions modify design attributes. The attribute to be added, modified, or deleted, *pAttribute,* is declared as a **TAttribute.**

```
TAddDesignAttribute(tContext, pAttribute)
TModifyDesignAttribute(tContext, pAttribute)
TDeleteDesignAttribute(tContext, pAttribute)
```

The following **TAttribute** properties are used when adding an attribute to a design or

modifying an existing design attribute: *type, value, formula, comment and units.* All other properties are ignored when adding or modifying a design attribute.

DBX Modify Design Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified attributes will be returned with all attribute properties updated. A deleted attribute will be returned with its DB ID set to zero.

6.17 Modifying NetClass Attributes

Three functions modify NetClass attributes. The attribute to be added, modified, or deleted, *pAttribute,* is declared as a **TAttribute.**

```
TAddNetClassAttribute(tContext, netClassId, pAttribute)
TModifyNetClassAttribute(tContext, netClassId, pAttribute)
TDeleteNetClassAttribute(tContext, netClassId, pAttribute)
```

The following **TAttribute** properties are used when adding a NetClass attribute to a design or modifying an existing NetClass attribute: *type, value, formula, comment and units.* All other properties are ignored when adding or modifying a NetClass attribute.

DBX Modify NetClass Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified NetClass attributes will be returned with all attribute properties updated. A deleted NetClass attribute will be returned with its DB ID set to zero.

6.18 Modifying ClassToClass Attributes

Three functions modify ClassToClass attributes. The attribute to be added, modified, or deleted, *pAttribute,* is declared as a **TAttribute.**

```
TAddClassToClassAttribute(tContext, netClassId1, netClassId2,
                          pAttribute)
TModifyClassToClassAttribute(tContext, netClassId1,
                          netClassId2, pAttribute)
TDeleteClassToClassAttribute(tContext, netClassId1,
                          netClassId2, pAttribute)
```

The following **TAttribute** properties are used when adding a ClassToClass attribute to a design or modifying an existing ClassToClass attribute: *type, value, formula, comment and units.* All other properties are ignored when adding or modifying a ClassToClass attribute.

DBX Modify ClassToClass Attribute functions return a DBX error return status, and an updated DBX **TAttribute** item. Added or modified ClassToClass attributes will be returned with all attribute properties updated. A deleted ClassToClass attribute will be returned with its DB ID set to zero.

6.19 Saving a Design

The following function is used to save the active design.

```
TSaveDesign(tContext)
```

The active design will be written to disk in the same manner as if the File Save command was chosen from the P-CAD application menu.

6.20 Selecting Print Jobs

Three functions select or deselect the print jobs in the active design. Selected print jobs are those that are output during the next print operation. Print jobs are specified in the *jobName* string.

```
TSelectPrintJob(tContext, jobName)
TDeselectPrintJob(tContext, jobName) TSelectAllPrintJob(tContext)
```

the **TSelectPrintJob** and **TDeselectPrintJob** functions will select or deselect the print job specified by its name. The **TSelectAllPrintJobs** function will select all the print jobs in the active design.

6.21 Printing Print Jobs

Two functions print the print jobs of the active design to the current printer. Print jobs are specified in the *jobName* string.

```
TOutputPrintJobByName(tContext, jobName)
TOutputSelectedPrintJobs(tContext)
```

The **TOutputPrintJobByName** function will print the job specified by its name, regardless of whether or not it has been selected for output. The **TOutputSelectedPrintJobs** function will print jobs that are selected for output.

6.22 Net Connectivity Issues When Modifying Objects

When modifying an object in the editor you receive visual feedback about the operations you are performing on object. This is extremely important when dealing with object that have net connectivity (i.e. symbols, wires). When you place, flip, rotate, move or otherwise modify an object you have a visual cue as what happens to the objects net connectivity.

Through DBX, however many times these visual cues will not be apparent to the DBX programmer. For example, if you flip a symbol so that one or more of its pins intersect a wire DBX has no way of knowing whether you wanted those pins to be connected to the wire. It would be inefficient for the DBX programmer to have to check every net to see if any of the pins on the symbol you flipped were added to a net.

DBX's Approach to Adding, Modifying Wires

When placing a wire through DBX, you must do so using an existing net ID.

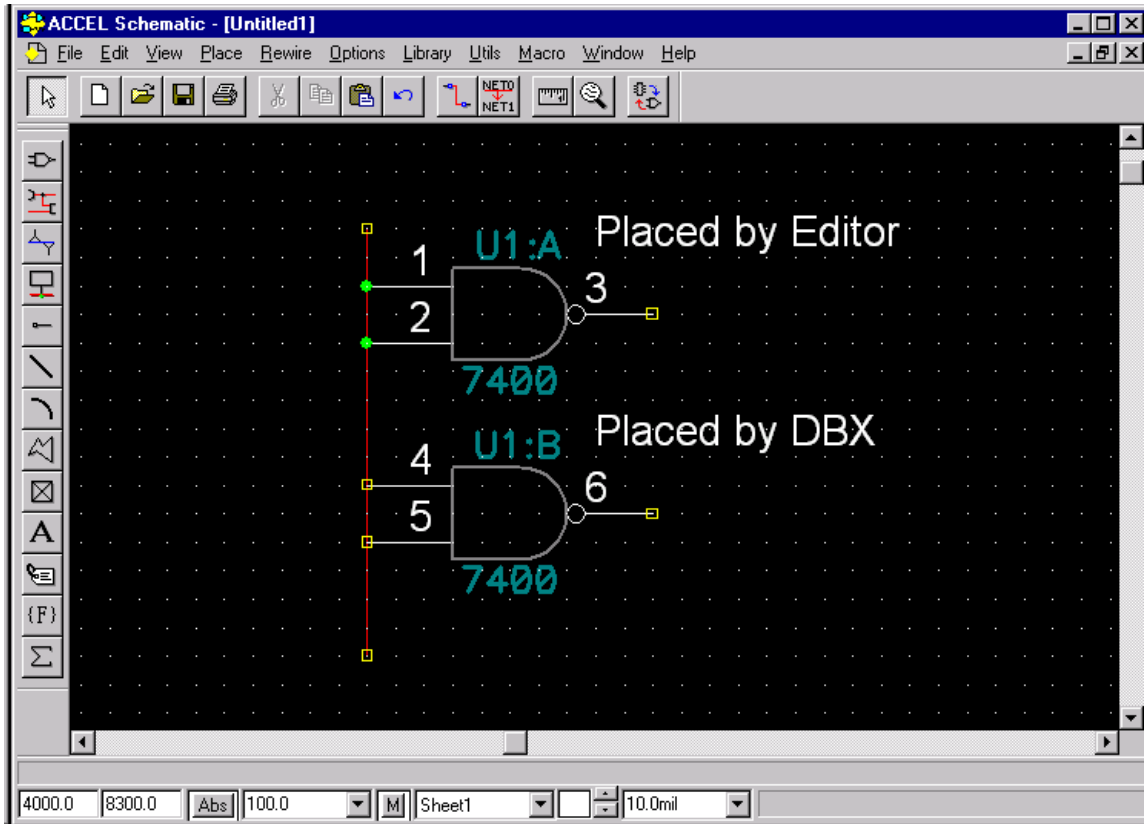
The following C++ example shows how to create a net and then place a wire using that net ID:

…

```
TNet aNet;
TWire aWire;
strcpy(aNet.netName,"NET1");
status = TCreateNet(tContext, aNet);          // create the net
if(status != DBX_OK)
{
    // return error message here
    …
}
aWire.netId = aNet.netId;                  // assign the wire to the net
// set the rest of the wire properties
…
TPlaceWire(tContext,aWire);                // create the wire
```

You could use the net ID to place more wires in the same net, however even if two of them intersected the visual cues (i.e. the Open Ends) will not be resolved.

The following figure shows an example of the differences between parts placed through the editor and DBX.



DBX's Approach to Adding, Modifying Symbols & Pins

Wires are just the visible representation that a symbol or pin has been added to a net. In the editor to add a symbol or pin to an existing wire (and therefore to its underlying net) you would either place or move one of the symbols pins over the wire. Using DBX the same method is not applicable. To add a symbol so that it belongs to a net: First determine where on the wire you want to place the symbol. Second place the part using

TPlaceSymbol. Then, add the corresponding pin to the net using the **TAddNetNode** function.

6.23 Add Sheets

One function adds a layer to the active design

```
TAddLayer(tContext, pLayer)
```

The **TAddSheet** function will add a layer with the specific data of layer name, layerID, type and bias, clearances, and plane status. If the layerID is already used it **TAddSheet** will use the next available number. If the layer name already exists then an error is returned.

6.24 Modifying Variants

These three functions modify variants. The variant to be added, deleted or renamed, *pVariant*, is declared as a **TVariant**.

```
long TAddVariant(DbxContext* pContext, TVariant* pVariant)
long TDeleteVariant(DbxContext* pContext, TVariant* pVariant)
long TRenameVariant(DbxContext* pContext, TVariant* pVariant)
```

TAddVariant is used for adding a uniquely named Variant to a design. In the TVariant structure the name must be unique otherwise an error is returned of duplicate name. TVariant.description and TVariant.id are optional.

TDeleteVariant is used for removing a Variant from a design. The Variant must exist otherwise an error is returned. Description and id are optional.

TRenameVariant method is used to rename a Variant. To use, first get the variant data by **TGetFirstVariant** and **TGetNextVariant**. When you have the one of interest change its name and call **TRenameVariant**. Be sure to not change the id as this is used to identify the variant for renaming purposes.

6.25 Modifying Text Styles

Below are Text style functions. These are similar to those for PCB Pad styles.

```
long  TAddTextStyle(DbxContext*   pContext,  // (i/o) dbx context info
       long         fromId,                  // (i) existing style Id
                                                to copy from
       TTextStyle*  pStyle);                 // (i/o) new text style
                                                return
```

Description: Add a new text style by copying it from an existing style.

```
long  TDeleteTextStyle(DbxContext* pContext,  // (i/o) dbx context info
         long          styleId);              // (i) style to delete
```

Description: Delete an existing text style. Use the styleId to specify the style to be removed. Note that Default style or styles that are in-used cannot be deleted.

```
long DLLX TModifyTextStyle(DbxContext* pContext,    // (i/o) dbx
                                                       context info
                           TTextStyle* pStyle);     // (i) style info
                                                       to modify
```

Description: Modify various fields of a text style. Use the styleId to specify the text style to modify. Valid fields that can be modified are:

- name
- strokeHeight
- strokePenWidth
- isTrueTypeAllowed
- isDisplayTrueType
- tTypeHeight

Note: This function returns DBX_BAD_INPUT when one of the following possible errors happened:

3) The name is a duplicate of another text style.

4) Height or width of the text are out of range

Note: This function returns DBX_ILLEGAL_OP when one of the following possible errors happened:

5) The style to be modified is a default style (i.e. (Default), (DefaultTTF))

6) The new font height and width might make the text fall out of the workspace.

7 Introduction to Library DBX

For version 13 a DBX interface has been added to the Component Library Manger and Library Executive that allow it to perform many operations using existing objects common to both Schematic and PCB. All functions operate on *static* library objects, including components, symbols and patterns. Because the objects are shared across all three products many fields are not applicable to Component Library Manger.

To enable code sharing between DBX applications used for PCB, Schematic, and Library Manager or Library Executive, function calls and DBX items are intentionally similar where possible.

7.1 Opening and Closing a Component Library

You establish a connection between your user program and the P-CAD Library Manager or Library Executive by opening a component library. To open the library, you use the function **TOpenLibrary,** providing some basic information: the version of the P-CAD DBX software you are using, and the type of language you are using to write your user program, and the library named by *libName*. *libName* must be the full path name to the library. **TOpenLibrary** returns the P-CAD-supplied **TContext** structure *tContext,* which describes the connection you just established. This **TContext** structure must be used in all subsequent DBX function calls to identify the DBX conversation. For example, to open the Component Library *libName*:

```
TOpenLibrary (DBX_LANGUAGE,DBX_VERSION, libName,tContext)
```

DBX_VERSION and DBX_LANGUAGE are constants provided by P-CAD DBX and may be used exactly as written here. Also defined in DBX32.H and DBX32.BAS is *tContext,* a global variable which may also be used exactly as written here, and in subsequent function calls, to accept and maintain the conversation context information. The 3rd argument specifies the full path of the P-CAD Component Library to open.

To end your DBX session, use **TCloseLibrary.** Note that *tContext* is the global structure returned by **TOpenLibrary**.

```
TCloseLibrary (tContext, "")
```

Note: The second parameter of the function **TCloseLibrary** is a string pointer and is currently ignored. Passing an empty string ("") is the best way to insure future compatibility with this function.

A DBX conversation with P-CAD Library Manager or Library Executive may also be terminated by clicking Cancel on the dialog which is presented in the design window when your DBX program begins executing. Using **TCloseLibrary** is the preferred method, but the Cancel dialog is useful should your DBX program terminate abnormally or find itself in an infinite loop from which you cannot exit. The Cancel dialog will return P-CAD Library Manager or Library Executive to a state from which you can safely continue.

7.2 Extracting Components

Three functions return component data stored in the current opened library. Component data is returned in the DBX item which you declared as **TComponent**.

```
TGetFirstComponent (tContext, TComponent)
TGetNextComponent (tContext, TComponent)
TGetCompByType(tContext, CompType,  TComponent)
```

The function **TGetCompByType** allows the user to get a specified component from the current component library. *CompType* is used to specify the component name.

Component functions **TGetFirstComponent**, **TGetNextComponent** and **TGetCompByType** return, component type, pattern name, library name, number of pads, number of pins, number of parts, is alpha, is heterogeneous, and connection type in a **TComponent** data item.

7.3 Extracting Patterns

Three functions return pattern data stored in the current opened library. Pattern data is returned in the DBX item which you declared as type **TPattern**.

```
TGetFirstPattern (tContext, TPattern)
TGetNextPattern (tContext, TPattern)
TGetPatternByName(tContext, PatternName,  TPattern)
```

The function **TGetPatternByName** allows the user to get a specified pattern from the current component library. *PatternName* is used to specify the pattern name.

Pattern functions **TGetFirstPattern**, **TGetNextPattern** and **TGetPatternByName** return pattern name in a **TPattern** data item.

7.4 Extracting Symbols from a given Component Type

Two functions return symbol data from a user named component stored in the current open library. Input to the **TGetFirstCompSymbol** function is the component type name. Symbol data is returned in the DBX item which you declared as type **TSymbol**. *CompType* is used to specify the component name.

```
TGetFirstCompSymbol (tContext, CompType, TSymbol)
TGetNextCompSymbol (tContext, TSymbol)
```

TGetNextCompSymbol will let the user cycle through Normal type symbols first, then IEEE type, then DeMorgan type. On Homogeneous components, only the first part is returned regardless of how many parts the component has.

Component symbol functions **TGetFirstCompSymbol** and **TGetNextCompSymbol** return symbol name, number of pins, part number (heterogeneous components only), and alt type (Normal, IEEE, DeMorgan),

7.5 Extracting Symbols

Three functions return symbol data stored in the current opened library. Symbol data is returned in the DBX item which you declared as type **TSymbol**.

```
TGetFirstSymbol (tContext, TSymbol)
TGetNextSymbol (tContext, TSymbol)
TGetSymbolByName(tContext, SymbolName,  TSymbol)
```

The function **TGetSymbolByName** allows the user to get a specified symbol from the current component library. *SymbolName* is used to specify the symbol name.

Symbol functions **TGetFirstSymbol**, **TGetNextCompSymbol** and **TGetSymbolByName** return symbol name, number of pins, and alt type (Normal, IEE, DeMorgan),

7.6 Extracting Pins from a given Component Type

Two functions return component pin data from a given component type. Pin data is returned in the DBX item which you declared as type **TPin**. *CompType* is used to specify the component name.

```
TGetFirstCompPin (tContext, CompType, TPin)
TGetNextCompPin (tContext, TPin)
```

Component Pin functions **TGetFirstCompPin** and **TGetNextCompPin** return, in the compPin sub-structure, symbol pin number, pin designator, pin name, gate equivalence, pin equivalence, and electrical type.

7.7 Extracting Pins from a given Symbol Name

Two functions return symbol pin data from a given symbol name. Pin data is returned in the DBX item which you declared as type **TPin**. *SymbolName* is used to specify the symbol name.

```
TGetFirstSymbolPin (tContext, SymbolName, TPin)
TGetNextSymbolPin (tContext, TPin)
```

Symbol Pin functions **TGetFirstSymbolPin** and **TGetNextSymbolPin** return outside style, outside edge style, inside style, inside edge style, pin length. In the compPin sub-structure the symbol pin number, pin designator, pin name, gate equivalence, pin equivalence, gate number, and electrical type are returned.

7.8 Extracting Attributes from a given Component Type

Two functions return attribute data from a given component type. Attribute data is returned in the DBX item which you declared as type **TAttribute**. *CompType* is used to specify the component name.

```
TGetFirstCompAttribute(tContext, CompType,TAttribute)
TGetNextCompAttribute(tContext, TAttribute)
```

Component Attribute functions **TGetFirstCompAttribute** and **TGetNextCompAttribute** return type and value.

7.9 Modifying Component Attributes

Three functions modify component attributes. Inputs to these functions are the component item and the attribute to be added, modified or deleted, *TAttribute*, declared as a **TAttribute**. *CompType* is used to specify the component name.

```
TAddCompAttribute(tContext, CompType, TAttribute)
TModifyCompAttribute(tContext, CompType, TAttribute)
TDeleteCompAttribute(tContext, CompType, TAttribute)
```

The following **TAttribute** properties are used when adding an attribute to a component: type, value. Reference point, text style, justification, and visibility are subject to editors' defaults when instantiated.

The value of the **TAttribute** property may be modified using the **TModifyCompAttribute** function.

DBX Modify Component Attribute functions return any of the following DBX error return status (DBX_COMPONENT_NOT_FOUND if the system can not find the specified component type and DBX_CREATE_ERROR if errors during adding, DBX_MODIFY_ERROR if error during modification and DBX_DELETE_ERROR if error during deletion). If successful, the functions return an updated DBX **TAttribute** item. Added or modified attributes will be returned with all attribute properties updated. A deleted attribute will be returned with its database id set to zero.

7.10 Copying Components, Symbols, or Pattern Information

Three functions copy an instance of component, symbol or pattern of the component library. A valid context must also be supplied to these functions. The library used to create the context does not need to be either the source library or the destation library. The source and destation libraries can be the same as long as the destation component name, symbol name, or pattern name is different then the source.

Inputs to the Copy Component function are the names of the source component, source library, destination component and destination library. The destination library must already exist and can not contain a component with the same name as you destination component.

```
TCopyComponent(tContext,sCompType,sLibName,dCompType,dLibName)
```

DBX Copy Component function returns any of the DBX error return status. If successful, the function returns DBX_OK to indicate the new component has been copied.

Inputs to the Copy Symbol function are the names of the source symbol, source library, destination symbol and destination library. The destination library must already exist and can not contain a symbol with the same name as you destination symbol.

```
TCopySymbol(tContext, sSymName, sLibName, dSymName, dLibName)
```

DBX Copy Symbol function returns any of the DBX error return status. If successful, the function returns DBX_OK to indicate the new symbol has been copied

Inputs to the Copy Pattern function are the source pattern, source library, destination pattern and destination library. The destination library must already exist and can not contain a pattern with the same name as you destination pattern.

```
TCopyPattern(tContext, sPatName, sLibName, dPatName, dLibName)
```

DBX Copy Pattern function returns any of the DBX error return status. If successful, the function returns DBX_OK to indicate the new pattern has been copied.

7.11 Opening, Closing and Saving a Component

Three functions are used to open, close and save a component. *compType* is the name of the component to open. A component must be opened before it can be modified, saved or queried. Only one component can be open at a time and should be closed when it is no longer needed.

```
TOpenComponent(tContext,compType)
TCloseComponent(tContext)
TSaveComponent(tContext)
```

8 Building a User Program

A similar process is used to develop a P-CAD DBX program in either a Visual Basic or C environment. There are 5 basic steps:

1) Determine what it is you want your program to do. This could be to generate a report, display a list of items having particular characteristics, or check a design for validity given some particular criteria.
2) Design and build your user interface using your development environment tools to retrieve any information needed from the user, e.g. a netname, a clearance tolerance, or a filename for output.
3) Include DBX32.H for 'C' or C++ programs, or DBX32.BAS for Visual Basic, in your project.
4) Include DBXUTILS.H for 'C' or C++ programs, or DBXUTILS.BAS for Visual Basic, for unit conversions, if needed.
5) Add your Schematic and PCB DBX extraction function calls and data processing code:
   a) TOpenDesign call
   b) TGet* and data processing and output code
   c) TCloseDesign call
   d) Error Handling

Build and execute your program, or if using Visual Basic, execute your program interpretively.

P-CAD DBX specifics of developing an application using Visual Basic or Visual C++ are detailed below.

8.1 DBX32.H and DBX32.BAS

The DBX32.BAS (for Visual Basic programs) and DBX32.H (for C programs) files are the cornerstone to integration between user programs, the P-CAD DBX function libraries, and the programming language you use to develop your user programs. These files provide all DBX function prototypes and data structures needed to use the P-CAD DBX interface. Function input and output argument types and structures which are not part of the native language are predefined in these files and may be used directly, as-is, without modification in your programs to access the P-CAD DBX function library.

These development files are used to develop DBX applications in a 32 bit development environment, for example Visual C++ 4.0, Borland 4.0, or Visual Basic 4.0.

8.2 Unit Conversion Utilities

DBXUTILS.H and DBXUTILS.BAS include utility functions which will convert numbers between database units, mils, millimeters, and strings. These functions provide a convenient mechanism to transform DBX output to printable or displayable strings, or to convert user input to database values. You may include this file in your project and reference the functions directly.

8.3 Status and Error values

All P-CAD DBX functions return a status value. These are integer values defined in DBX32.H and DBX32.BAS as mnemonic constants. A return value of zero (DBX_OK) indicates that the function completed normally with no exceptions. Your programs should check this status value after each function call, and handle any non-zero status returns appropriately. Note that many non-zero values are not fatal and are to be expected during normal program execution. DBX_NO_MORE_ITEMS, is a good example. This is a value which should be expected after the last item has been returned from a GetFirst* or GetNext* call.

The DBXUTILS files also have a function which provide a textual string given a DBX status value. This is useful for printing or displaying status messages at run-time.

8.4 Using Visual Basic as a Development Environment for DBX

8.4.1 Overview

Visual Basic provides an environment where you graphically create a Windows application and user interface using Visual Basic tools, and add your own applications code in the Visual Basic language to accomplish your specific programming task. Visual Basic uses projects, forms, and Visual Basic source programs to create a Windows application. A project is composed of forms and source code, where a form is a user interface "palette" on which you develop your user interface. Visual Basic source code, which you either write yourself or has been provided to you, is then associated with parts of the user interface you just designed, and executed depending on the actions and state of the user interface. An example is a program which gets invoked when a button on the form is selected.

Source programs are either developed within a form (maintained in the project as a separate file named <form name>.FRM) or as separate files with the .BAS extension.

This section is not intended to be a complete tutorial on Visual Basic program development. It is provided to indicate P-CAD specific information. For a complete description of developing programs using Visual Basic, refer to your *Microsoft Visual Basic Programming System for Windows* documentation.

NOTE: When compiling a Visual Basic application, you may receive the following error: “Too many local, nonstatic variables.” This error is a symptom of a Visual Basic bug that Microsoft describes as occurring “when compiling an application that calls functions or subroutines in a DLL and that passes large User Defined Types to the DLL”. Further details can be found in Microsoft’s Support article, ID# Q179140. There are two ways around this problem:

7) declare the function where the problem is occurring as “static”, or

8) use a level of indirection in your DBX function calls by creating functions that call the DBX functions.

8.4.2 Creating a project

To create a new project, start Visual Basic by selecting the Visual Basic icon in the Visual

Basic icon group and select File/New Project from the main menu. Give the project a name that makes it easy to determine what the program is supposed to do, for example, "NETPRINT" for a netlist print utility.

8.4.3 Including DBX32.BAS

To add DBX32.BAS to your project (which you *must* do to use P-CAD DBX), select File/Add File... from the main menu. Use the File Add directory browse utility to locate and add DBX32.BAS to your project. The file may exist in a single location and be referenced by many projects, or you may make copies of this file and keep a copy in each project directory.

You may make additions to this file, but for convenience and future upgrade ease, it is recommended that you make all code additions to your own forms and Visual Basic source programs.

8.4.4 Adding User Code

User programs can be added in one of two places, either as part of a form in response to a user interface action, or as a separate Visual Basic source file and called by statements you add to a form.

In general, it is most straightforward to add your source within the form which causes it to be executed. You do this by double-clicking on the user interface object which you wish to associate to the source. This invokes a text editor, and allows you to add code which responds to each of the actions possible by that user interface object.

Utility functions referenced by different forms and global variables (a **TNet** item to be used by different forms or different user interface objects, for instance) must be created and referenced in a separate .BAS source file. Global items and variables are created within the *declarations* section of the module, using the GLOBAL keyword. Examples are DBX32.BAS, which provides global variables and function declarations, and DBXUTILS.BAS, which provides several utility functions for use by your programs.

8.4.5 Using Existing Basic Programs

Existing BASIC programs may be called from Visual Basic programs by adding the file to the project (using the File/Add File... menu option) and referencing the program from a project form or source program.

8.4.6 String Handling

Visual Basic uses a scheme for string termination and padding which is different from that used by C programs and Windows Dynamic Link Libraries. DBX function calls return null terminated strings, which is consistent with Windows DLL protocol. Visual Basic fixed length strings are padded with blanks from the end of a string, however, instead of terminating the string with a null character ('\0'). Additionally, Visual Basic ignores null characters when printing these strings to a file and when concatenating strings. To remove trailing spaces from a string returned by P-CAD DBX, use a variant string type and the TRIM command as shown below, or use the **stripnulls** function provided in UTILS.BAS and UTILS.C.

```
status = TGetFirstLayer(tContext, myLayer)
varString = myLayer.layerName
trimmedString = Trim (varString)
```

Removing the padded blanks from a Visual Basic text control is particularly important when using Text items to get user input if the input is to be used as an input string to a Modify function. If your program is getting user input to modify a component Value attribute, for example, the following will modify the component Value attribute to be "NewValue" padded with enough blanks to fill in the rest of the attribute value:

```
myAttr.value = Text1.text
status = TModifyCompAttribute(tContext, pRefDes, myAttr)
```

To strip the unwanted blanks and add the end of string null character from the user input, use the imbedded Visual Basic functions Trim: and Chr() as follows:

```
myAttr.value = Trim(Text1.text) & Chr(0)
status = TModifyCompAttribute(tContext, pRefDes, myAttr)
```

Null terminated strings cause no problems when these strings are used as input to a DBX function or written to Visual Basic controls like list boxes and text boxes.

String comparisons are also handled somewhat differently in Visual Basic. The default is to compare all strings as case sensitive (not too different from other languages) and to allow the user to set all compares to be case insensitive by setting a global program option instead of providing different functions to do different types of compares. This is particularly important when comparing user input to DBX data and looking for the same string (padstyle names, for example) where string case can be used for readability but has no meaning to P-CAD applications. In this case, "Simple" is the same style as "simple", but be considered different strings by a Visual Basic string comparison. To set all compares to be case insensitive, use the **Options Compare Text** command in the *Declarations* section of your program.

8.4.7 Executing the Program

Visual Basic programs may be executed interpretively within the development environment. This is the recommended method to run your DBX program during the development and debugging phase.

Once your program is debugged and ready to be used repeatedly during production design sessions, you can create an executable file (.exe) which may be run similar to other Windows applications.

See Section 9, Running a DBX User Program with P-CAD, for details.

8.5 Using Visual C++ as a Development Environment for DBX

This section assumes a working knowledge of Visual C++ and development of Windows applications. What is presented here is intended to provide an overview of P-CAD DBX interaction with Visual C++, and any specifics needed to integrate a DBX application into a Visual C++ application. To build a Visual C++ application, consult your *Microsoft Visual C++ Development System for Window* documentation.

8.5.1 Overview

Microsoft Visual C++ provides an application environment with an integrated set of Windows applications development tools. The tools include a Visual Workbench, the

main editing and debugging tool, the App Studio, for creating and editing an application's resources, and the Class and AppWizards for development of new classes, member functions, and generation of source for a Microsoft Foundation Class Library application.

Creating an application (referred to as a "project") with Visual C++ creates a set of source, header, and project template files. Dialogs and user interface objects are added to the project using App Studio. C++ classes, methods, and source files are added using AppWizard and ClassWizard. You use the Visual Workbench to add P-CAD DBX function calls and data processing code to your application source files. You also use Visual Workbench to include supplied P-CAD DBX32.LIB and DBX32.DLL library files as part of the project definition and build process. Specifics of how to accomplish DBX related activities are detailed below.

8.5.2 Installing Visual C++

When you install Visual C++, either do a full install, or be sure to make selections that will enable you to build Windows *executables* using a *large* memory model.

8.5.3 Creating a project

To create a new project, start Visual C++ by selecting the Visual C++ icon in the Visual C++ icon group. Use the Project AppWizard command to create a project. Give the project a name that makes it easy to determine what the program is supposed to do, for example, "NETPRINT" for a netlist print utility. Visual C++ will create a new sub-directory with the name of your project under MSVC\BIN. AppWizard also creates a suite of source templates for the project and copies them to the new directory. You add your class definitions, code, and user interface objects to these templates.

8.5.4 Setting the Project Options

There are 5 compiler and linker parameters which must be set prior to developing a DBX application. These settings effect how the executable is built, and how references to the DBX library are made. Set these parameters after creating the project using the Options Project... command.

1) From the Project Options dialog, select either "Release" or "Debug" Build Mode. Selecting "Release" mode will significantly reduce the resulting executable size, but will prevent you from using the debugging tools during development. Select this option when you are done developing the project and want to save disk space on the final product.
2) From the Project Options dialog, select the Compiler... button.
3) Select the Memory Model category. Set the Model to "Large" with the Segment Setup as "SS == DS *".
4) Select the Windows Prolog/Epilog category. Verify that the "Protected Mode Applications Function" Prolog/Epilog radio button is selected. Neither of the Protected Mode Options is required.
5) Select OK to return to the Project Options dialog.
6) From the Project Options dialog, select the Linker... button.
7) Select the Input category.
8) Enable the "Ignore Default Libraries" option. Enable the "Prevent Use of Extended Dictionary" option.

9) Add "P-CADDBX" to the list of libraries, using a comma to delimit the new entry.

10) If you correctly set the memory model size, you should also find the library "llibcew" listed. If "mlibcew" is listed, you are still using the medium memory model libraries. Check to be sure that step 2 was completed properly.

11) Select OK to return to the Project Options dialog. Select OK

8.5.5 Including DBX32.H

DBX32.H must be included in any source files referencing P-CAD DBX functions, data types, or constants. Typically, these files will be the C or C++ source programs where you do your DBX data extraction and processing and will be source modules supporting a derived class of CView.

8.5.6 Linking to DBX32.DLL and DBX32.LIB

Copy these files from your P-CAD installation directory into your Visual C++ project directory. After setting the project linking options as described under Setting the Project Options, Visual C++ will automatically reference these library files as needed.

8.5.7 Adding User Code

User code is added to the project by editing source files using the Edit command. You can either create new source, or copy and paste from other applications or DBX sample programs.

8.5.8 Executing the Program

Visual C++ programs may be executed within the development environment. This is the recommended method to run your DBX program during the development and debugging phase.

Once your program is debugged and ready to be used repeatedly during production design sessions, you can create an executable file (.exe) which may be run similar to other Windows applications.

See Section 9, Running a DBX User Program with P-CAD, for details.

8.6 Examples

The following are Visual Basic code samples which provide examples of some commonly used functions. Complete and more complex examples are included on your P-CAD DBX installation directory and installation CD. The online examples are described in Section 10.0.

8.6.1 Opening and closing a design

This example opens and closes an active PCB design session. Note that it uses the global variables tContext and tStatus, and program constants DBX_LANGUAGE and DBX_VERSION. These values and variables are provided in DBX32.BAS.

```
tStatus = TOpenDesign(DBX_LANGUAGE,DBX_VERSION,
```

```
                                                "pcb",tContext)
If (tStatus = DBX_OK) Then
     statusBox.Text = "Design Opened"
     tStatus = TCloseDesign(tContext, "")
End If
```

8.6.2 Retrieving layer and component data

Print the reference designators for all components in the active PCB design.

```
Dim myComponent As TComponent

Open "FileName" For Output As 10

tStatus = TOpenDesign(DBX_LANGUAGE,DBX_VERSION,"pcb",tCont ext)

If (tStatus <> DBX_OK) Then
     statusBox.Text = "Error Opening Design"
     Exit Sub
End If

tStatus = TGetFirstComponent(tContext,myComponent)

Do While (tStatus = DBX_OK)
     Print #10, "Component RefDes = " &
                                    myComponent.refDes
     tStatus = TGetNextComponent(tContext,
                                    myComponent)
Loop

tStatus = TCloseDesign(tContext, "")
Close
```

8.6.3 Retrieving specific item data

Get and display the net ID and net length for a net name which the user has entered into textbox "text1".

```
Dim myNet As TNet
Dim netName As String

tStatus = TOpenDesign(DBX_LANGUAGE,DBX_VERSION,"pcb",tCont ext)

netName = Text1.Text
tStatus = TGetNetByName(tContext, netName, myNet)

If (tStatus = DBX_NO_MORE_NETS) Then
    statusBox.Text = "Net Not Found"
ElseIf (tStatus <> DBX_OK) Then
    statusBox.Text = "Error"
Else
    outputBox.Text = "Net ID = " & myNet.netId & ",
                      Net length = " & myNet.length
End If

tStatus = TCloseDesign(tContext, "")
```

8.7 Common Run-time Errors

Listed below are some of the more common problems encountered when executing or building a DBX program.

(C or C++) "Error Executing Application. The file or one of its components could not be found."

1) DBX32.DLL is not in the current directory on in any directory that is in the PATH specified in AUTOEXEC.BAT. Add the directory to the current path, or copy DBX32.DLL to a directory that is in the path.

(Visual Basic) "File Error. Cannot Find DBX32.DLL"

1) DBX32.DLL is not in the current directory on in any directory that is in the PATH specified in AUTOEXEC.BAT. Add the directory to the current path, or copy DBX32.DLL to a directory that is in the path.

(Visual Basic) "Type Not Defined" for a P-CAD DBX Item Type

1) DBX32.BAS is not included in the project. Use File Add to add DBX32.BAS to the project.

DBX_INCOMPATIBLE_VERSION returned from TOpenDesign

1) Your user program has been compiled with a version of DBX32.H or DBX32.BAS which is too old to be used with the current DBX32.DLL
2) There is a mismatch between DBX32.DLL and the version of P-CAD PCB you are running. Versions of DBX32.DLL and PCB.EXE must always be the same.

DBX_SERVER_BUSY returned from TOpenDesign

1) An P-CAD dialog is open.
2) P-CAD is processing a command or waiting for user input for a command. Place Component, for example.
3) The router is executing.

(C or C++) Function call returns DBX_OK but item property data is unreadable

1) Check the compiler/linker byte alignment settings. While a byte alignment setting of 4 will work in most environments, and is more efficient, your environment may require an alignment of 2 or 1.

9 Running a DBX User Program with P-CAD

P-CAD must be currently running and have a design open in order for an P-CAD DBX program extracting or modifying design data to run.

Also, because the DBX library interacts directly with the P-CAD database, it is important that you not execute DBX programs while you are modifying your design. Depending on the type of operation, running DBX at the same time as database manipulation commands could possibly cause database corruption in worst cases. This is possible because Windows is a message based system which allows multiple processes to interact with a single program simultaneously, and to queue up commands before the current command is complete. An example of this is the P-CAD highlight feature.

To protect users from this type of problem, several safeguards are used by P-CAD DBX. First, DBX checks for some of the more time intensive operations and will not establish a conversation if that operation is in progress. Autorouting is an example. Second, if an P-CAD dialog is open or an P-CAD tool is active, during line placement, for example, the DBX connection will be refused by P-CAD PCB or Schematic. Third, when a DBX conversation is established, the P-CAD application displays a dialog box alerting you to the fact, and disables the user interface from beginning any new commands. These are not foolproof safeguards, however, so with some effort these safeguards can be circumvented. It is not recommended that you do so.

To execute an P-CAD DBX program, do the following:

1) Start the P-CAD PCB, Schematic, Library Manager, or Library Executive.
2) Open the design from which you wish to extract or modify data.
3) Either minimize the P-CAD workspace to a smaller screen, or reduce it to an icon. If you do not need to see the workspace while DBX is running, it is faster and most convenient to reduce it to an icon.
4) Start your P-CAD DBX program. This will suspend the P-CAD design session and display the "DBX Program in Progress" dialog. You can move this dialog to the side where it will be out of your way.
5) Run your DBX program. On successful exit and TCloseDesign execution, P-CAD PCB, Schematic, Library Manager, or Library Executive will remove the "DBX Program In Progress" dialog, and again be accessible.

If your DBX program should terminate abnormally, due to a crash or other programming error, or your program does not call **TCloseDesign**, PCB will remain in a suspended state. You can regain control over PCB by selecting "Cancel" on the DBX In Progress dialog and resume your design session without problems.

10 Online Sample Programs

Your P-CAD DBX installation includes online sample programs. The sample files include all the necessary code modules and project makefiles to reproduce and execute the samples. One sample program is a Visual C++ example, four are Visual Basic examples. They range from simple to moderately complex. They also present a variety of different interfaces and output styles including list box input and output to both the screen and a report file. There are also additional sample programs on the Installation CD. These programs are typically more complex and have more source files.

Browsing through the code modules can provide an insight into how various DBX applications might be developed. The samples can also provide excellent "templates" for development of your own DBX programs. We recommend that you choose a sample having the basic structure you need, copy the code modules to your own project, and modify the sample program to suit your particular need. As an example, to create a DBX program which cycles through the nets in the design and retrieves all tracks (lines) to verify the widths of the tracks, you would use DBXSAMP4 as a template. This sample cycles through all nets and retrieves all of the items in each net. It is a simple change to modify the program to check for lines instead of vias. DBXSAMP4 can be found on the Installation CD in the DBXSRC\DBXSAMP4 directory.

ANULRCHK - Annular Ring Check (Visual Basic)

This program verifies that each pad meets a minimum annular ring distance, where the minimum distance between pad and hole is specified by the user. It is the most complex of the examples, using text boxes for input, writing output to multiple list-boxes, and allowing the user to choose between display of different units. It also does the most processing of PCB data to determine the validity of each pad. As with most of the samples, though, the internal structure is straightforward with a loop getting each component, and within that loop, a loop getting each pad on the component.

This is a good example demonstrating various input and output techniques, conversion between database and user units, and using the STRIP_NULLS utility for output.

TSTPTLOC - Testpoint Locator (Visual Basic)

This program examines each net in the active design to determine if a via exists which is defined on either top or bottom layer. The program generates a report listing nets having vias defined on top or bottom layers, and those having no vias meeting the requirement. This sample, and PDSTYLOC, are of medium complexity, using a text-box for input of the filename, generating a report, and displaying the current design name and status messages as processing proceeds. The internal structure of the program is basically a loop getting each net, and within that loop, a loop retrieving each item in the net. All items that are not vias are ignored.

Note that this project (TSTPTLOC.MAK) includes a file TSTPTLOC.BAS. TSTPTLOC.BAS includes a "user defined" type called NETINFO, which is used to maintain net information including the net name, and any vias meeting the top or bottom requirement. This is a good example of creating a user defined type when a particular structure is needed in addition to those provided with P-CAD DBX.

PDSTYLOC - Pad Style Locator (Visual Basic)

This program displays a list of all components in the active design having a particular pad style. The padstyle is input by the user. Output is written to a list box, which is scrollable

if the number of components exceed the listbox size. Visual Basic automatically handles alphabetizing the items in the list-box, and adding scroll bars if needed. The program structure is nearly identical to TSTPTLOC, and was in fact created by copying TSTPTLOC and changing the calls from retrieving nets and net items, to retrieving components and component pads.

This is a good sample to follow for making printed reports or for programs requiring nested loops to retrieve lower level data.

DBXSAMP4 - Netlist Report (C++)

This program generates a printed report of all nets in the active design, their lengths, and their total length. It is functionally equivalent to NTLSTRPT, described below, but was developed using Visual C++. It is programmatically the simplest of the examples, where building the project using Visual C++ or another Windows environment presents the most challenge. For this reason, the interface has been kept to an absolute minimum, with only a start and exit button provided. The program cycles through each net in the design, getting and printing the net name and length, writing the output to the file NETLIST.RPT in the current directory. This project is a good project to copy as a starting point for your Visual C++ DBX applications. DBXSAMP4 can be found on the Installation CD in the DBXSRC\DBXSAMP4 directory.

NTLSTRPT - Netlist Report (Visual Basic)

This sample generates the same report as DBXSAMP4, only it is written in Visual Basic and will give net data for either Schematic or PCB designs. The program generates a printed report of all nets in the active design, number of nodes, their lengths (for PCB), and their total length (for PCB). It is the simplest of all the samples, both programmatically and from a user interface perspective. The program makes one cycle through the active design nets, and prints net names and lengths to the user specified file. A default filename is provided, which the user can overwrite by changing an input text box. This sample provides a good starting point for simple programs needing to generate a printed report.

11 Upgrading DBX Programs

In version 14.0, there are several new objects and extensions to old objects. Function calls using these structures may need modification. For a summary of these objects, see the *Important Information about Version 14.0* in the P-CADDBX.doc introduction. See appendix B for detailed information on these DBX objects.

V13.0 DBX source is compatible with the V14.0 runtime library, DBX32.DLL. 16 bit TangoPRO DBX and P-CAD DBX source programs are also compatible with the 32 bit DBX32.DLL function calls. To upgrade your DBX applications to use the new DBX32.DLL:

1) The 16 bit TANGODBX.DLL and P-CADDBX.DLL and the V13.0 DBX32.DLL have been superceded by the V14.0 DBX32.DLL. These files must be replaced with DBX32.DLL shipped on the V14.0 installation CD and copied into your WINDOWS\SYSTEM directory, or another directory in your DOS PATH. If you do not replace these files with the V14.0 DBX32.DLL, you will experience unpredictable results when retrieving design data.
2) If you are accessing grid information using the TDesign.currentGrid field, update the source to use the new TGetFirstGrid( ), TGetNextGrid( ), and TGrid item, removing any references to the TDesign.currentGrid field. Changes to this object are summarized in *Important Information about Version 14.0.*
3) Recompile and relink your source programs using the supplied DBX32.H (for C or C++ programs) or DBX32.BAS (for Visual Basic programs).

Appendix A: DBX Data Constants

These data structures are excerpted from DBX32.H.

```
The following are interface constants used to define error messages,array
sizes, object types, and various design data. These values MUST NOT BE MODIFIED
by the user or by user programs or results may be undefined, including design
data corruption.

//     String Lengths
//
#define DBX_MAX_NAME_LEN            100     //  Name variables
#define DBX_MAX_TYPE_LEN            100     //  Type attributes
#define DBX_MAX_VALUE_LEN           100     //  Value attributes
#define DBX_MAX_TEXTITEM_LEN        256     //  Text Items
#define DBX_MAX_ATTRIBUTE_LEN       256     //  Attribute Values
#define DBX_MAX_FILENAME_LEN        256     //  Filenames
#define DBX_MAX_GRIDSPACING_LEN     256     //  Grid spacing
#define DBX_MAX_FORMULA_LEN         512     //  Formula, comment in TAttribute

#define DBX_NUM_LAYER_VALS_TO_SET     6     //  Number of fields to set in
                                               TLayer during call to
                                               ModifyLayer

//     Fatal Errors
//
#define DBX_CONNECTION_ERROR        32001
#define DBX_DISCONNECT_ERROR        32002
#define DBX_BAD_SERVER_DATA         32003
#define DBX_SERVER_ERROR            32004
#define DBX_FATAL_ERROR             32005
#define DBX_VERSION_INCOMPATIBLE    32006
#define DBX_CLIENT_ERROR            32007
#define DBX_SECURITY_ERROR          32008

#define DBX_FLIP_ERROR              32009  // 32009-32016 new for V12.0
#define DBX_ROTATE_ERROR            32010
#define DBX_MOVE_ERROR              32011
#define DBX_HIGHLIGHT_ERROR         32012
#define DBX_DELETE_ERROR            32013
#define DBX_MODIFY_ERROR            32014
#define DBX_CREATE_ERROR            32015
#define DBX_DATABASE_ERROR          32016


//
//     Status Return Values - Normal return values indicating
//  success or failure due to normal run-time conditions
//
#define DBX_OK                      0

#define DBX_INVALID_CONV_HANDLE     32101
#define DBX_SERVER_TERMINATED       32102
#define DBX_LOW_MEMORY              32103
#define DBX_ALREADY_CONNECTED       32104
```

```
#define DBX_NO_CONNECTION           32105
#define DBX_SERVER_BUSY             32106

#define DBX_ILLEGAL_OP              32121
#define DBX_BAD_INPUT               32122
#define DBX_ARRAY_TOO_SMALL         32123
#define DBX_INVALID_ITEM_ID         32124

#define DBX_ITEM_NOT_FOUND          32141
#define DBX_ITEM_NOT_SUPPORTED      32142
#define DBX_ILLEGAL_ITEM            32143
#define DBX_SHAPE_NOT_DEFINED       32144
#define DBX_GETFIRST_NOT_CALLED     32145
#define DBX_GETCOMP_NOT_CALLED      32146
#define DBX_GETNET_NOT_CALLED       32147


#define DBX_NO_MORE_LAYERS          32161
#define DBX_NO_MORE_NETS            32162
#define DBX_NO_MORE_ITEMS           32163
#define DBX_NO_MORE_NETCLASSES      32164
#define DBX_NO_MORE_CLASSTOCLASSES  32165  // New for v14.0
#define DBX_NO_MORE_ROOMS           32166  // New for v14.0
#define DBX_NO_MORE_ROOMPOINTS      32167  // New for v14.0
#define DBX_NO_MORE_COMPONENTS      32168  // New for v14.0

#define DBX_INVALID_ANGLE           32170  // New for V12.0
#define DBX_INVALID_HIGHLIGHT_COLOR 32171
#define DBX_INVALID_ITEM_TYPE       32172
#define DBX_INVALID_JUSTIFICATION   32173
#define DBX_INVALID_LAYER           32174
#define DBX_INVALID_LOCATION        32175
#define DBX_INVALID_NET_ID          32176
#define DBX_INVALID_NETNAME         32177
#define DBX_INVALID_PADSTYLE        32178
#define DBX_INVALID_PIN_NUMBER      32279
#define DBX_INVALID_RADIUS          32180
#define DBX_INVALID_REFDES          32181
#define DBX_INVALID_ROTATION_ANGLE  32182
#define DBX_INVALID_TEXTSTYLE       32183
#define DBX_INVALID_WIDTH           32184
#define DBX_INVALID_LAYERNAME       32185
#define DBX_INVALID_COMPNAME        32186
#define DBX_NOT_SAME_NUM_PARTS      32187  // New for 13.0
#define DBX_NOT_SAME_NUM_PINS       32188  // New for 13.0
#define DBX_NOT_SAME_COMP_TYPE      32189  // New for 13.0
#define DBX_NOT_SAME_HETERO         32190  // New for 13.0
#define DBX_INVALID_LINESTYLE       32191
#define DBX_INVALID_PINLENGTH       32192
#define DBX_INVALID_PINSTYLE        32193
#define DBX_INVALID_ALTTYPE         32194
#define DBX_INVALID_NETCLASSNAME    32195  // New for v14.0
#define DBX_INVALID_CLASSTOCLASSID  32196  // New for v14.0
#define DBX_INVALID_NETCLASS_ID     32197  // New for v14.0
#define DBX_INVALID_LAYER_ID        32198  // New for v14.0
#define DBX_INVALID_ROOMNAME        32199  // New for v14.0
#define DBX_INVALID_ATTRIBUTE_VALUE 32200
```

```
#define DBX_INVALID_ATTRIBUTE_NAME  32201

#define DBX_DUPLICATE_ATTRIBUTE     32205
#define DBX_DUPLICATE_NETNAME       32206
#define DBX_DUPLICATE_NETCLASSNAME  32207  // New for v14.0
#define DBX_DUPLICATE_CLASSTOCLASSNAME 32208  // New for v14.0
#define DBX_DUPLICATE_REFDES        32209

#define DBX_COMPONENT_NOT_FOUND     32215
#define DBX_LAYER_NOT_FOUND         32216
#define DBX_NET_NOT_FOUND           32217
#define DBX_NET_NOT_EMPTY           32218
#define DBX_NET_EXISTS              32219
#define DBX_NET_CONFLICT            32220
#define DBX_NETCLASS_NOT_FOUND      32221  // New for v14.0
#define DBX_NETCLASS_NOT_EMPTY      32222
#define DBX_NETCLASS_EXISTS         32223
#define DBX_CLASSTOCLASS_NOT_FOUND  32224
#define DBX_ROOM_NOT_FOUND          32225

#define DBX_COMP_CACHE_CONFLICT     32230
#define DBX_LIBRARY_NOT_OPEN        32231
#define DBX_ITEM_OUTSIDE_WORKSPACE  32232
#define DBX_NO_ATTACHED_PATTERN     32233
#define DBX_TOO_FEW_PINS            32234
#define DBX_PINDES_NOT_FOUND        32235
#define DBX_NAME_TOO_LONG           32236

#define DBX_PRINT_JOB_NOT_FOUND     32237
#define DBX_NO_PRINT_JOBS_SELECTED  32238
#define DBX_FILE_OPEN_FAILURE       32239


//
// For Library Manager
//
#define DBX_INVALID_LIBRARY_NAME    32250
#define DBX_LIBRARY_CANT_OPEN       32251
#define DBX_COMP_ALREADY_OPEN       32252
#define DBX_OPENCOMP_NOT_CALLED     32253
#define DBX_SAVECOMP_FAILED         32254
#define DBX_LIBRARY_READ_ONLY       32255
#define DBX_SAVESYMBOL_FAILED       32256
#define DBX_SAVEPATTERN_FAILED      32257


//
// Internal Parameter Error
//
#define DBX_INTERNAL_PARAMETER_ERROR 32258


//
//     Interface Description
//
#define DBX_VERSION                 15000   // 14.000 (beta version!)
#define DBX_LANGUAGE                0
```

```
//
//  Item Types: Returned in TItem.itemType
//
#define DBX_PAD                     3       //  Component Pad
#define DBX_VIA                     4       //  Via
#define DBX_LINE                    5       //  Line
                                            //  Routed Connection
                                            //  Mitered Line
#define DBX_FROM_TO                 6       //  Unrouted Net Conn.
#define DBX_ARC                     7       //  Arc
                                            //  Ortho 90/90 Arc
                                            //  90 Arc Miter
#define DBX_POLYGON                 8       //  Filled Polygon
#define DBX_COMPONENT               9       //  Component
#define DBX_CONNECTION              10      //
#define DBX_COPPERPOUR              11      //  Copper Pour

#define DBX_TEXT                    13      //  Text
#define DBX_ATTR                    14      //  Attribute
#define DBX_FIELD                   15      //  Field
#define DBX_REFPOINT                16      //  Ref Point
#define DBX_GLUEPOINT               17      //  Glue Dot Point
#define DBX_PICKPOINT               18      //  Pick and Place Pt.
#define DBX_PATTERN                 19
#define DBX_LINE_KEEPOUT            20      //  Line Keepout
#define DBX_POLY_KEEPOUT            21      //  Polygon Keepout

#define DBX_PAD_STYLE               24      //  Pad Style
#define DBX_VIA_STYLE               25      //  Via Style
#define DBX_TEXT_STYLE              26      //  Text Style
#define DBX_KEEPOUT                 27      //
#define DBX_COMP_PIN                28      //  Component pin
#define DBX_POLY_CUTOUT             29      //  Cutout Polygon

#define DBX_LAYER                   33      //  Layer, sheet

#define DBX_SYMBOL                  35      //  Symbol
#define DBX_NET                     36      //  Net

#define DBX_PIN                     39      //  Pin
#define DBX_INFOPOINT               40      //  Info Point

#define DBX_WIRE                    44      //  Sch field item
#define DBX_BUS                     45      //  Sch field item

#define DBX_PORT                    50      //  Sch net port

#define DBX_NETCLASS                54      //  NetClass

#define DBX_TABLE                   58
#define DBX_METAFILE                59

#define DBX_DIAGRAM                 66
#define DBX_DETAIL                  67
#define DBX_CLASSTOCLASS            68      //  ClassToClass
```

```
#define DBX_ROOM                    69

#define DBX_DESIGNVIEW              74

#define DBX_COORDINATE              901     //  Coordinate Point
                                            //  for Polygon and
                                            //  Pour Outlines
#define DBX_PADVIA_SHAPE            902     //  Shape Description
                                            //  for Pads and Vias
#define DBX_DESIGN_INFO             903     //  Design Information
                                            //
#define DBX_POINT                   904     //  Point for Room boundaries
                                            //
#define DBX_PRINT_JOB               905     //  Print job


//
//  Text Justifications: Returned in TText.justPoint
//
#define DBX_JUSTIFY_TOP          9   //  UL-----T-----UR
#define DBX_JUSTIFY_BOTTOM       1   //  |      |      |
#define DBX_JUSTIFY_LEFT         4   //  L------C------R
#define DBX_JUSTIFY_RIGHT        6   //  |      |      |
#define DBX_JUSTIFY_CENTER       5   //  LL-----B-----LR
#define DBX_JUSTIFY_LOWER_LEFT   0   //
#define DBX_JUSTIFY_LOWER_RIGHT  2   //  TangoFont defines LL as
#define DBX_JUSTIFY_UPPER_LEFT   8   //  the Ref Point.
#define DBX_JUSTIFY_UPPER_RIGHT  10  //


//
// Font Types:  Returned in TTextStyle.fontType
//
#define DBX_FONT_WINSTROKE       1
#define DBX_FONT_OUTLINE         2
#define DBX_FONT_STROKE          3
#define DBX_FONT_TRUE_TYPE       4


//
//  Layer Types: Returned in TLayer.layerType
//
#define DBX_LAYERTYPE_UNDEFINED -1   //  Used by GetPadShapeByLayer
                                     //  if the layer doesn't exist
#define DBX_LAYERTYPE_SIGNAL     0   //  Layer Type: Signal
#define DBX_LAYERTYPE_PLANE      1   //  Layer Type: Plane
#define DBX_LAYERTYPE_NON_SIGNAL 2   //  Layer Type: Non Signal


//
//  Pre-defined Layers
//
#define DBX_LAYER_MULTI          0   // "Multi-Layer', used for comps, pads,
                                             etc.
#define DBX_LAYER_TOP_SIGNAL     1
#define DBX_LAYER_BOTTOM_SIGNAL  2
#define DBX_LAYER_BOARD          3
```

```
#define DBX_LAYER_TOP_MASK       4
#define DBX_LAYER_BOTTOM_MASK    5
#define DBX_LAYER_TOP_SILK       6
#define DBX_LAYER_BOTTOM_SILK    7
#define DBX_LAYER_TOP_PASTE      8
#define DBX_LAYER_BOTTOM_PASTE   9
#define DBX_LAYER_TOP_ASSY      10
#define DBX_LAYER_BOTTOM_ASSY   11


//
//  Layer Bias: Returned in TLayer.layerBias
//
#define DBX_LAYERBIAS_AUTO       0   //  AutoRouter Bias: Automatic
#define DBX_LAYERBIAS_HORIZ      1   //  AutoRouter Bias: Horizontal
#define DBX_LAYERBIAS_VERT       2   //  AutoRouter Bias: Vertical


//
//  Pour Thermals: Returned in TPour.thermalType
//
#define DBX_POUR_THERMAL_NONE    0   //  Pour Thermals: None
#define DBX_POUR_THERMAL_45      1   //  Pour Thermals: 45 degree
#define DBX_POUR_THERMAL_90      2   //  Pour Thermals: 90 degree


//
//  Pour Patterns: Returned in TPour.pourType
//
#define DBX_POUR_SOLID           0   //  Pour Pattern: Solid
#define DBX_POUR_HORIZ           1   //  Pour Pattern: Horizontal
#define DBX_POUR_VERT            2   //  Pour Pattern: Vertical
#define DBX_POUR_HATCH45         3   //  Pour Pattern: 45 deg Cross
#define DBX_POUR_HATCH90         4   //  Pour Pattern: 90 deg Cross


//
//    Pad and Via Shapes: Returned in TPadViaShape.shape
//
#define DBX_SHAPE_ELLIPSE        0   //  Elliptical, Circular
#define DBX_SHAPE_OVAL           1   //  Oval
#define DBX_SHAPE_RECT           2   //  "Square" Rectangle
#define DBX_SHAPE_RND_RECT       3   //  "Rounded" Rectangle
#define DBX_SHAPE_THRM_2         4   //  "Two Spoke" Thermal
#define DBX_SHAPE_THRM_2_90      5   //  "Two Spoke With 90
                                     //   Rotation" Thermal
#define DBX_SHAPE_THRM_4         6   //  "Four Spoke" Thermal
#define DBX_SHAPE_THRM_4_45      7   //  "Four Spoke With 45
                                     //  Rotation" Thermal
#define DBX_SHAPE_DIRECT         8   //  Direct Plane Connect
#define DBX_SHAPE_TARGET         9   //  Registration Target
                                     //  (Not Used for Vias)
#define DBX_SHAPE_MT_HOLE        10  //  Mounting Hole
                                     //  (Not Used for Vias)
#define DBX_SHAPE_POLYGON        11  //  Polygonal
#define DBX_SHAPE_NOCONNECT      12  //  Disconnect (plane layers)
```

```
//
//    Pin Electrical Types: Returned in TPad.pinType
//
#define DBX_PIN_ETYPE_UNKNOWN    0   //  Elec. Type: Unknown
#define DBX_PIN_ETYPE_PASSIVE    1   //  Elec. Type: Passive
#define DBX_PIN_ETYPE_INPUT      2   //  Elec. Type: Input
#define DBX_PIN_ETYPE_OUTPUT     3   //  Elec. Type: Output
#define DBX_PIN_ETYPE_BIDIRECT   4   //  Elec. Type: Bidirectional
#define DBX_PIN_ETYPE_OPEN_H     5   //  Elec. Type: Open-H
#define DBX_PIN_ETYPE_OPEN_L     6   //  Elec. Type: Open-L
#define DBX_PIN_ETYPE_PASS_H     7   //  Elec. Type: Passive-H
#define DBX_PIN_ETYPE_PASS_L     8   //  Elec. Type: Passive-L
#define DBX_PIN_ETYPE_3STATE     9   //  Elec. Type: 3-State
#define DBX_PIN_ETYPE_POWER     10   //  Elec. Type: Power


//
//    Port Types: Returned in TPort.portType
//
#define DBX_PORT_NOANGLE_SGL_VERT    1
#define DBX_PORT_NOANGLE_SGL_HORZ    2
#define DBX_PORT_LEFTANGLE_SGL_VERT  3
#define DBX_PORT_LEFTANGLE_SGL_HORZ  4
#define DBX_PORT_RIGHTANGLE_SGL_VERT 5
#define DBX_PORT_RIGHTANGLE_SGL_HORZ 6
#define DBX_PORT_BOTHANGLE_SGL_VERT  7
#define DBX_PORT_BOTHANGLE_SGL_HORZ  8
#define DBX_PORT_NOANGLE_DBL_VERT    9
#define DBX_PORT_NOANGLE_DBL_HORZ    10
#define DBX_PORT_LEFTANGLE_DBL_VERT  11
#define DBX_PORT_LEFTANGLE_DBL_HORZ  12
#define DBX_PORT_RIGHTANGLE_DBL_VERT 13
#define DBX_PORT_RIGHTANGLE_DBL_HORZ 14
#define DBX_PORT_BOTHANGLE_DBL_VERT  15
#define DBX_PORT_BOTHANGLE_DBL_HORZ  16
#define DBX_PORT_VERTLINE_SGL_VERT   17
#define DBX_PORT_VERTLINE_SGL_HORZ   18
#define DBX_PORT_NOOUTLINE_SGL_VERT  19
#define DBX_PORT_NOOUTLINE_SGL_HORZ  20
#define DBX_PORT_VERTLINE_DBL_VERT   21
#define DBX_PORT_VERTLINE_DBL_HORZ   22
#define DBX_PORT_NOOUTLINE_DBL_VERT  23
#define DBX_PORT_NOOUTLINE_DBL_HORZ  24


//
//  Pin Length Types: Returned in TPort.pinLength
//
#define DBX_PORT_PIN_LONG            300
#define DBX_PORT_PIN_SHORT           100
#define DBX_PORT_PIN_CUSTOM          0


//
//  Line Style & Widths (in mils)
//
```

```
#define DBX_SOLID_LINE      0
#define DBX_DASHED_LINE     1
#define DBX_DOTTED_LINE     2
#define DBX_THIN_LINE       10
#define DBX_THICK_LINE      30


//
//  Pin Display Styles:  Returned in inside and outside styles
//
#define DBX_PIN_DISPLAY_O_NONE       0
#define DBX_PIN_DISPLAY_O_FLOW_IN    1
#define DBX_PIN_DISPLAY_O_FLOW_OUT   2
#define DBX_PIN_DISPLAY_O_FLOW_BI    3
#define DBX_PIN_DISPLAY_O_ANALOG     4
#define DBX_PIN_DISPLAY_O_DIGITAL    5
#define DBX_PIN_DISPLAY_O_NONLOGIC   6

#define DBX_PIN_DISPLAY_OE_NONE      0
#define DBX_PIN_DISPLAY_OE_DOT       1
#define DBX_PIN_DISPLAY_OE_POL_IN    2
#define DBX_PIN_DISPLAY_OE_POL_OUT   3

#define DBX_PIN_DISPLAY_IE_NONE      0
#define DBX_PIN_DISPLAY_IE_CLOCK     1

#define DBX_PIN_DISPLAY_I_NONE       0
#define DBX_PIN_DISPLAY_I_OPEN       1
#define DBX_PIN_DISPLAY_I_OPEN_H     2
#define DBX_PIN_DISPLAY_I_OPEN_L     3
#define DBX_PIN_DISPLAY_I_PASS_UP    4
#define DBX_PIN_DISPLAY_I_PASS_DOWN  5
#define DBX_PIN_DISPLAY_I_3_STATE    6
#define DBX_PIN_DISPLAY_I_AMPLIFIER  7
#define DBX_PIN_DISPLAY_I_GENERATOR  8
#define DBX_PIN_DISPLAY_I_HYSTERESIS 9
#define DBX_PIN_DISPLAY_I_POSTPONED  10
#define DBX_PIN_DISPLAY_I_SHIFT      11


//
//  Symbol Alternate types
//
#define DBX_ALTTYPE_NORMAL       0
#define DBX_ALTTYPE_IEE          1
#define DBX_ALTTYPE_DEMORGAN     2


//
//  Drill Symbol Types:  Returned in TPadViaStyle.drillSymbol
//
#define DBX_DRILLSYM_UNDEFINED      -1
#define DBX_DRILLSYM_CROSS           0
#define DBX_DRILLSYM_X               1
#define DBX_DRILLSYM_Y               2
#define DBX_DRILLSYM_T               3
#define DBX_DRILLSYM_HOUR            4
```

```
#define DBX_DRILLSYM_SIDE_HOUR       5
#define DBX_DRILLSYM_BOX_LINE        6
#define DBX_DRILLSYM_DIAMOND_LINE    7
#define DBX_DRILLSYM_BOX_V           8
#define DBX_DRILLSYM_DIAMOND_V       9
#define DBX_DRILLSYM_BOX_X           10
#define DBX_DRILLSYM_DIAMOND_CROSS   11
#define DBX_DRILLSYM_BOX_CROSS       12
#define DBX_DRILLSYM_DIAMOND_X       13
#define DBX_DRILLSYM_BOX_Y           14
#define DBX_DRILLSYM_DIAMOND_Y       15
#define DBX_DRILLSYM_BOX_T           16
#define DBX_DRILLSYM_DIAMOND_T       17
#define DBX_DRILLSYM_CIRCLE_LINE     18
#define DBX_DRILLSYM_CIRCLE_V        19
#define DBX_DRILLSYM_CIRCLE_CROSS    20
#define DBX_DRILLSYM_CIRCLE_X        21
#define DBX_DRILLSYM_CIRCLE_Y        22
#define DBX_DRILLSYM_CIRCLE_T        23
#define DBX_DRILLSYM_UPPER_A         24
#define DBX_DRILLSYM_UPPER_B         25
#define DBX_DRILLSYM_UPPER_C         26
#define DBX_DRILLSYM_UPPER_D         27
#define DBX_DRILLSYM_UPPER_E         28
#define DBX_DRILLSYM_UPPER_F         29
#define DBX_DRILLSYM_UPPER_G         30
#define DBX_DRILLSYM_UPPER_H         31
#define DBX_DRILLSYM_UPPER_I         32
#define DBX_DRILLSYM_UPPER_J         33
#define DBX_DRILLSYM_UPPER_K         34
#define DBX_DRILLSYM_UPPER_L         35
#define DBX_DRILLSYM_UPPER_M         36
#define DBX_DRILLSYM_UPPER_N         37
#define DBX_DRILLSYM_UPPER_O         38
#define DBX_DRILLSYM_UPPER_P         39
#define DBX_DRILLSYM_UPPER_Q         40
#define DBX_DRILLSYM_UPPER_R         41
#define DBX_DRILLSYM_UPPER_S         42
#define DBX_DRILLSYM_UPPER_U         43
#define DBX_DRILLSYM_UPPER_V         44
#define DBX_DRILLSYM_UPPER_W         45
#define DBX_DRILLSYM_UPPER_Z         46
#define DBX_DRILLSYM_LOWER_A         47
#define DBX_DRILLSYM_LOWER_B         48
#define DBX_DRILLSYM_LOWER_C         49
#define DBX_DRILLSYM_LOWER_D         50
#define DBX_DRILLSYM_LOWER_E         51
#define DBX_DRILLSYM_LOWER_F         52
#define DBX_DRILLSYM_LOWER_G         53
#define DBX_DRILLSYM_LOWER_H         54
#define DBX_DRILLSYM_LOWER_I         55
#define DBX_DRILLSYM_LOWER_J         56
#define DBX_DRILLSYM_LOWER_K         57
#define DBX_DRILLSYM_LOWER_L         58
#define DBX_DRILLSYM_LOWER_M         59
#define DBX_DRILLSYM_LOWER_N         60
#define DBX_DRILLSYM_LOWER_O         61
```

```
#define DBX_DRILLSYM_LOWER_P         62
#define DBX_DRILLSYM_LOWER_Q         63
#define DBX_DRILLSYM_LOWER_R         64
#define DBX_DRILLSYM_LOWER_S         65
#define DBX_DRILLSYM_LOWER_T         66
#define DBX_DRILLSYM_LOWER_U         67
#define DBX_DRILLSYM_LOWER_V         68
#define DBX_DRILLSYM_LOWER_W         69
#define DBX_DRILLSYM_LOWER_Y         70
#define DBX_DRILLSYM_LOWER_Z         71

//  DBX Color Index Types for Hilight
#define DBX_COLOR_BLACK              1
#define DBX_COLOR_DARKRED            2
#define DBX_COLOR_DARKGREEN          3
#define DBX_COLOR_DARKYELLOW         4
#define DBX_COLOR_DARKBLUE           5
#define DBX_COLOR_DARKMAGENTA        6
#define DBX_COLOR_DARKCYAN           7
#define DBX_COLOR_LIGHTGRAY          8
#define DBX_COLOR_DARKGRAY           9
#define DBX_COLOR_RED                10
#define DBX_COLOR_GREEN              11
#define DBX_COLOR_YELLOW             12
#define DBX_COLOR_BLUE               13
#define DBX_COLOR_MAGENTA            14
#define DBX_COLOR_CYAN               15
#define DBX_COLOR_WHITE              16
#define DBX_COLOR_POWDERGREEN        17
#define DBX_COLOR_POWDERBLUE         18
#define DBX_COLOR_POWDER             19
#define DBX_COLOR_GRAY               20


//
//  Field Key Types
//
#define DBX_FIELD_DATE               1
#define DBX_FIELD_CURDATE            2
#define DBX_FIELD_TIME               3
#define DBX_FIELD_CURTIME            4
#define DBX_FIELD_AUTHOR             5
#define DBX_FIELD_REV                6
#define DBX_FIELD_FILENAME           7
#define DBX_FIELD_TITLE              8
#define DBX_FIELD_SHEETNUM           10
#define DBX_FIELD_NUMSHEETS          11
#define DBX_FIELD_DRAWING_NUMBER     12
#define DBX_FIELD_NOTE               13
#define DBX_FIELD_REVISION_NOTE      14


//
//    Table types
//
#define DBX_TABLE_NET_INDEX            0
#define DBX_TABLE_NOTES                1
```

```
#define DBX_TABLE_POWER                2
#define DBX_TABLE_SPARE_GATE           3
#define DBX_TABLE_DRILL                4
#define DBX_TABLE_REVISION_NOTES       5


//
//    Diagram Types
//
#define    DBX_DIAGRAM_LAYER_STACK       1
#define    DBX_DIAGRAM_PROFILE           2
#define    DBX_DIAGRAM_DESIGNVIEW        3


//
//    Fill Pattern Types: Returned in TRoom.roomFillPattern
//
#define DBX_ROOM_UNDEFINED_FILLPATTERN   0
#define DBX_ROOM_CLEAR_FILLPATTERN       1
#define DBX_ROOM_SOLID_FILLPATTERN       2
#define DBX_ROOM_HATCHED_FILLPATTERN     3


//
//    Placement Types: Returned in TRoom.placementSide
//
#define DBX_ROOM_UNDEFINED_PLACEMENT     0
#define DBX_ROOM_TOP_PLACEMENT           1
#define DBX_ROOM_BOTTOM_PLACEMENT        2
#define DBX_ROOM_TOP_OR_BOTTOM_PLACEMENT 3


//
//  RuleCategory
//
#define DBX_RULE_CATEGORY_UNKNOWN               0

#define DBX_RULE_CATEGORY_ERC_BUS_NET           1
#define DBX_RULE_CATEGORY_ERC_COMPONENT         2
#define DBX_RULE_CATEGORY_ERC_ELECTRICAL        3
#define DBX_RULE_CATEGORY_ERC_HIERARCHY         4
#define DBX_RULE_CATEGORY_ERC_NET_CONNECTIVITY  5
#define DBX_RULE_CATEGORY_ERC_NO_NODE_NET       6
#define DBX_RULE_CATEGORY_ERC_SINGLE_NODE_NET   7
#define DBX_RULE_CATEGORY_ERC_UNCONNECTED_PIN   8
#define DBX_RULE_CATEGORY_ERC_UNCONNECTED_WIRE  9

#define DBX_RULE_CATEGORY_DRC_CLEARANCE        10
#define DBX_RULE_CATEGORY_DRC_NETLIST          11
#define DBX_RULE_CATEGORY_DRC_UNROUTED_NETS    12
#define DBX_RULE_CATEGORY_DRC_UNCONNECTED_PINS 13
#define DBX_RULE_CATEGORY_DRC_NET_LENGTH       14
#define DBX_RULE_CATEGORY_DRC_SILK             15
#define DBX_RULE_CATEGORY_DRC_TEXT             16
#define DBX_RULE_CATEGORY_DRC_WIDTH            17
#define DBX_RULE_CATEGORY_DRC_POUR             18
#define DBX_RULE_CATEGORY_DRC_PLANE            19
```

```
#define DBX_RULE_CATEGORY_DRC_COMPONENT         20
#define DBX_RULE_CATEGORY_DRC_DRILL             21


//
//  RuleType
//
#define DBX_RULE_TYPE_UNKNOWN                    0

#define DBX_RULE_ERC_ELEC_3STATE_OUTPUT          1
#define DBX_RULE_ERC_ELEC_3STATE_POWER           2
#define DBX_RULE_ERC_ELEC_BI_OH                  3
#define DBX_RULE_ERC_ELEC_BI_OL                  4
#define DBX_RULE_ERC_ELEC_BI_OUTPUT              5
#define DBX_RULE_ERC_ELEC_BI_POWER               6
#define DBX_RULE_ERC_ELEC_OH_BI                  7
#define DBX_RULE_ERC_ELEC_OH_OH                  8
#define DBX_RULE_ERC_ELEC_OH_OL                  9
#define DBX_RULE_ERC_ELEC_OH_OUTPUT             10
#define DBX_RULE_ERC_ELEC_OH_POWER              11
#define DBX_RULE_ERC_ELEC_OL_BI                 12
#define DBX_RULE_ERC_ELEC_OL_OH                 13
#define DBX_RULE_ERC_ELEC_OL_OUTPUT             14
#define DBX_RULE_ERC_ELEC_OL_POWER              15
#define DBX_RULE_ERC_ELEC_OUTPUT_BI             16
#define DBX_RULE_ERC_ELEC_OUTPUT_OUTPUT         17
#define DBX_RULE_ERC_ELEC_OUTPUT_3STATE         18
#define DBX_RULE_ERC_ELEC_OUTPUT_OH             19
#define DBX_RULE_ERC_ELEC_OUTPUT_OL             20
#define DBX_RULE_ERC_ELEC_OUTPUT_POWER          21
#define DBX_RULE_ERC_ELEC_POWER_3STATE          22
#define DBX_RULE_ERC_ELEC_POWER_BI              23
#define DBX_RULE_ERC_ELEC_POWER_OH              24
#define DBX_RULE_ERC_ELEC_POWER_OL              25
#define DBX_RULE_ERC_ELEC_POWER_OUTPUT          26
#define DBX_RULE_ERC_ELEC_NO_INPUT              27
#define DBX_RULE_ERC_ELEC_NO_OUTPUT             28
#define DBX_RULE_ERC_NET_CON_DIFFRNT_NET_NAME   29
#define DBX_RULE_ERC_NET_CON_MERGED_NETS_NOT    30
#define DBX_RULE_ERC_NET_CON_NET_NAME_NOT_VIS   31
#define DBX_RULE_ERC_NET_CON_PIN_NAME_MISSING   32
#define DBX_RULE_ERC_HIER_EXCEED_SHEET_LIMIT    33
#define DBX_RULE_ERC_HIER_INF_SHEET_DIFF        34
#define DBX_RULE_ERC_HIER_INT_ATTR_NOT_DEFINED  35
#define DBX_RULE_ERC_HIER_INT_ILLEGAL           36
#define DBX_RULE_ERC_HIER_INT_MULTIPLE          37
#define DBX_RULE_ERC_HIER_INT_NOT_DEFINED       38
#define DBX_RULE_ERC_HIER_INT_UNUSED            39
#define DBX_RULE_ERC_HIER_MULT_SHEET            40
#define DBX_RULE_ERC_HIER_PIN_DES_MISMATCH      41
#define DBX_RULE_ERC_HIER_PIN_ELECT_TYPE        42
#define DBX_RULE_ERC_HIER_PIN_MISMATCH          43
#define DBX_RULE_ERC_HIER_RECURSIVE             44
#define DBX_RULE_ERC_HIER_SAME_SHEET            45
#define DBX_RULE_ERC_HIER_INFPIN_NOT_CONNECTED  46
#define DBX_RULE_ERC_HIER_INFPIN_PASSTHROUGH    47
#define DBX_RULE_ERC_HIER_MODPIN_NOT_CONNECTED  48
```

```
#define DBX_RULE_ERC_COMP_TWO_COMPS             49
#define DBX_RULE_ERC_BUS_NET_BUS_NO_NET         50
#define DBX_RULE_ERC_BUS_NET_BUS_SINGLE_NET     51
#define DBX_RULE_ERC_NO_NODES                   52
#define DBX_RULE_ERC_SINGLE_NODE                53
#define DBX_RULE_ERC_UNCON_PIN_BLANK_NET_NAME   54
#define DBX_RULE_ERC_UNCON_PIN_UNCONNECTED_PIN  55
#define DBX_RULE_ERC_UNCONNECTED_WIRE           56

#define DBX_RULE_DRC_CLEARANCE                  57
#define DBX_RULE_DRC_SHORT                      58
#define DBX_RULE_DRC_NET_SHORTED_TO_TIE         59
#define DBX_RULE_DRC_COMP_PINS_SHORTED          60
#define DBX_RULE_DRC_CONNECTED_POINT_TO_POINT   61
#define DBX_RULE_DRC_PSEUDO                     62
#define DBX_RULE_DRC_VIASTYLE_NONEXISTANT       63
#define DBX_RULE_DRC_VIASTYLE_VIOLATION         64
#define DBX_RULE_DRC_MAXVIAS                    65
#define DBX_RULE_DRC_UNCONNECTED_PIN            66
#define DBX_RULE_DRC_CU_POUR_CLEARANCE          67
#define DBX_RULE_DRC_CU_POUR_TERRA_INCOGNITA    68
#define DBX_RULE_DRC_CU_POUR_UNFILLED           69
#define DBX_RULE_DRC_CU_POUR_NO_NET             70
#define DBX_RULE_DRC_DRILL_CLEARANCE            71
#define DBX_RULE_DRC_DRILL_HOLE_SAME_LAYER      72
#define DBX_RULE_DRC_DRILL_HOLE_INTERFERENCE    73
#define DBX_RULE_DRC_UNROUTED_NET               74
#define DBX_RULE_DRC_HOLE_NO_CONNECT            75
#define DBX_RULE_DRC_HOLE_RANGE_VIOLATION       76
#define DBX_RULE_DRC_TIE_NET_WITH_NO_POLY       77
#define DBX_RULE_DRC_TIE_POLY_WITHOUT_NETS      78
#define DBX_RULE_DRC_NET_FREE_PIN               79
#define DBX_RULE_DRC_NET_LENGTH                 80
#define DBX_RULE_DRC_SILK_CLEARANCE             81
#define DBX_RULE_DRC_TEXT_CLEARANCE             82
#define DBX_RULE_DRC_WIDTH                      83
#define DBX_RULE_DRC_COMPONENT_SIDE             84
#define DBX_RULE_DRC_COMPONENT_HEIGHT           85
#define DBX_RULE_DRC_ROOM_INCLUDED_COMPS        86
#define DBX_RULE_DRC_EMPTY_ROOM                 87
#define DBX_RULE_DRC_PLANE_CLEARANCE            88
#define DBX_RULE_DRC_PLANE_SPLIT                89
#define DBX_RULE_DRC_PLANE_PARTIAL_CONNECTION   90
#define DBX_RULE_DRC_PLANE_HOLE_SHORT           91
#define DBX_RULE_DRC_PLANE_NO_CONNECTIONS       92
#define DBX_RULE_DRC_PLANE_NO_NET               93
#define DBX_RULE_DRC_PLANE_COPPER_INTERSECTION  94


//
//  ViolationType
//
#define DBX_ERROR_VIOLATION                      0
#define DBX_WARNING_VIOLATION                    1
#define DBX_IGNORED_VIOLATION                    2
#define DBX_OVERRIDDEN_VIOLATION                 3
```

```
//
//  Unit Definitions
//
#define DBX_UNIT_DB                0       // database units
#define DBX_UNIT_MIL               1       // mil
#define DBX_UNIT_MM                2       // millimeter
#define DBX_UNIT_CM                3       // centimeter
#define DBX_UNIT_INCH              4       // inch
#define DBX_UNIT_APP               5       //
#define DBX_UNIT_GLOBAL            6       // ie. Use gpDesign's current
                                              DBX_UNITs.
#define DBX_UNIT_LAYER             7       // String represents a layer name.
#define DBX_UNIT_VIASTYLE          8       // String represents a via style
                                              name.
#define DBX_UNIT_M                 9       // meter
#define DBX_UNIT_UM                10      // micrometer
#define DBX_UNIT_NM                11      // nanometer
#define DBX_UNIT_PM                12      // picometer
#define DBX_UNIT_RAD               13      // radian
#define DBX_UNIT_DEG               14      // degree
#define DBX_UNIT_OHM               15      // ohms
#define DBX_UNIT_MHO               16      // conductance
#define DBX_UNIT_VOLT              17      // volts
#define DBX_UNIT_MVOLT             18      // millivolts
#define DBX_UNIT_UVOLT             19      // microvolts
#define DBX_UNIT_NVOLT             20      // nanovolts
#define DBX_UNIT_PVOLT             21      // picovolts
#define DBX_UNIT_AMP               22      // ampere
#define DBX_UNIT_MAMP              23      // milliamp
#define DBX_UNIT_UAMP              24      // microamp
#define DBX_UNIT_NAMP              25      // nanoamp
#define DBX_UNIT_PAMP              26      // picoamp
#define DBX_UNIT_HENRY             27      // Henry
#define DBX_UNIT_MHENRY            28      // millihenry
#define DBX_UNIT_UHENRY            29      // microhenry
#define DBX_UNIT_NHENRY            30      // nanohenry
#define DBX_UNIT_PHENRY            31      // picohenry
#define DBX_UNIT_FARAD             32      // Farad
#define DBX_UNIT_MFARAD            33      // millifarad
#define DBX_UNIT_UFARAD            34      // microfarad
#define DBX_UNIT_NFARAD            35      // nanofarad
#define DBX_UNIT_PFARAD            36      // picofarad
#define DBX_UNIT_SECOND            37      // second
#define DBX_UNIT_MSECOND           38      // millisecond
#define DBX_UNIT_USECOND           39      // microsecond
#define DBX_UNIT_NSECOND           40      // nanosecond
#define DBX_UNIT_PSECOND           41      // picosecond
#define DBX_UNIT_BOOL              42      // Boolean
#define DBX_UNIT_QUANTITY          43      // quantity
#define DBX_UNIT_STRING            44      // string
#define DBX_UNIT_HERTZ             45      // hertz
#define DBX_UNIT_KHERTZ            46      // kilohertz
#define DBX_UNIT_MHERTZ            47      // megahertz
#define DBX_UNIT_WATT              48      // wattage
#define DBX_UNIT_MWATT             49      // milliwatt
#define DBX_UNIT_UWATT             50      // microwatt
```

```
#define DBX_UNIT_NWATT              51      // nanowatt
#define DBX_UNIT_PWATT              52      // picowatt
#define DBX_UNIT_GHERTZ             53      // gigahertz
#define DBX_UNIT_FAHRENHEIT         54      // fahrenheit
#define DBX_UNIT_CELSIUS            55      // celsius

#define DBX_UNIT_NO_UNIT            56      // a unit has not been defined for
                                               the constraint
```

Appendix B: P-CAD DBX Data Types and Globals

```
//     Conversation and Context Structure - Returned by TOpenDesign
//

typedef struct                //           (pcb/sch/cmp)
{
    HCONV     hConv;          // Converstation handle
    long      appInst;        // Application instance
    long      version;        // DBX version
    long      language;       // Language type
    HWND      hWnd;           // Server's window handle.
    HANDLE    hMmf;           // Handle to memory mapped file.
    void*     pMmf;           // Pointer to mapped view of MMF.

} DbxContext;

//
//    TCoord and TBoundRect are declared first.  They are
//    used in the definition of structures for other items
//

typedef struct  // (pcb/sch/cmp)
{
    long            x;
    long            y;
} TCoord;

typedef struct  // (pcb/sch)
{
    TCoord          lowerLeft;              //  point structures
    TCoord          upperRight;             //  as defined above
} TBoundRect;

//
//    The following structures are the data structures for
//      all extracted items, listed in alphabetical order
//

typedef struct  // (pcb)
{
    long            itemId;
    long            width;              // in database units
    long            radius;             // in database units
    TCoord          centerPt;           // in database units
    long            startAng;           // in database units
    long            sweepAng;           // in database units
    TBoundRect      boundRect;
    long            netId;
    long            layerId;
    long            isHighlighted;
} TArc ;

typedef struct  // (pcb/sch)
{
```

```
    long            itemId;                         // pcb  sch
    char            type[DBX_MAX_TYPE_LEN];         // pcb  sch
    char            value[DBX_MAX_ATTRIBUTE_LEN];   // pcb  sch
    char            formula[DBX_MAX_FORMULA_LEN];   // pcb  sch
    char            comment[DBX_MAX_TEXTITEM_LEN];  // pcb  sch
    long            typeLength;                     // pcb  sch
    long            valueLength;                    // pcb  sch
    long            formulaLength;                  // pcb  sch
    long            commentLength;                  // pcb  sch
    long            textStyleId;                    // pcb  sch
    long            justPoint;                      // pcb  sch
    TCoord          refPoint;                       // pcb  sch
    TBoundRect      boundRect;                      // pcb  sch
    long            rotationAng;                    // pcb  sch
    long            compId;                         // not used
    long            netId;                          // not used
    long            netClassId;                     // not used
    long            layerId;                        // pcb  sch
    long            isFlipped;                      // pcb  sch
    long            isHighlighted;                  // pcb  sch
    long            isVisible;                      // pcb  sch
    long            units;                          // pcb  sch; one of the
                                                       unit types defined
                                                       above
} TAttribute;

typedef struct  // (sch)
{
    long            itemId;
    TCoord          startPt;
    TCoord          endPt;
    long            layerId;
    char            busName[DBX_MAX_NAME_LEN];
    long            isFlipped;
    long            isHighlighted;
    long            isNameVisible;              //  =0 not visible
} TBus;

typedef struct  // (pcb/sch)
{
    long            netClassId1;                        // pcb  sch
    long            netClassId2;                        // pcb  sch
    char            netClassName1[DBX_MAX_NAME_LEN];    // pcb  sch
    char            netClassName2[DBX_MAX_NAME_LEN];    // pcb  sch
} TClassToClass;

typedef struct  // (pcb/sch/cmp)
{
    long            compId;                         // pcb  sch
    char            refDes[DBX_MAX_NAME_LEN];       // pcb  sch
    char            compType[DBX_MAX_TYPE_LEN];     // pcb  sch
    char            value[DBX_MAX_VALUE_LEN];       // pcb  sch
    char            patternName[DBX_MAX_NAME_LEN];  // pcb  sch
    char            libraryName[DBX_MAX_FILENAME_LEN];//pcb  sch
    TCoord          refPoint;                       // pcb
    TBoundRect      boundRect;                      // pcb
    long            rotationAng;                    // pcb
```

```
    long            numberPads;                     // pcb
    long            numberPins;                     // pcb  sch
    long            numberParts;                    // pcb  sch
    long            isAlpha;                        // pcb  sch
    long            isFlipped;                      // pcb
    long            isHighlighted;                  // pcb  sch
    long            isHetero;                       // heterogeneous vs.
                                                       homogeneous
    long            connectionType;                 // the type of component:
                                                       power, connector, or
                                                       link...
    long            isFixed;                        // pcb (only)
} TComponent;

typedef struct  // (pcb/sch/cmp)
{
    //long            itemId;
    long            compPinNumber;
    char            pinDes[DBX_MAX_NAME_LEN];
    long            gateNumber;
    long            symPinNumber;
    long            padNumber;
    char            pinName[DBX_MAX_NAME_LEN];
    long            gateEq;
    long            pinEq;
    long            electype;                   // pin type
} TCompPin;

typedef struct  // (pcb/sch)
{
    char            designName[DBX_MAX_NAME_LEN];
    char            title[DBX_MAX_NAME_LEN];
    char            designer[DBX_MAX_NAME_LEN];     //  Author
    char            version[DBX_MAX_NAME_LEN];
    char            lastModifiedDate[DBX_MAX_NAME_LEN];
    char            lastModifiedTime[DBX_MAX_NAME_LEN];
    char            drawingNumber[DBX_MAX_NAME_LEN];
    char            guidString[DBX_MAX_NAME_LEN];
    TCoord          workSpaceSize;                  //  in database units
    TBoundRect      layerExtents;                   //  in database units
    long            absGridId;                      //  absolute grid index
    long            relGridId;                      //  relative grid index
    long            isGridAbsolute;                 //  1=absolute mode;
                                                       0=relative mode
    long            userUnits;                      //  DBX_UNIT_MILS,
                                                       DBX_UNIT_MM
    TCoord          relGridOrigin;                  //  coordinates of
                                                       relative grid origin
    long            isModified;                     //  1=Design has been
                                                       modified; 0=no
                                                       modification
} TDesign;

typedef struct  // (sch)
{
    long            itemId;                     //
    long            textStyleId;                //
```

```
    char            text[DBX_MAX_TEXTITEM_LEN];  //
    long            layerId;                     //
    long            justPoint;                   //
    TCoord          refPoint;                    //
    long            rotationAng;                 //
    long            isFlipped;                   //
    long            isHighlighted;               //
    long            isVisible;                   //
    long            fieldKeyType;                //
} TField;

typedef struct  // (pcb/sch)
{
    long            gridId;                                  //  ordinal number
    char            gridSpacing[DBX_MAX_GRIDSPACING_LEN];    //  the grid
                                                                 spacing; values
                                                                 are held as a
                                                                 string
} TGrid;

typedef struct  // (pcb/sch)
{
    long            layerId;                     // pcb  sch
    char            layerName[DBX_MAX_NAME_LEN]; // pcb  sch
    long            layerType;                   // pcb
    long            layerBias;                   // pcb
    long            planeNetId;                  // pcb  =0 if none
    long            lineLineClearance;           // pcb  in database units
    long            padLineClearance;            // pcb  in database units
    long            padPadClearance;             // pcb  in database units
    long            viaPadClearance;             // pcb  in database units
    long            viaLineClearance;            // pcb  in database units
    long            viaViaClearance;             // pcb  in database units
    long            isEnabled;                   // pcb  =0 not enabled
} TLayer;

typedef struct  // (pcb/sch)
{
    long            itemId;
    long            lineType;                    //  DBX item type
    long            width;                       //  in database units
    TCoord          startPt;
    TCoord          endPt;
    TBoundRect      boundRect;
    long            netId;
    long            layerId;
    long            isHighlighted;
} TLine;

typedef struct  // (pcb/sch)
{
    long            netId;                       // pcb  sch
    char            netName[DBX_MAX_NAME_LEN];   // pcb  sch
    long            nodeCount;                   // pcb  sch
    long            length;                      // pcb  in database units
    long            isPlane;                     // pcb  =0 not plane
} TNet;
```

```
typedef struct  // (pcb/sch)
{
    long            netClassId;                     // pcb  sch
    char            netClassName[DBX_MAX_NAME_LEN]; // pcb  sch
    long            numberOfNets;                   // pcb  sch
} TNetClass;

typedef struct // (pcb)
{
    long            itemId;
    //long            padNumber; duplicated in compPin
    char            compRefDes[DBX_MAX_NAME_LEN];//      "
    long            padStyleId;
    TCoord          center;
    long            layerId;
    long            isFlipped;                  //  =0 not flipped
    long            rotationAng;                //  in database units
    long            isHighlighted;
    long            netId;
    TBoundRect      boundRect;
    TCompPin        compPin;
    char            defaultPinDes[DBX_MAX_NAME_LEN];// may be NULL
} TPad;

typedef struct  // (pcb)
{
    long            layerId;
    long            layerType;
    long            styleType;                  // Pad, Via Style
    long            shape;
    long            holeDia;                    // in database units
    long            width;                      // Normal Types, in db units
    long            height;                     // Normal Types, in db units
    long            outerDia;                   // Thermals, in db units
    long            innerDia;                   // Thermals, in db units
    long            spokeWidth;                 // Thermals, in db units
    long            isPourNoConn;               // Prohibit Cu pour
                                                thermalizing
} TPadViaShape;


typedef struct  // (pcb)
{
    long            styleId;
    long            styleType;                  // DBX_PAD or DBX_VIA
    char            name[DBX_MAX_NAME_LEN];
    long            holeDia;                    // in database units
    long            isHolePlated;               // =0 nonplated
    long            xOffset;                    // in database units
    long            yOffset;                    // in database units
    long            holeStartLayer;
    long            holeEndLayer;
    long            drillSymbol;
} TPadViaStyle;

typedef struct  // (cmp)
```

```
{
    long            itemId;
    char            patternName[DBX_MAX_NAME_LEN];
    long            rotationAng;    // for future use in PCB
    long            isFlipped;        // for future use in PCB
    long            isHighlighted;    // for future use in PCB
} TPattern;


typedef struct  // (sch/cmp)
{
    long            itemId;
    //long            symPinNumber;     // duplicated in compPin
    char            compRefDes[DBX_MAX_NAME_LEN];
    long            outsideStyle;
    long            outsideEdgeStyle;
    long            insideStyle;
    long            insideEdgeStyle;
    TCoord          refPoint;
    long            layerId;
    long            isFlipped;                    //  =0 not flipped
    long            isHighlighted;
    long            rotationAng;                  //
    long            netId;
    TBoundRect      boundRect;
    long            pinLength;
    TCompPin        compPin;
    char            defaultPinDes[DBX_MAX_NAME_LEN];// may be NULL
} TPin;

typedef struct  // (pcb/sch)
{
    long            itemId;                     //  =0 for DBX_COORDINATE
    long            x;                          //  x value in database units
    long            y;                          //  y value in database units
    long            pointType;                  //  Glue, Pick, Ref, Info
    long            number;                     // Used for InfoPoint only
    char            textInfo[DBX_MAX_TEXTITEM_LEN]; // Used for InfoPoint only
    long            layerId;
    long            isFlipped;                  //
    long            isVisible;                  //
    long            isHighlighted;
    long            ruleCategory;               // ERC/DRC rule category;
                                                    InfoPoint only; new V15
    long            ruleType;                   // ERC/DRC rule type;
                                                    InfoPoint only; new V15
    long            violationType;              // violation type of InfoPoint;
                                                    InfoPoint only; new V15
} TPoint;


typedef struct  // (pcb)
{
    long            itemId;
    long            polyType;
    long            numPts;
    TBoundRect      boundRect;
```

```
    long            netId;
    long            layerId;
    long            isHighlighted;
} TPoly ;

typedef struct // (sch)
{
    long            itemId;
    long            netId;
    long            portType;
    long            pinLength;
    long            rotationAng;
    long            isFlipped;
    long            isHighlighted;
    long            textStyleId;
    long            layerId;
    TCoord          refPoint;
} TPort;

typedef struct // (pcb)
{
    long            itemId;
    long            lineSpacing;                //  in database units
    long            lineWidth;                  //  in database units
    long            thermalType;
    long            pourType;
    long            numPts;                     //  number of outline points
    TBoundRect      boundRect;
    long            netId;
    long            layerId;
    long            isFlooded;                  //  =0 not flooded
    long            isHighlighted;              //  Highlight color: 0,1,2,3,4
} TPour;

typedef struct // (pcb/sch)
{
    long            isSelected;                    // Is PrintJob selected for
                                                       output.
    char            jobName[DBX_MAX_FILENAME_LEN]; // Print job name.
    long            isRotated;                     // Is PrintJob marked for
                                                       rotated output.
} TPrintJob;

typedef struct  // (pcb)
{
    long            roomId;
    char            roomName[DBX_MAX_NAME_LEN];
    long            layerId;
    long            numberOfIncludedComps;
    long            numberOfExcludedComps;
    TBoundRect      boundRect;
    long            placementSide;
    long            isFixed;
    long            isFlipped;
    long            isHighlighted;
    long            roomFillPattern;
    TCoord          refPoint;
```

```
    long            rotationAngle;
} TRoom;

typedef struct  // (sch/cmp)
{
    long            symbolId;
    char            symbolName[DBX_MAX_NAME_LEN];
    char            refDes[DBX_MAX_NAME_LEN];
    long            numberPins;
    long            partNumber;
    long            altType;
    TCoord          refPoint;                   // reserved for future use
    TBoundRect      boundRect;                  // reserved for future use
    long            rotationAng;                // reserved for future use
    long            isFlipped;                  // reserved for future use
    long            isHighlighted;              // reserved for future use
    long             layerId;                    // new
    char            compType[DBX_MAX_NAME_LEN];     // component name
    char libraryName[DBX_MAX_NAME_LEN];             // library name
} TSymbol;


typedef struct  // (pcb/sch)
{
    long            itemId;
    long            textStyleId;
    char            text[DBX_MAX_TEXTITEM_LEN];
    long            layerId;
    long            justPoint;
    TCoord          refPoint;
    TBoundRect      boundRect;
    long            rotationAng;                //  in database units
    long            isFlipped;                  //  =0 not flipped
    long            isVisible;                  //  =0 not visible
    long            isHighlighted;              //  Highlight color: 0,1,2,3,4
} TText;

typedef struct  // (pcb)
{
    long            styleId;
    char            name[DBX_MAX_NAME_LEN];
    long            strokePenWidth;                 //  in integer widths
    long            strokeHeight;                   //  in database units
    char            tTypeFaceName[DBX_MAX_NAME_LEN]; //  in integer widths
    long            tTypeHeight;                    //  in database units
    long            isTrueTypeAllowed;              //  = 0  TrueType not
                                                        allowed
    long            isDisplayTrueType;              //  = 0  display stroke
} TTextStyle;

typedef struct  // (pcb)
{
    long            itemId;
    long            netId;
    TCoord          center;
    long            rotationAng;
    long            viaStyleId;
```

```
    TBoundRect      boundRect;
    long            isFlipped;                  //  =0 not flipped
    long            isHighlighted;              //  Highlight color: 0,1,2,3,4
} TVia;

typedef struct  // (pcb/sch)
{
    long            itemId;
    TCoord          startPt;
    TCoord          endPt;
    long            layerId;
    long            isFlipped;                  //  =0 not flipped
    long            isHighlighted;              //  Highlight color: 0,1,2,3,4
    long            isNameVisible;              //  =0 not visible
    long            netId;
    long            width;
} TWire;


typedef struct    // (pcb/sch)
{
    long            itemId;
    TCoord          refPoint;
    long            layerId;
    TBoundRect      boundRect;
    long            rotationAng;
    long            isHighlighted;
    long            isFlipped;
    long            tableType;
    char            tableTitle[DBX_MAX_TEXTITEM_LEN];
    long            textStyleId;
    long            lineWidth;
} TTable;


typedef struct    // (pcb)
{
    long            itemId;
    TCoord          refPoint;
    long            layerId;
    TBoundRect      boundRect;
} TMetaFile;


typedef struct    // (pcb)
{
    long            itemId;
    TCoord          refPoint;
    long            layerId;
    TBoundRect      boundRect;
    long            diagramType;
    char            title[DBX_MAX_TEXTITEM_LEN];
    char            subTitle[DBX_MAX_TEXTITEM_LEN];
    long            textStyleId;    // of title, subtitle
    long            lineWidth;      // for stackup diagram type only
} TDiagram;
```

```
typedef struct    // (pcb)
{
    long            itemId;
    TCoord          refPoint;
    long            layerId;
    TBoundRect      boundRect;
    char            title[DBX_MAX_TEXTITEM_LEN];
    char            subTitle[DBX_MAX_TEXTITEM_LEN];
    char            fileName[DBX_MAX_FILENAME_LEN];
    long            textStyleId;    // of title, subtitle
} TDetail;


//
//  The following structure defines TItem, which may contain
//  any one of the supported item types.
//

typedef struct  // (pcb/sch)
{
    long            itemType;       // pcb  sch      DBX Item type
    TArc            arc;            // pcb  sch
    TAttribute      attribute;      // pcb  sch
    TBus            bus;            //      sch
    TClassToClass   classToClass;   // pcb  sch
    TComponent      component;      // pcb  sch
    TCompPin        compPin;        //      sch  clm
    TDesign         design;         // pcb  sch
    TDetail         detail;         // pcb
    TDiagram        diagram;        // pcb
    TField          field;          //      sch
    TGrid           grid;           // pcb
    TLayer          layer;          // pcb  sch
    TLine           line;           // pcb  sch
    TMetaFile       metaFile;       // pcb
    TNet            net;            // pcb  sch
    TNetClass       netClass;       // pcb  sch
    TPad            pad;            // pcb       clm
    TPadViaShape    padViaShape;    // pcb
    TPadViaStyle    padViaStyle;    // pcb
    TPattern        pattern;        //           clm
    TPin            pin;            //      sch  clm
    TPoint          point;          // pcb
    TPoly           poly;           // pcb
    TPort           port;           //      sch
    TPour           pour;           // pcb
    TPrintJob       printJob;       // pcb  sch
    TRoom           room;           // pcb
    TSymbol         symbol;         //      sch  clm
    TTable          table;          // pcb  sch
    TText           text;           // pcb  sch
    TTextStyle      textStyle;      // pcb
    TVia            via;            // pcb
    TWire           wire;           //      sch
}TItem;
```

```
extern DbxContext     tContext;       //  Global DBX Conversation Data
extern long           tStatus;        //  Global Return status variable

#define DLLX __stdcall


typedef struct  // (pcb/sch)
{
    char            name[DBX_MAX_NAME_LEN];            // Variant name
    char            description[DBX_MAX_TEXTITEM_LEN]; // Variant description
    long            id;                                // Internal id
} TVariant;
```

Appendix C: P-CAD DBX Functions

This section is excerpted directly from DBX32.H

```
//
long
DLLX TAddClassToClassAttribute     (DbxContext* pContext,       // (i/o)
                                    long      netClassId1,      // (i)
                                    long      netClassId2,      // (i)
                                    TAttribute* TAttribute);    // (i/o)
//
// TAddClassToClassAttribute - (PCB) Add an attribute (rule) to the
// given class to class.
// -------------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netClassId1         long         net  class Id number #1
//  netClassId2         long         net  class Id number #1
//  pAttribute          TAttribute*  Attribute to add
//
//  Returns             long         DBX completion status
//


long
DLLX TAddCompAttribute     (DbxContext* pContext,       // (i/o)
                                  char* pCompRefDes,    // (i)
                             TAttribute* TAttribute);    // (i/o)
//
// TAddCompAttribute - (PCB) Add an attribute to a component
// -----------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pCompRefDes         char*        Input string containing the
//                                   Component RefDes
//  pAttribute          TAttribute*  Attribute to add
//
//  Returns       long         DBX completion status
//


long
DLLX  TAddDesignAttribute     (DbxContext* pContext,  // (i) dbx context info
                               TAttribute* pTAttr);   // (i/o) design attribute
//
// TAddDesignAttribute - Add an attribute to the Design
// -------------------
//
//  parameter           Type/Description
```

```
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pAttribute          TAttribute*  Attribute to add
//
//  Returns       long         DBX completion status
//


long
DLLX  TAddIncludedRoomComponent (DbxContext* pContext,      // (i/o) dbx
                                                               context info
                                 long        roomId,        // (i) room to add
                                                               component to
                                 TComponent* pTComponent);  // (i/o) the
                                                               component to
                                                               add
//
// TAddIncludedRoomComponent - Add a component to the included
// room component list
// -------------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  roomId              long         Room to add the component to
//  pTComponent         TComponent*  Component to add
//
//  Returns       long         DBX completion status
//

long
DLLX TAddLayer      (DbxContext* pContext,
                     TLayer* pLayer);

// TAddLayer - (Pcb) Add a layer to the given design.
// -----------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pLayer              TLayer*      Layer to add.
//
//  Returns             long         DBX completion status
//                                   If layer number is
//                                   0 or already exists system auto
//                                   changes number to next available.
//                                   If layer name already exists error
//                                   is returned.
//

long
DLLX  TAddLayerAttribute     (DbxContext* pContext,     // (i) dbx context info
                              long        layerId,      // (i) layer id
                              TAttribute* pTAttr);      // (o) design attribute
```

```
//
// TAddLayerAttribute - (PCB) Add an attribute to the specified layer.
// -------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  layerId             long         layer Id number
//  pTAttr              TAttribute*  Attribute to add
//
//  Returns             long         DBX completion status
//


long
DLLX TAddNetAttribute     (DbxContext* pContext,       // (i/o)
                           long        netId,          // (i)
                           TAttribute* TAttribute);    // (i/o)
//
// TAddNetAttribute - (PCB) Add an attribute to the given net.
// ----------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netId               long         net Id number
//  pAttribute          TAttribute*  Attribute to add
//
//  Returns             long         DBX completion status
//


long
DLLX TAddNetClassAttribute     (DbxContext* pContext,       // (i/o)
                                long      netClassId,       // (i)
                               TAttribute* TAttribute);     // (i/o)
//
// TAddNetClassAttribute - (PCB) Add an attribute to the given net class.
// ---------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netClassId          long         net  class Id number
//  pAttribute          TAttribute*  Attribute to add
//
//  Returns             long         DBX completion status
//


long
DLLX TAddNetClassNet     (DbxContext* pContext,       // (i/o)
                          long      netClassId,       // (i)
                          TNet*           pTNet);     // (i/o)
```

```
//
// TAddNetClassNet - (PCB) Add a net to the given net class.
// ---------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netClassId          long         Net class Id number
//  pTNet               TNet*        Net to add
//
//  Returns             long         DBX completion status
//


long
DLLX TAddNetNode          (DbxContext* pContext,       // (i/o)
                           long        netId,          // (i)
                           TItem*      pTItem);        // (i/o)
//
// TAddNetNode - (SCH) Add a node to a specified net
// -----------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netId               long         net Id number
//  pPad                TPad*        Node to add (a free or component pad)
//
//  Returns             long         DBX completion status
//


long
DLLX  TAddRoomAttribute     (DbxContext* pContext, // (i) dbx context info
                             long        roomId,   // (i) room id
                             TAttribute* pTAttr);  // (i) room attribute to add
//
// TAddRoomAttribute - (PCB) Add an attribute to the given room.
// -----------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  roomId              long         Room Id
//  pTAttr              TAttribute*  Room attribute to add
//
//  Returns             long         DBX completion status
//

long
DLLX TAddSheet      (DbxContext* pContext,
                     TLayer* pLayer);

// TAddSheet - (Sch) Add a sheet to the given design.
```

```
// -----------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pLayer              TLayer*      Sheet to add.
//
//  Returns             long         DBX completion status
//                                   If sheet number is 0 or already
//                                   exists system auto changes number
//                                   to next available.
//                                   If sheet name already exists error
//                                   is returned.
//

long
DLLX TCloseComponent  (DbxContext*   pContext);


long
DLLX TCloseDesign(DbxContext* pContext,
                  const char* pDesignName);
//
// TCloseDesign - (PCB/Sch) Terminates communication with the application
// ------------              indicated by pContext.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation to close
//  pDesignName   char*        For future use (not used).  Input an
//                             empty string.
//
//  returns       long         DBX completion status
//


long
DLLX  TCloseLibrary(DbxContext* pContext,       //  (i/o) dbx context info
                    const char* pLibraryName);  //  (i) library name, currently
                                                    ignored


long
DLLX TCopyComponent   (DbxContext*   pContext,
                       const char*   pSCompType,
                       const char*   pSLibName,
                       const char*   pDCompType,
                       const char*   pDLibName);

long
DLLX TCopyPattern     (DbxContext*   pContext,
                       const char*   pSPatName,
                       const char*   pSLibName,
                       const char*   pDPatName,
                       const char*   pDLibName);
```

```
long
DLLX TCopySymbol      (DbxContext*   pContext,
                       const char*   pSSymbolName,
                       const char*   pSLibName,
                       const char*   pDSymbolName,
                       const char*   pDLibName);


long
DLLX TCreateClassToClass  (DbxContext* pContext,           // (i/o) dbx
                                                              context info
                           TClassToClass* pTClassToClass);  // (i/o) class to
                                                              class

//
// TCreateClassToClass - (PCB/SCH) Create a new class to class in the active
design
// -------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pClassToClass       TClassToClass*   Class to class to add
//                                   Valid fields used for CreateClassToClass:
//                                      - netClassName
//
//  Returns       long         DBX completion status
//


long
DLLX TCreateNet  (DbxContext* pContext,     // (i/o) dbx context info
                  TNet*       pTNet);       // (i/o) net to place

//
// TCreateNet - (PCB/SCH) Create a new net in the active design
// ----------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pNet                TNet*        Net to add
//                                   Valid fields used for CreateNet:
//                                      - netName
//
//  Returns       long         DBX completion status
//


long
DLLX TCreateNetClass  (DbxContext* pContext,     // (i/o) dbx context info
                       TNetClass*  pTNetClass);  // (i/o) net class to place

//
```

```
// TCreateNetClass - (PCB/SCH) Create a new net class in the active design
// ----------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pNetClass           TNetClass*   Net to add
//                                   Valid fields used for CreateNetClass:
//                                      - netClassName
//
//  Returns       long         DBX completion status
//


//
//  Delete Functions - (PCB) delete an item from the active design
//
//
//    All functions have the following parameters
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  p<item>       <DbxItem>*   DBX item to be deleted
//
//
//  returns       long         DBX completion status
//                             and p<item> updated to include have dbId=0
//
long
DLLX TDeleteArc (DbxContext* pContext,          // (i/o) dbx context info
                 TArc*       pTArc);            // (i/o) item to Delete

long
DLLX TDeleteAttribute(DbxContext* pContext,     // (i/o) dbx context info
                      TAttribute* pTAttr);      // (i/o) item to Delete

long
DLLX TDeleteBus      (DbxContext* pContext,     // (i/o) dbx context info
                      TBus*       pTBus);       // (i/o) item to Delete

long
DLLX TDeleteClassToClass (DbxContext*     pContext,         // (i/o) dbx
                                                               context info
                          TClassToClass*  pTClassToClass);  // (i/o)
                                                               classtoclass to
                                                               Delete

long
DLLX TDeleteClassToClassAttribute  (DbxContext* pContext,        // (i/o)
                                    long        netClassId1,     // (i)
                                    long        netClassId2,     // (i)
                                    TAttribute* TAttribute);     // (i/o)
//
```

```
// TDeleteClassToClassAttribute - (PCB) Delete an attribute associated with a
// ---------------------------          given class to class.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netClassId1         long         net class Id number 1
//  netClassId2         long         net class Id number 2
//  pAttribute          TAttribute*  ClassToClass attribute to delete
//
//  Returns             long         DBX completion status
//


long
DLLX TDeleteCompAttribute  (DbxContext* pContext,       // (i/o)
                                char* pCompRefDes,    // (i)
                             TAttribute* TAttribute);    // (i/o)
long
DLLX TDeleteComponent(DbxContext* pContext,     // (i/o) dbx context info
                    TComponent* pTComponent);   // (i/o) item to Delete

//
// TDeleteCompAttribute - (PCB) Delete an attribute associated with a
// --------------------         given component.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pCompRefDes         char*        Input string containing the
//                                   Component RefDes
//  pAttribute          TAttribute*  Component attribute to delete
//
//  Returns             long         DBX completion status
//


long
DLLX TDeleteDetail(DbxContext* pContext,    // (i/o) dbx context info
                   TDetail*    pTDetail);   // (i/o) item to Delete


long
DLLX  TDeleteDesignAttribute     (DbxContext* pContext,     // (i) dbx context
                                                               info
                                   TAttribute* pTAttr);      // (i/o) design
                                                               attribute
//
// TDeleteDesignAttribute - Delete an attribute associated with the Design.
// ----------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
```

```
//  pTAttr              TAttribute*  Component attribute to delete
//
//  Returns             long         DBX completion status
//


long
DLLX TDeleteDiagram(DbxContext* pContext,   // (i/o) dbx context info
                    TDiagram*   pTDiagram);    // (i/o) item to Delete


long
DLLX TDeleteField  (DbxContext* pContext,     // (i/o) dbx context info
                    TField*     pTField);     // (i/o) item to Delete


long
DLLX  TDeleteIncludedRoomComponent (DbxContext* pContext,   // (i/o) dbx
                                                               context info
                                    long        roomId,     // (i) room to
                                                               delete
                                                               component from
                                    TComponent* pTComponent); // (i/o) the
                                                               component to
                                                               delete
//
// TDeleteIncludedRoomComponent - Delete a component from the specified room's
included list.
// ----------------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  roomId              long         Room to delete component from
//  pTComponent         TComponent*  Component to delete
//
//  Returns             long         DBX completion status
//


long
DLLX TDeleteItem(DbxContext* pContext,     // (i/o) dbx context info
                 TItem*      pTItem);      // (i/o) item to Delete


long
DLLX  TDeleteLayerAttribute     (DbxContext* pContext,     // (i) dbx context
                                                              info
                                 long        layerId,      // (i) layer id
                                 TAttribute* pTAttr);      // (o) design
                                                              attribute
//
// TDeleteLayerAttribute - (PCB) Delete an attribute associated with a
// ---------------------         specified layer.
//
//  parameter           Type/Description
```

```
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  layerId             long         layer Id number
//  pTAttr              TAttribute*  layer attribute to delete
//
//  Returns             long         DBX completion status
//


long
DLLX TDeleteLine(DbxContext* pContext,     // (i/o) dbx context info
                 TLine*      pTLine);      // (i/o) item to Delete

long
DLLX TDeleteMetaFile(DbxContext* pContext,    // (i/o) dbx context info
                     TMetaFile*  pTMetaFile); // (i/o) item to Delete

long
DLLX TDeleteNet (DbxContext* pContext,     // (i/o) dbx context info
                 TNet*       pTNet);       // (i/o) net to Delete

long
DLLX TDeleteNetAttribute  (DbxContext* pContext,       // (i/o)
                           long        netId,          // (i)
                           TAttribute* TAttribute);    // (i/o)
//
// TDeleteNetAttribute - (PCB) Delete an attribute associated with a
// -------------------         given net.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netId               long         net Id number
//  pAttribute          TAttribute*  Net attribute to delete
//
//  Returns             long         DBX completion status
//


long
DLLX TDeleteNetClass (DbxContext* pContext,// (i/o) dbx context info
                      TNetClass*  pTNetClass);// (i/o) net class to Delete

long
DLLX TDeleteNetClassAttribute  (DbxContext* pContext,       // (i/o)
                                long        netClassId,     // (i)
                                TAttribute* TAttribute);    // (i/o)
//
// TDeleteNetClassAttribute - (PCB) Delete an attribute associated with a
// ------------------------         given net class.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
```

```
//  netClassId          long         net class Id number
//  pAttribute          TAttribute*  Net attribute to delete
//
//  Returns             long         DBX completion status
//


long
DLLX TDeleteNetClassNet  (DbxContext* pContext,       // (i/o)
                          long        netClassId,     // (i)
                          TNet*       pTNet);         // (i/o)
//
// TDeleteNetClassAttribute - (PCB) Delete a net from the
// ------------------------         given net class.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netClassId          long         Net class Id number
//  pTNet               TNet*        Net to delete
//
//  Returns             long         DBX completion status
//

long
DLLX TDeleteNetNode       (DbxContext* pContext,       // (i/o)
                           long        netId,          // (i)
                           TItem*      pTItem);        // (i/o)
//
// TDeleteNetNode - (PCB) Delete a node from the specified net
// --------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netId               long         net Id number
//  pPad                TPad*        Node to delete (a free or component pad)
//                                   (Note that this does not delete the pad,
//                                   it just removes the pad from the net)
//
//  Returns             long         DBX completion status
//

long
DLLX TDeletePad (DbxContext* pContext,     // (i/o) dbx context info
                 TPad*       pTPad);       // (i/o) item to Delete

long
DLLX TDeletePin (DbxContext* pContext,     // (i/o) dbx context info
                 TPin*       pTPin);       // (i/o) item to Delete

long
DLLX TDeletePoint(DbxContext* pContext,    // (i/o) dbx context info
                  TPoint*     pTPoint);    // (i/o) item to Delete
```

```
long
DLLX TDeletePort (DbxContext* pContext,    // (i/o) dbx context info
                  TPort*      pTPort);     // (i/o) item to Delete


long
DLLX  TDeleteRoomAttribute  (DbxContext* pContext, // (i) dbx context info
                             long        roomId,   // (i) room id
                             TAttribute* pTAttr);  // (i/o) room attribute
//
// TDeleteRoomAttribute - (PCB) Delete an attribute from the specified room
// --------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  roomId              long         Room Id
//  pTAttr              TAttribute*  Attribute to delete
//
//  Returns             long         DBX completion status
//

long
DLLX TDeleteSymbol (DbxContext* pContext,    // (i/o) dbx context info
                    TSymbol*    pTSymbol);   // (i/o) item to Delete

long
DLLX TDeleteTable(DbxContext* pContext,     // (i/o) dbx context info
                  TTable*     pTTable);     // (i/o) item to Delete

long
DLLX TDeleteText(DbxContext* pContext,     // (i/o) dbx context info
                 TText*      pTText);      // (i/o) item to Delete

long
DLLX TDeleteVia (DbxContext* pContext,     // (i/o) dbx context info
                 TVia*       pTVia);       // (i/o) item to Delete

long
DLLX TDeleteWire(DbxContext* pContext,     // (i/o) dbx context info
                 TWire*      pTWire);      // (i/o) item to Delete


long
DLLX TDeselectPrintJob (DbxContext * pContext,      // (i/o) dbx context info
                        const char * pJobName);     // (i)   print job name.
//
//  TDeselectPrintJob - (PCB/SCH) Deselect the print job from output list.
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pJobName      char*        Name of PCB Print Job or SCH Sheet
//
```

```
//  returns       long         DBX completion status
//
//
// Flip Functions - (PCB/SCH) Flip the input dbx item about
// the input coordinate
//
//
//    All functions have the following parameters
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTCoord       TCoord*      Point about which to flip the item.
//                             if pTCoord values = (-1,-1) the default
//                             item origin will be used as the flip point
//  p<item>       <DbxItem>*   DBX item, or TItem to be flipped
//
//
//  returns       long         DBX completion status
//                             and p<item> updated to reflect new orientation
//
// Note: TFlipComponent only applies for PCB components, and fails
// for SCH components
//
long
DLLX TFlipArc      (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i) flip point
                    TArc*       pTArc);       // (i/o) item to flip

long
DLLX TFlipAttribute(DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TAttribute* pTAttr);      // (i/o) item to flip

long
DLLX TFlipBus      (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TBus*       pTBus);       // (i/o) item to flip

long
DLLX TFlipComponent(DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TComponent* pTComponent); // (i/o) item to flip

long
DLLX TFlipField    (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TField*     pTField);     // (i/o) item to flip

long
DLLX TFlipItem     (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TItem*      pTItem);      // (i/o) item to flip

long
```

```
DLLX TFlipLine     (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TLine*      pTLine);      // (i/o) item to flip

long
DLLX TFlipPad      (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TPad*       pTPad);       // (i/o) item to flip

long
DLLX TFlipPin      (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TPin*       pTPin);       // (i/o) item to flip

long
DLLX TFlipPoint    (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TPoint*     pTPoint);     // (i/o) item to flip

long
DLLX TFlipPort     (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TPort*      pTPort);      // (i/o) item to flip

long
DLLX TFlipSymbol   (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TSymbol*    pTSymbol);    // (i/o) item to flip

long
DLLX TFlipTable    (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i) flip point
                    TTable*     pTTable);     // (i/o) item to flip

long
DLLX TFlipText     (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TText*      pTText);      // (i/o) item to flip

long
DLLX TFlipVia      (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TVia*       pTVia);       // (i/o) item to flip

long
DLLX TFlipWire     (DbxContext* pContext,     // (i/o) dbx context info
                    TCoord*     pTCoord,      // (i)   flip point
                    TWire*      pTWire);      // (i/o) item to flip


long
DLLX TGetClassToClassById(DbxContext* pContext,         // (i/o)
                      long netClassId1,                 // (i)
                      long netClassId2,                 // (i)
                      TClassToClass* pClassToClass);    // (o)
//
// TGetClassToClassById - (PCB/Sch) Get class to class given net class ids.
```

```
// ------------------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*     Input DBX conversation data
//  netClassId1         long            net class Id number #1
//  netClassId2         long            net class Id number #2
//  pClassToClass       TClassToClass*  class to class to fill in
//  Returns             long            DBX completion status
//


long
DLLX TGetCompByRefDes(DbxContext* pContext,
                      const char* pCompRefDes,
                      TComponent* pComponent);
//
// TGetCompByRefDes - (PCB/Sch) Get Component Data by Reference Designator.
// ----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pCompRefDes   char*        Input string containing the Component
//                             RefDes
//  pComponent    TComponent*  Output Component item
//
//  returns       long         DBX completion status
//

long
DLLX TGetCompByType  (DbxContext* pContext,
                      const char* pCompType,
                      TComponent* pComponent);
//
// TGetCompByType - (CMP) Get Component Data by Component name.
// ----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pCompType     char*        Input string containing the Component Name
//  pComponent    TComponent*  Output Component item
//
//  returns       long         DBX completion status
//


long
DLLX TGetCompSymbolByPartNumber (DbxContext* pContext,
                                 const char* pRefDes,
                                 long        partNumber,
                                 TSymbol*    pTSymbol);
//
```

```
// TGetCompSymbolByRefDes - (Sch) Get data for a symbol by the
// Symbol part number.
// ----------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  pRefDes       char*        Input string containing the
//                             Component RefDes (e.g. U1)
//  partNumber    long         Symbol part number (1, 2, 3...)
//  pTSymbol      TSymbol*     Output component symbol
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetCompSymbolByRefDes (DbxContext* pContext,
                             const char* pSymRefDes,
                             TSymbol*    pTSymbol);
//
// TGetCompSymbolByRefDes - (Sch) Get data for a symbol by the Symbol RefDes.
// ----------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  pSymRefDes    char*        Input string containing the
//                             Symbol RefDes (e.g. U1:A)
//  pTSymbol      TSymbol*     Output component symbol
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetDesignInfo(DbxContext* pContext,
                    TDesign* pDesignInfo);
//
// TGetDesignInfo - (PCB/Sch) Return Design Information for the Current Design.
// --------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pDesignInfo   TComponent*  Output Design Information
//
//
//  returns       long         DBX completion status
//
```

```
long
DLLX TGetFirstBusNet  (DbxContext* pContext,
                       const char* pBusName,
                       TNet*       pTNet);

long
DLLX TGetFirstClassToClass(DbxContext* pContext,
                           TClassToClass* pTClassToClass);
//
// TGetFirstClassToClass - (PCB/Sch) Return data for the first
// ---------------------             classToclass in the design.
//
//  parameter      Type/Description
//  ---------      ----------------
//
//  pContext       DbxContext*       Input DBX conversation data
//  pTClassToClass TClassToClass*    Output ClassToClass data
//
//
//  Returns        long         DBX completion status
//


long
DLLX TGetFirstClassToClassAttribute(DbxContext* pContext,       // (i/o)
                                    long        netClassId1,    // (i)
                                    long        netClassId2,    // (i)
                                    TAttribute* TAttribute);    // (o)
//
// TGetFirstClassToClassAttribute - (PCB/Sch) Get First Attribute
// (Rule) associated
// ------------------------------             with a given classToclass.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netClassId1         long         net class Id number #1
//  netClassId2         long         net class Id number #2
//  pAttribute          TAttribute*  Output net attribute
//
//  Returns             long         DBX completion status
//


long
DLLX TGetFirstCompAttribute(DbxContext* pContext,      // (i/o)
                            const char* pCompRefDes,   // (i)
                            TAttribute* TAttribute);   // (o)
//
// TGetFirstCompAttribute - (PCB/Sch) Get First Attribute Associated with a
// ----------------------             given Component.
//
//  parameter           Type/Description
//  ---------           ----------------
//
```

```
//  pContext            DbxContext*  Input DBX conversation data
//  pCompRefDes         char*        Input string containing the
//                                   Component RefDes
//  pAttribute          TAttribute*  Output component Attribute
//
//  Returns       long         DBX completion status
//


long
DLLX TGetFirstCompItem(DbxContext* pContext,
                       const char* pCompRefDes,
                       TItem* pCompItem);
//
// TGetFirstCompItem - (PCB) Get first pattern item defining a component.
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pCompRefDes   char*        Input string containing the
//                             Component RefDes
//  pCompItem     TItem*       Output component pattern item
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetFirstComponent (DbxContext* pContext,
                         TComponent* pComponent);
//
// TGetFirstComponent - (PCB/Sch) Get data for first component.
// ------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pComponent    TComponent*  Output component item
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetFirstCompPad (DbxContext* pContext,
                       const char* pCompRefDes,
                       TPad *pCompPad);
//
// TGetFirstCompPad - (PCB) Get data for a components first pad.
// ----------------
//
//  parameter     Type/Description
```

```
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  pCompRefDes   char*        Input string containing the
//                             Component RefDes
//  pCompPad      TPad*        Output component pad
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetFirstCompPin    (DbxContext* pContext,
                          const char* pCompRefDes,
                          TPin*   pTPin);
//
// TGetFirstCompPin - (Sch) Get data for a component's first Pin.
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  pCompRefDes   char*        Input string containing the
//                             Component RefDes (e.g. U1)
//  pTPin         TPin*        Output component pin
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetFirstCompSymbol (DbxContext* pContext,
                          const char* pCompRefDes,
                          TSymbol*    pTSymbol);
//
// TGetFirstCompSymbol - (Sch) Get data for a component's first symbol.
// -------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  pCompRefDes   char*        Input string containing the
//                             Component RefDes (e.g. U1)
//  pTSymbol      TSymbol*     Output component symbol
//
//
//  Returns       long         DBX completion status
//


long
DLLX  TGetFirstDesignAttribute (DbxContext*  pContext,  // (i) dbx context info
```

```
                                TAttribute*    pTAttr); // (o) dbx attribute
//
// TGetFirstDesignAttribute - (PCB/Sch) Get First global Attribute associated
// ------------------------             with the design.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pTAttr              TAttribute*  Output net attribute
//
//  Returns             long         DBX completion status
//


long
DLLX  TGetFirstGrid (DbxContext* pContext,   // (i/o) dbx context info
                     TGrid*      pTGrid);    // (o) grid data
//
// TGetFirstGrid -  Get first Grid on GridList.
// -------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pTGrid              TGrid*       Grid data
//
//  Returns             long         DBX completion status
//


long
DLLX  TGetFirstIncludedRoomComponent(DbxContext* pContext,  // (i/o) cntext
                                     long roomId,  // (i) room id
                                TComponent* pTComponent); // (o) component
                                                             data
//
// TGetFirstIncludedRoomComponent - (PCB) Get first Component on the
//                                        specified room's
// ------------------------------         Component Include list.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  roomId              long         Room Id
//  pTComponent         TComp*       Component data
//
//  Returns             long         DBX completion status
//


long
DLLX TGetFirstLayer (DbxContext* pContext,
                     TLayer* pLayer);
//
```

```
// TGetFirstLayer - (PCB/Sch) Return data for the first layer in the design.
// --------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pLayer        TLayer*      Output layer data
//
//
//  Returns       long         DBX completion status
//


long
DLLX  TGetFirstLayerAttribute (DbxContext*  pContext,  // (i) dbx context info
                               long         layerId,   // (i) layer id
                               TAttribute*  pTAttr);   // (i/o) TAttribute
//
// TGetFirstLayerAttribute - (PCB/Sch) Get the first attribute of a layer.
// -----------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  layerId       long         Input Layer number
//  pTAttr        TAttribute*  Output layer attribute
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetFirstLayerItem (DbxContext* pContext,
                         long layerId,
                         TItem* pLayerItem);
//
// TGetFirstLayerItem - (PCB/Sch) Get the first item on a layer.
// ------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  layerId       long         Input Layer number
//  pLayerItem    TItem*       Output layer item
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetFirstLayerStackup (DbxContext* pContext,
                            TLayerStackup* pStackup);
//
```

```
// TGetFirstLayer - (PCB) Return data for the first layer stackup
// --------------         in the design.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pStackup      TLayerStackup*      Output layer stackup data
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetFirstNet(DbxContext* pContext,
                  TNet* pNet);
//
// TGetFirstNet - (PCB/Sch) Return data for the first net in the design.
// ------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pNet          TNet*        Output Net data
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetFirstNetClass(DbxContext* pContext,
                  TNetClass* pNetClass);
//
// TGetFirstNetClass - (PCB/Sch) Return data for the first net class
// in the design.
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pNetClass     TNetClass*   Output NetClass data
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetFirstNetAttribute(DbxContext* pContext,       // (i/o)
                           long        netId,          // (i)
                           TAttribute* TAttribute);    // (o)
//
// TGetFirstNetAttribute - (PCB/Sch) Get First Attribute associated with a
// ---------------------             given net.
//
//  parameter           Type/Description
```

```
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netId               long         net Id number
//  pAttribute          TAttribute*  Output net attribute
//
//  Returns             long         DBX completion status
//


long
DLLX TGetFirstNetClassAttribute(DbxContext* pContext,       // (i/o)
                                long        netClassId,     // (i)
                                TAttribute* TAttribute);    // (o)
//
// TGetFirstNetClassAttribute - (PCB/Sch) Get First Attribute associated with a
// --------------------------             given net class.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netClassId          long         net class Id number
//  pAttribute          TAttribute*  Output net attribute
//
//  Returns             long         DBX completion status
//


long
DLLX  TGetFirstNetClassNet (DbxContext* pContext,   // (i/o) dbx context info.
                            long        netClassId, // (i) which net class
                            TNet*       pTNet);     // (o) net data
//
// TGetFirstNetClassNet - (PCB/Sch) Get first net in a given netclass.
// --------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  netClassId    long         Input NetClass number
//  pTNet         TNet*        Output net node item
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetFirstNetItem (DbxContext* pContext,
                       long netId,
                       TItem* pNetItem);
//
// TGetFirstNetItem - (PCB) Get first item in a net given a netID.
// ----------------
```

```
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  netId         long         Input Net number
//  pNetItem      TItem*       Output net item
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetFirstNetNode(DbxContext* pContext,
                      long netId,
                      TItem* pNetNode);
//
// TGetFirstNetNode - (PCB/Sch) Get first node in a net given netID.
// ----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  netId         long         Input Net number
//  pNetItem      TItem*       Output net node item
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetFirstPadStyle(DbxContext*      pContext,
                       TPadViaStyle*    pTPadStyle);
//
// TGetFirstPadStyle - (PCB) Get first pad style
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*   Input DBX conversation data
//  pTPadStyle    TPadViaStyle* Output pad style
//
//
//  Returns       long          DBX completion status
//

long
DLLX TGetFirstPattern (DbxContext*  pContext,
                       TPattern*    pTPattern);

long
DLLX TGetFirstPolyPoint(DbxContext* pContext,
                        long polyId,
```

```
                        TPoint* pPolyPoint );
//
// TGetFirstPolyPoint - (PCB) Get first point defining a polygon.
// ------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  polyId        long         Input Polygon ID number
//  pPolyPoint    TPoint*      Output polygon point
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetFirstPrintJob(DbxContext * pContext,
                       TPrintJob  * pPrintJob );
//
// TGetFirstPrintJob - (PCB/SCH) Get first print job.
// ------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pPrintJob     TPrintJob*   Output print job.
//
//
//  Returns       long         DBX completion status
//


long
DLLX  TGetFirstRoom (DbxContext* pContext,    // (i/o) dbx context info.
                     TRoom*      pTRoom);     // (o) room data
//
// TGetFirstRoom - (PCB) Get first room.
// -------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTRoom        TRoom*       Output room.
//
//
//  Returns       long         DBX completion status
//


long
DLLX  TGetFirstRoomAttribute (DbxContext*  pContext,    // (i) dbx context info
                              long         roomId,      // (i) room id
                              TAttribute*  pTAttr);     // (o) attribute to
```

```
                                                                    return
//
// TGetFirstRoomAttr - (PCB) Get first room attribute.
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  roomId        long         Room identifier
//  pTAttr        TAttribute*  Attribute data
//
//  Returns       long         DBX completion status
//


long
DLLX  TGetFirstRoomPoint (DbxContext* pContext,   // (i/o) context
                                long roomId,   // (i) room dbid
                              TPoint* pTPoint);  // (o) point data
//
// TGetFirstRoomPoint - (PCB) Get first room point--the first of
// the defining points.
// ------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  roomId        long         Room dbId
//  pTPoint       TPoint*      Point data
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetFirstSelectedItem (DbxContext* pContext,
                            TItem* pSelectItem);
//
// TGetFirstSelectedItem - (PCB/Sch) Get the first item in the selection list
// ---------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pLayerItem    TItem*       Output selected item
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetFirstSymAttribute(DbxContext* pContext,      // (i/o)
```

```
                            const char* pCompRefDes,   // (i)
                            TAttribute* TAttribute);   // (o)
//
// TGetFirstSymAttr         - (Sch) Get First Attribute Associated with a
// -----------------------             given Symbol.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pCompRefDes         char*        Input string containing the
//                                   Component RefDes:PartNumber
//  pAttribute          TAttribute*  Output component Attribute
//
//  Returns       long         DBX completion status
//


long
DLLX TGetFirstSymbol  (DbxContext*  pContext,
                       TSymbol*     pTSymbol);

long
DLLX TGetFirstSymbolPin  (DbxContext* pContext,
                          const char* pSymbolRefDes,
                          TPin*       pTPin);
//
// TGetFirstSymbolPin - (Sch) Get data for a symbols's first Pin.
// ------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  pSymbolRefDes char*        Input string containing the
//                             Symbol RefDes (e.g. U1:A)
//  pTPin         TPin*        Output component pin
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetFirstViaStyle(DbxContext*      pContext,
                       TPadViaStyle*    pTViaStyle);
//
// TGetFirstViaStyle - (PCB) Get first via style
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*   Input DBX conversation data
//  pTViaStyle    TPadViaStyle* Output via style
//
//
```

```
//  Returns       long          DBX completion status
//

long
DLLX TGetLayerById  (DbxContext* pContext,
                     long layerId,
                     TLayer* pLayer);
//
// TGetLayerByID - (PCB/Sch) Return layer data by layer number.
// -------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  layerId       long         Input layer number
//  pLayer        TLayer*      Output layer data
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetLayerByName(DbxContext* pContext,
                     const char* pLayerName,
                     TLayer* pLayer);
//
// TGetLayerByName - (PCB/Sch) Return layer data by layer name.
// ---------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  layerName     char*        Input string containing Layer name
//  pLayer        TLayer*      Output layer data
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNetById  (DbxContext* pContext,
                   long netId,
                   TNet* pNet);
//
// TGetNetByID - (PCB/Sch) Get net data by net ID.
// -----------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  NetId         long         Input net number
//  pNet          TNet*        Output net data
```

```
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNetByName(DbxContext* pContext,
                   const char* pNetName,
                   TNet* pNet);
//
// TGetNetByName - (PCB/Sch) Return design net data by net name.
// -------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pNetName      char*        Input string containing net name
//  pNet          TNet*        Output net data
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNetClassById (DbxContext* pContext,
                            long netClassId,
                       TNetClass* pNetClass);
//
// TGetNetClassById - (PCB/Sch) Get net data by net class ID.
// ----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  NetClassId    long         Input net class number
//  pNetClass     TNetClass*   Output net class data
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextBusNet   (DbxContext* pContext,
                       TNet*       pTNet);

long
DLLX TGetNextClassToClass(DbxContext* pContext,
                          TClassToClass* pTClassToClass);
//
// TGetNextClassToClass - (PCB/Sch) Return data for the next classToclass
// in the design.
// --------------------
```

```
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*       Input DBX conversation data
//  pClassToClass TClassToClass*    Output ClassToClass data
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextClassToClassAttribute(DbxContext* pContext,       // (i/o)
                                   TAttribute* TAttribute);    // (o)
//
// TGetNextClassToClassAttribute - (PCB/Sch) Get Next Attribute
// ------------------------------   (Rule) associated with a given classToclass.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pAttribute          TAttribute*  Output net attribute
//
//  Returns             long         DBX completion status
//


long
DLLX TGetNextCompAttribute (DbxContext* pContext,      // (i/o)
                            TAttribute* TAttribute);   // (o)
//
// TGetNextCompAttribute - (PCB/Sch) Get the next Attribute Associated with a
// ---------------------              given Component.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pAttribute          TAttribute*  Output component Attribute
//
//  Returns             long         DBX completion status
//


long
DLLX TGetNextComponent (DbxContext* pContext,
                        TComponent* pComponent);
//
// TGetNextComponent - (PCB/Sch) Get next component.  Must be preceded by
// -----------------              TGetFirstComponent.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
```

```
//  pComponent    TComponent*  Output component item
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextCompItem (DbxContext* pContext,
                       TItem* pCompItem);
//
// TGetNextCompItem - (PCB) Get next item defining a component pattern.
// ----------------         Must be preceded by TGetFirstCompItem.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pCompItem     TItem*       Output component item
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextCompPad (DbxContext* pContext,
                      TPad *pCompPad);
//
// TGetNextCompPad - (PCB) Get data for next pad on the same component.
// ---------------         Must be preceded by TGetFirstCompPad.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pCompPad      TPad*        Output component pad
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetNextCompPin    (DbxContext* pContext,
                         TPin*   pTPin);
//
// TGetNextCompPin - (Sch) Get data for a component's Next Pin.
// ---------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTPin         TPin*        Output component pin
//
//
```

```
//  Returns       long         DBX completion status
//


long
DLLX TGetNextCompSymbol  (DbxContext* pContext,
                          TSymbol*    pTSymbol);
//
// TGetFirstCompSymbol - (Sch) Get data for a component's next symbol.
// -------------------         Must be preceded by GetFirstCompSymbol().
//
//  parameter     Type/Description
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTSymbol      TSymbol*     Output component symbol
//
//
//  Returns       long         DBX completion status
//


long
DLLX  TGetNextDesignAttribute (DbxContext*  pContext,  // (i) dbx context info
                               TAttribute*    pTAttr); // (o) dbx attribute
//
// TGetNextDesignAttribute - (PCB/Sch) Get Next global Attribute associated
// -----------------------             with the design.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pTAttr              TAttribute*  Output net attribute
//
//  Returns             long         DBX completion status
//


long
DLLX  TGetNextGrid (DbxContext* pContext,  //  (i/o) dbx context info
                    TGrid*      pTGrid);   //  (o) Grid data
//
// TGetNextGrid -  Get next Grid on GridList.
// ------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pTGrid              TGrid*       Grid data
//
//  Returns             long         DBX completion status
//
```

```
long
DLLX  TGetNextIncludedRoomComponent (DbxContext* pContext,   // (i/o) context
                                     TComponent* pTComponent); // (o) component
                                                                  data
//
// TGetNextIncludedRoomComponent - (PCB) Get Next Component on the Rooms
// ----------------------------          Included Component list.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pTComponent         TComponent*  Output room component
//
//  Returns             long         DBX completion status
//



long
DLLX TGetNextLayer (DbxContext* pContext,
                    TLayer* pLayer);
//
// TGetNextLayer - (PCB/Sch) Return next layer data.  Must be preceded by
// -------------             TGetFirstLayer.
//
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pLayer        TLayer*      Output layer item
//
//
//  Returns       long         DBX completion status
//


long
DLLX  TGetNextLayerAttribute (DbxContext*  pContext,  // (i) dbx context info
                              TAttribute*    pTAttr); // (o) TAttribute
//
//
// TGetNextLayerAttribute - (PCB/Sch) Get the next attribute of a layer.
// ----------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTAttr        TAttribute*  Output layer attribute
//
//
//  Returns       long         DBX completion status
//
```

```
long
DLLX TGetNextLayerItem (DbxContext* pContext,
                        TItem* pLayerItem);
//
//
// TGetNextLayerItem - (PCB/Sch) Get the next item on a layer.
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pLayerItem    TItem*       Output layer item
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetNextLayerStackup (DbxContext* pContext,
                           TLayerStackup* pStackup);
//
// TGetNextLayer - (PCB) Return next layer stackup data.  Must be preceded by
// -------------              TGetFirstLayerStackup.
//
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pLayer        TLayer*      Output layer stackup item
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetNextNet (DbxContext* pContext,
                  TNet* pNet);
//
// TGetNextNet - (PCB/Sch) Return next net data.  Must be preceded by
// -----------             GetFirstNet.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pNet          TNet*        Output Net data
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextNetAttribute (DbxContext* pContext,       // (i/o)
```

```
                          TAttribute* TAttribute);    // (o)
//
// TGetNextNetAttribute - (PCB/Sch) Get the next Attribute associated with a
// --------------------             given net.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pAttribute          TAttribute*  Output component Attribute
//
//  Returns             long         DBX completion status
//


long
DLLX TGetNextNetClass (DbxContext* pContext,
                       TNetClass* pNetClass);
//
// TGetNextNetClass - (PCB/Sch) Return next net class data.
// Must be preceded by
// ----------------             GetFirstNetClass.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pNetClass     TNetClass*   Output NetClass data
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextNetClassAttribute (DbxContext* pContext,       // (i/o)
                                TAttribute* TAttribute);    // (o)
//
// TGetNextNetClassAttribute - (PCB/Sch) Get the next Attribute
//  ------------------------    associated with a given net class.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pAttribute          TAttribute*  Output component Attribute
//
//  Returns             long         DBX completion status
//


long
DLLX  TGetNextNetClassNet (DbxContext* pContext,   // (i/o) dbx context info.
                           TNet*       pTNet);     // (o) net data
//
// TGetNextNetClassNet - (PCB/Sch) Get next net in a given netclass.
// -------------------
```

```
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTNet         TNet*        Output net node item
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextNetItem (DbxContext* pContext,
                      TItem* pNetItem);
//
// TGetNextNetItem - (PCB) Get the next item from same net.  Must be
// ---------------         preceded by TGetFirstNetItem.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pNetItem      TItem*       Output net item
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetNextNetNode(DbxContext* pContext,
                     TItem* pNetNode);
//
// TGetNextNetNode - (PCB/Sch) Get next node from a net.  Must be
// ---------------             preceded by TGetFirstNetNode.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pNetItem      TItem*       Output net node item
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextPadStyle(DbxContext*      pContext,
                      TPadViaStyle*    pTPadStyle);
//
// TGetNextPadStyle - (PCB) Get next pad style
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
```

```
//  pContext      DbxContext*   Input DBX conversation data
//  pTPadStyle    TPadViaStyle* Output pad style
//
//
//  Returns       long          DBX completion status
//


long
DLLX TGetNextPattern  (DbxContext*  pContext,
                       TPattern*    pTPattern);

long
DLLX TGetNextPolyPoint(DbxContext* pContext,
                       TPoint* pPolyPoint);
//
// TGetNextPolyPoint - (PCB) Get next point from a polygon.  Must be
// -----------------         preceded by TGetFirstPolyPoint.
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pPolyPoint    TPoint*      Output polygon point
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextPrintJob(DbxContext * pContext,
                      TPrintJob  * pPrintJob );
//
// TGetNextPrintJob - (PCB/SCH) Get next print job.  Must be
// ------------------           preceded by TGetFirstPrintJob.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pPrintJob     TPrintJob*   Output print job.
//
//
//  Returns       long         DBX completion status
//


long
DLLX  TGetNextRoom (DbxContext* pContext,    // (i/o) dbx context info.
                    TRoom*      pTRoom);     // (o) room data
//
// TGetNextRoom - (PCB) Get next room in the selection list.
// ------------
//
//  parameter     Type/Description
//  ---------     ----------------
```

```
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTRoom        TRoom*       Output room.
//
//
//  Returns       long         DBX completion status
//


long
DLLX  TGetNextRoomAttribute (DbxContext*  pContext,    // (i) dbx context info
                             TAttribute*  pTAttr);     // (o) attribute to
                                                          return
//
// TGetNextRoomAttr - (PCB) Get next room attribute from the list of
attributes.
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTAttr        TAttribute*  Attribute data
//
//  Returns       long         DBX completion status
//


long
DLLX  TGetNextRoomPoint (DbxContext* pContext,   // (i/o) context
                              TPoint* pTPoint);  // (o) point data
//
// TGetNextRoomPoint - (PCB) Get next room point in the selection list.
// -----------------         This is a defining point of the room.
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTPoint       TPoint*      Point data
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextSelectedItem (DbxContext* pContext,
                           TItem* pSelectItem);
//
// TGetNextSelectedItem - (PCB/Sch) Get the next item in the selection list
// --------------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
```

```
//  pLayerItem    TItem*       Output selected item
//
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextSymAttribute (DbxContext* pContext,      // (i/o)
                           TAttribute* TAttribute);   // (o)
//
// TGetNextSymAttr            - (Sch) Get the next Attribute Associated with a
// ---------------------             given Symbol.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pAttribute          TAttribute*  Output component Attribute
//
//  Returns       long         DBX completion status
//


long
DLLX TGetNextSymbol   (DbxContext*  pContext,
                       TSymbol*     pTSymbol);

long
DLLX TGetNextSymbolPin    (DbxContext* pContext,
                           TPin*       pTPin);
//
// TGetNextSymbolPin - (Sch) Get data for a symbols's Next Pin.
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTPin         TPin*        Output symbol pin
//
//
//  Returns       long         DBX completion status
//

long
DLLX TGetNextViaStyle(DbxContext*      pContext,
                      TPadViaStyle*    pTViaStyle);
//
// TGetNextViaStyle - (PCB) Get next via style
// -----------------
//
//  parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*   Input DBX conversation data
```

```
//  pTViaStyle    TPadViaStyle* Output via style
//
//
//  Returns       long          DBX completion status
//

long
DLLX TGetPadShapeByLayer(DbxContext* pContext,
                         long padStyleId,
                         long layerId,
                         TPadViaShape* pPadShape);
//
// TGetPadShapeByLayer - (PCB) Get pad shape information for a pad
// -------------------   style on a given layer.
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*   Input DBX conversation data
//  padStyleId    long          Input pad style ID number
//  layerId       long          Input layer number
//  pPadShape     TPadViaShape* Output pad shape data
//
//
//  Returns       long          DBX completion status
//


long
DLLX TGetPadStyle (DbxContext* pContext,
                   long padStyleId,
                   TPadViaStyle* pPadStyle);
//
// TGetPadStyle  - (PCB) Return pad style data given a PadStyleID.
// ------------
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*   Input DBX conversation data
//  padStyleId    long          Input pad style ID number
//  pPadStyle     TPadViaStyle* Output pad style data
//
//
//  Returns       long          DBX completion status
//


long
DLLX TGetPatternByName(DbxContext*  pContext,
                       const char*  pPatternName,
                       TPattern*    pTPattern);


long
DLLX TGetSymbolByName (DbxContext*  pContext,
                       const char*  pSymbolName,
```

```
                        TSymbol*     pTSymbol);

long
DLLX TGetTextStyle (DbxContext* pContext,
                    long textStyleId,
                    TTextStyle* pTextStyle);
//
// TGetTextStyle - (PCB) Return text style data given a TextStyleID.
// -------------
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*   Input DBX conversation data
//  textStyleId   long          Input text style ID number
//  pTextStyle    TTextStyle*   Output text style data
//
//
//  Returns       long          DBX completion status
//


long
DLLX TGetViaShapeByLayer(DbxContext* pContext,
                         long viaStyleId,
                         long layerId,
                         TPadViaShape* pViaShape);
//
// TGetViaShapeByLayer - (PCB) Get via shape information for a via
// -------------------         style on a given layer.
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*   Input DBX conversation data
//  viaStyleId    long          Input via style ID number
//  layerId       long          Input layer number
//  pViaShape     TPadViaShape* Output via shape data
//
//
//  Returns       long          DBX completion status
//


long
DLLX TGetViaStyle (DbxContext* pContext,
                   long viaStyleId,
                   TPadViaStyle* pViaStyle);
//
// TGetViaStyle - (PCB) Return via style data given a ViaStyleID.
// ------------
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*   Input DBX conversation data
//  viaStyleId    long          Input via style ID number
```

```
//  pViaStyle     TPadViaStyle* Output via style data
//
//
//  Returns       long          DBX completion status
//


//
//  Highlight Functions - (PCB/SCH) highlight the input DBX item
//
//
//    All functions have the following parameters
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  color         long         highlight color (1 or 2)
//  p<item>       <DbxItem>*   DBX item to be moved
//
//  returns       long         DBX completion status
//                             and p<item> updated to reflect highlight status
//
// Note: THighlightComponent only applies for PCB components,
// and fails for SCH components
//


long
DLLX THighlightArc (DbxContext* pContext,     // (i/o) dbx context info
                    long        color,        // (i)   highlight color
                    TArc*       pTArc);       // (i/o) item to highlight

long
DLLX THighlightAttribute(DbxContext* pContext,// (i/o) dbx context info
                         long        color,   // (i)   highlight color
                         TAttribute* pTAttr); // (i/o) item to highlight

long
DLLX THighlightBus      (DbxContext* pContext,// (i/o) dbx context info
                         long        color,   // (i)   highlight color
                         TBus*       pTBus);  // (i/o) item to highlight

long
DLLX THighlightComponent(DbxContext* pContext,     // (i/o) dbx context info
                    long        color,        // (i)   highlight color
                    TComponent* pTComponent); // (i/o) item to highlight

long
DLLX THighlightDetail (DbxContext* pContext,      // (i/o) dbx context info
                       long        color,         // (i)   highlight color
                       TDetail*    pTDetail);     // (i/o) item to unhighlight

long
DLLX THighlightDiagram (DbxContext* pContext,      // (i/o) dbx context info
                        long        color,         // (i)   highlight color
```

```
                        TDiagram*   pTDiagram);    // (i/o) item to unhighlight

long
DLLX THighlightField  (DbxContext* pContext,     // (i/o) dbx context info
                   long        color,        // (i)   highlight color
                   TField*     pTField);     // (i/o) item to highlight

long
DLLX THighlightItem(DbxContext* pContext,     // (i/o) dbx context info
                   long        color,        // (i)   highlight color
                   TItem*      pTItem);      // (i/o) item to highlight

long
DLLX THighlightLine(DbxContext* pContext,     // (i/o) dbx context info
                   long        color,        // (i)   highlight color
                   TLine*      pTLine);      // (i/o) item to highlight

long
DLLX THighlightMetaFile (DbxContext* pContext,      // (i/o) dbx context info
                         long        color,         // (i)   highlight color
                         TMetaFile*  pTMetaFile);   // (i/o) item to
                                                       unhighlight

long
DLLX THighlightNet (DbxContext* pContext,     // (i/o) dbx context info
                   long        color,        // (i)   highlight color
                   TNet*       pTNet);       // (i/o) net to highlight

long
DLLX THighlightPad (DbxContext* pContext,     // (i/o) dbx context info
                   long        color,        // (i)   highlight color
                   TPad*       pTPad);       // (i/o) item to highlight

long
DLLX THighlightPin (DbxContext* pContext,     // (i/o) dbx context info
                   long        color,        // (i)   highlight color
                   TPin*       pTPin);       // (i/o) item to highlight

long
DLLX THighlightPoint(DbxContext* pContext,    // (i/o) dbx context info
                   long        color,        // (i)   highlight color
                   TPoint*     pTPoint);     // (i/o) item to highlight

long
DLLX THighlightPort (DbxContext* pContext,    // (i/o) dbx context info
                   long        color,        // (i)   highlight color
                   TPort*      pTPort);      // (i/o) item to highlight

long
DLLX  THighlightRoom  (DbxContext* pContext,  // (i/o) dbx context info
                       long        color,     // (i)   highlight color
                       TRoom*      pTRoom);   // (i/o) object to highlight

long
DLLX THighlightSymbol (DbxContext* pContext,     // (i/o) dbx context info
                       long        color,        // (i)   highlight color
                       TSymbol*    pTSymbol);    // (i/o) item to highlight
```

```
long
DLLX THighlightTable(DbxContext* pContext,    // (i/o) dbx context info
                     long        color,       // (i)   highlight color
                     TTable*     pTTable);    // (i/o) item to highlight

long
DLLX THighlightText(DbxContext* pContext,     // (i/o) dbx context info
                    long        color,        // (i)   highlight color
                    TText*      pTText);      // (i/o) item to highlight

long
DLLX THighlightVia (DbxContext* pContext,     // (i/o) dbx context info
                    long        color,        // (i)   highlight color
                    TVia*       pTVia);       // (i/o) item to highlight

long
DLLX THighlightWire(DbxContext* pContext,     // (i/o) dbx context info
                    long        color,        // (i)   highlight color
                    TWire*      pTWire);      // (i/o) item to highlight


long
DLLX TModifyArc (DbxContext* pContext,          // (i/o) dbx context info
                 TArc*       pTArc);            // (i/o) item to modify
//
//  TModifyArc - (PCB/SCH) modify the input DBX arc
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTArc         TArc*        DBX item to be modified
//                             Valid fields to modify:
//                                  - width
//                                  - radius
//                                  - centerPt
//                                  - startAng
//                                  - sweepAng
//                                  - layerId
//
//  returns       long         DBX completion status
//                             and pTArc updated to reflect changes
//


long
DLLX TModifyAttribute(DbxContext* pContext,     // (i/o) dbx context info
                      TAttribute* pTAttr);      // (i/o) item to modify
//
//  TModifyAttribute - (PCB/SCH) modify the input DBX design attribute
//                               (Use TModifyCompAttribute for component
//                                attributes)
//
//
//  Parameter     Type/Description
```

```
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTAttr        TAttribute*  DBX item to be modified
//                             Valid fields to modify:
//                                  - value
//                                  - refPoint
//                                  - textStyleId
//                                  - justPoint
//                                  - isVisible
//                                  - layerId
//
//  returns       long         DBX completion status
//                             and pAttr updated to reflect changes
//

long
DLLX TModifyBus      (DbxContext* pContext,     // (i/o) dbx context info
                      TBus*       pTBus);       // (i/o) item to modify
//
//  TModifyBus - (SCH)      modify the input DBX schematic bus
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTBus         TBus*        DBX item to be modified
//                             Valid fields to modify:
//                                  - startPt
//                                  - endPt
//                                  - busName
//                                  - isVisible
//                                  - isVisible
//
//  returns       long         DBX completion status
//                             and pTBus updated to reflect changes
//


long
DLLX TModifyClassToClassAttribute  (DbxContext* pContext,       // (i/o)
                                    long        netClassId1,    // (i)
                                    long        netClassId2,    // (i)
                                    TAttribute* TAttribute);    // (i/o)
//
// TModifyClassToClassAttribute - (PCB) Modify an attribute associated to the
// ----------------------------         given class to class.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netClassId1         long         net class Id number 1
//  netClassId2         long         net class Id number 2
//  pAttribute          TAttribute*  Attribute to modify
//                                   Valid fields to modify:
```

```
//                                        - value
//
//  Returns             long         DBX completion status
//


long
DLLX TModifyCompAttribute  (DbxContext* pContext,       // (i/o)
                                  char* pCompRefDes,    // (i)
                             TAttribute* TAttribute);   // (i/o)
//
// TModifyCompAttribute - (PCB) Modify a component attribute
// --------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pCompRefDes         char*        Input string containing the
//                                   Component RefDes
//  pAttribute          TAttribute*  Attribute to modify
//                             Valid fields to modify:
//                                  - value
//                                  - refPoint
//                                  - textStyleId
//                                  - justPoint
//                                  - isVisible
//                                  - layerId
//
//
//  Returns       long         DBX completion status
//


long
DLLX TModifyComponent(DbxContext* pContext,     // (i/o) dbx context info
                      TComponent* pTComponent); // (i/o) item to modify
//
//  TModifyComponent - (PCB) modify the input DBX Component
//  ----------------
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTComponent   TComponent*  DBX item to be modified
//                             Valid fields to modify:
//                                  - refDes
//                                  - value
//                                  - refPoint (moves component location)
//                                  - isFixed  (allows the component to move)
//                                               new for v14.00
//
//  returns       long         DBX completion status
//                             and pTComponent updated to reflect changes
```

```
long
DLLX  TModifyDesignAttribute     (DbxContext* pContext,     // (i) dbx context
                                                             info
                                  TAttribute* pTAttr);      // (i/o) design
                                                             attribute
//
// TModifyDesignAttribute - Modify a Design attribute
// ----------------------
//
//  parameter       Type/Description
//  ---------       ----------------
//
//  pContext        DbxContext*  Input DBX conversation data
//  pTAttr          TAttribute*  Attribute to modify
//                               Valid fields to modify:
//                                  - value
//
//  Returns         long         DBX completion status
//


long
DLLX TModifyDesignInfo (DbxContext * pContext,
                        TDesign    * pDesignInfo);
//
//  TModifyDesignInfo - (PCB/SCH) modify the design info.
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pDesignInfo   TDesign*     DBX Design info to be modified
//                             Valid fields to modify:
//                                  - isModified
//
//  returns       long         DBX completion status
//                             and pDesignInfo updated to reflect changes

long
DLLX TModifyField (DbxContext* pContext,     // (i/o) dbx context info
                   TField*     pTField);     // (i/o) item to modify
//
//  TModifyField - (SCH) modify the input DBX Field
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTField       TField*      DBX item to be modified
//                             Valid fields to modify:
//                                  -
//                                  -
//
//  returns       long         DBX completion status
//                             and pTField updated to reflect changes
```

```
long
DLLX  TModifyLayerAttribute     (DbxContext* pContext,     // (i) dbx context
                                                                info
                                 long        layerId,      // (i) layer id
                                 TAttribute* pTAttr);      // (o) design
                                                                attribute
//
// TModifyLayerAttribute - (PCB) Modify an attribute associated to the
// ---------------------         specified layer.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  layerId             long         layer Id number
//  pTAttr              TAttribute*  Attribute to modify
//                                   Valid fields to modify:
//                                      - value
//
//  Returns             long         DBX completion status
//


long
DLLX TModifyItem (DbxContext* pContext,     // (i/o) dbx context info
                  TItem*      pTItem);      // (i/o) item to modify
//
//  TModifyItem - (PCB/SCH) modify the input DBX Item
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTItem        TItem*       DBX item to be modified
//                             Valid fields to modify:
//                                  - see the individual item description
//
//  returns       long         DBX completion status
//                             and pTItem updated to reflect changes
//
//  Move Functions - (PCB) Move the input dbx item by the specified (x,y)
//
//
//    All functions have the following parameters
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  dx            long         direction in the x direction to move
//                             (in db units)
//  dy            long         direction in the y direction to move
//                             (in db units)
```

```
//  p<item>       <DbxItem>*   DBX item to be moved
//
//
//  returns       long         DBX completion status
//                             and p<item> updated to reflect changes
//


long
DLLX TModifyLine (DbxContext* pContext,     // (i/o) dbx context info
                  TLine*      pTLine);      // (i/o) item to modify
//
//  TModifyLine - (PCB/SCH) modify the input DBX Line
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTLine        TLine*       DBX item to be modified
//                             Valid fields to modify:
//                                  - width
//                                  - startPt
//                                  - endPt
//                                  - layerId     (PCB only)
//                                  - line style  (SCH only)
//
//  returns       long         DBX completion status
//                             and pTLine updated to reflect changes


long
DLLX TModifyNet  (DbxContext* pContext,     // (i/o) dbx context info
                  TNet*       pTNet);       // (i/o) net to modify
//
//  TModifyNet - (PCB) modify the input DBX Net
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTNet         TNet*       DBX item to be modified
//                             Valid fields to modify:
//                                  - netName
//
//  returns       long         DBX completion status
//                             and pTNet updated to reflect changes


long
DLLX TModifyNetAttribute  (DbxContext* pContext,       // (i/o)
                           long        netId,          // (i)
                           TAttribute* TAttribute);    // (i/o)
//
// TModifyNetAttribute - (PCB) Modify an attribute associated to the
// -------------------          given net.
```

```
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netId               long         net Id number
//  pAttribute          TAttribute*  Attribute to modify
//                                   Valid fields to modify:
//                                      - value
//
//  Returns             long         DBX completion status
//


long
DLLX TModifyNetClass  (DbxContext* pContext,     // (i/o) dbx context info
                       TNetClass*  pTNetClass);  // (i/o) net class to modify
//
//  TModifyNetClass - (PCB) modify the input DBX NetClass
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTNetClass    TNetClass*   DBX item to be modified
//                             Valid fields to modify:
//                                  - netClassName
//
//  returns       long         DBX completion status
//                             and pTNetClass updated to reflect changes


long
DLLX TModifyNetClassAttribute  (DbxContext* pContext,       // (i/o)
                                long        netId,          // (i)
                                TAttribute* TAttribute);    // (i/o)
//
// TModifyNetClassAttribute - (PCB) Modify an attribute associated to the
// ------------------------         given net class.
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  netClassId          long         net class Id number
//  pAttribute          TAttribute*  Attribute to modify
//                                   Valid fields to modify:
//                                      - value
//
//  Returns             long         DBX completion status
//


long
DLLX TModifyPad  (DbxContext* pContext,     // (i/o) dbx context info
                  TPad*       pTPad);       // (i/o) item to modify
```

```
//
//  TModifyPad - (PCB) modify the input DBX Pad
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTPad         TPad*        DBX item to be modified
//                             Valid fields to modify:
//                                  - styleId
//                                  - location (valid for free pads only)
//                                  - padNum   (valid for free pads only)
//                                  - defaultPinDes
//
//  returns       long         DBX completion status
//                             and pTPad updated to reflect changes


long
DLLX TModifyPin  (DbxContext* pContext,     // (i/o) dbx context info
                  TPin*       pTPin);       // (i/o) item to modify
//
//  TModifyPin - (SCH) modify the input DBX Pin
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTPin         TPin*        DBX item to be modified
//                             Valid fields to modify:
//                                  - Pin type
//                                  - pin designator
//                                  - pin equivalency
//                                  - defaultPinDes
//
//  returns       long         DBX completion status
//                             and pTPin updated to reflect changes


long
DLLX TModifyPoint(DbxContext* pContext,     // (i/o) dbx context info
                  TPoint*     pTPoint);     // (i/o) item to modify
//
//  TModifyPoint - (PCB) modify the input DBX Point
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTPoint       TPoint*      DBX item to be modified
//                             Valid fields to modify:
//                                  - x
//                                  - y
//
```

```
//  returns       long         DBX completion status
//                             and pTPoint updated to reflect changes


long
DLLX TModifyPort (DbxContext* pContext,     // (i/o) dbx context info
                  TPort*      pTPort);      // (i/o) item to modify
//
//  TModifyPort - (SCH) modify the input DBX Port
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTPort        TPort*       DBX item to be modified
//                             Valid fields to modify:
//                                  - netname
//                                  - netid
//                                  - port type
//                                  - rotate angle
//                                  - reference point
//
//  returns       long         DBX completion status
//                             and pTPort updated to reflect changes


long
DLLX TModifyPrintJob (DbxContext * pContext,      // (i/o) dbx context info
                      TPrintJob  * pPrintJob);    // (i/o) print job.
//
//  TModifyPrintJob - (PCB/SCH) modify the print job info.
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pPrintJob     TPrintJob*   DBX PrintJob to be modified
//                             Valid fields to modify:
//                                  - isSelected
//                                  - isRotated
//
//  returns       long         DBX completion status
//                             and pPrintJob updated to reflect changes


long
DLLX  TModifyRoom (DbxContext* pContext,  // (i/o) dbx context info
                   TRoom*      pTRoom);   // (i/o) room to modify
//
//  TModifyRoom - (PCB) modify the input Room
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
```

```
//  pContext      DbxContext*  Input DBX conversation data
//  pTRoom        TRoom*       DBX item to be modified
//                             Valid fields to modify:
//                                  - room name
//                                  - fill pattern
//                                  - placement side
//                                  - is fixed
//
//  returns       long         DBX completion status
//                             and pTRoom updated to reflect changes


long
DLLX  TModifyRoomAttribute  (DbxContext* pContext, // (i) dbx context info
                             long        roomId,   // (i) room id
                             TAttribute* pTAttr);  // (i) room attribute
//
// TModifyRoomAttribute - (PCB) Modify a room attribute
// --------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  roomId              long         Id of Room containing attribute to modify
//  pAttr               TAttribute*  Attribute to modify
//                             Valid fields to modify:
//                                  - value
//
//  Returns       long         DBX completion status
//


long
DLLX TModifySymbol(DbxContext* pContext,     // (i/o) dbx context info
                   TSymbol*    pTSymbol);    // (i/o) item to modify
//
//  TModifySymbol - (SCH) modify the input DBX Symbol
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTSymbol      TSymbol*     DBX item to be modified
//                             Valid fields to modify:
//                                  - refDes
//                                  - value
//                                  - refPoint (moves component location)
//
//  returns       long         DBX completion status
//                             and pTSymbol updated to reflect changes


long
DLLX TModifySymbolAttribute  (DbxContext* pContext,     // (i/o)
                              char* symbRefDes,         // (i) Symbol Ref. Des.
```

```
                                TAttribute* TAttribute);  // (i/o)
//
// TModifySymbolAttribute - (SCH) Modify a symbol attribute
// --------------------
//
//  parameter           Type/Description
//  ---------           ----------------
//
//  pContext            DbxContext*  Input DBX conversation data
//  pSymName            char*        Input string containing the
//                                   Symbol RefDes
//  pAttribute          TAttribute*  Attribute to modify
//                             Valid fields to modify:
//                                  - value
//                                  - refPoint
//                                  - textStyleId
//                                  - justPoint
//                                  - isVisible
//
//  Returns       long         DBX completion status
//


long
DLLX TModifyText (DbxContext* pContext,     // (i/o) dbx context info
                  TText*      pTText);      // (i/o) item to modify
//
//  TModifyText - (PCB/SCH) modify the input DBX Text
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTText        TText*       DBX item to be modified
//                             Valid fields to modify:
//                                  - text
//                                  - refPoint
//                                  - textStyleId
//                                  - justPoint
//                                  - isVisible
//                                  - layerId
//
//  returns       long         DBX completion status
//                             and pTText updated to reflect changes

long
DLLX TModifyVia  (DbxContext* pContext,     // (i/o) dbx context info
                  TVia*       pTVia);       // (i/o) item to modify
//
//  TModifyVia - (PCB) modify the input DBX Via
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
```

```
//  pTVia         TVia*        DBX item to be modified
//                             Valid fields to modify:
//                                  - styleId
//                                  - location
//
//  returns       long         DBX completion status
//                             and pTVia updated to reflect changes

long
DLLX TModifyWire (DbxContext* pContext,     // (i/o) dbx context info
                  TWire*      pTWire);      // (i/o) item to modify
//
//  TModifyWire - (SCH) modify the input DBX Wire
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTWire        TWire*       DBX item to be modified
//                             Valid fields to modify:
//                                  - startPt
//                                  - endPt
//                                  - isVisible
//
//  returns       long         DBX completion status
//                             and pTWire updated to reflect changes
//

//
//  Move Functions - (PCB/SCH) highlight the input DBX item
//
//
//    All functions have the following parameters
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  dx            long         Delta x to move
//  dy            long         Delta y to move
//  p<item>       <DbxItem>*   DBX item to be moved
//
//  returns       long         DBX completion status
//                             and p<item> updated to reflect highlight status
//
// Note: TMoveComponent only applies for PCB components, and
//       fails for SCH components
//

long
DLLX TMoveArc      (DbxContext* pContext,     // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
                    long        dy,           // (i) delta y to move
                    TArc*       pTArc);       // (i/o) item to move
```

```
long
DLLX TMoveAttribute(DbxContext* pContext,     // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
                    long        dy,           // (i) delta y to move
                    TAttribute* pTAttr);      // (i/o) item to move

long
DLLX TMoveBus      (DbxContext* pContext,     // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
                    long        dy,           // (i) delta y to move
                    TBus*       pTBus);       // (i/o) item to move

long
DLLX TMoveComponent(DbxContext* pContext,     // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
                    long        dy,           // (i) delta y to move
                    TComponent* pTComponent); // (i/o) item to move

long
DLLX TMoveDetail   (DbxContext* pContext,     // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
                    long        dy,           // (i) delta y to move
                    TDetail*    pTDetail);    // (i/o) item to move

long
DLLX TMoveDiagram  (DbxContext* pContext,      // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
                    long        dy,           // (i) delta y to move
                    TDiagram*   pTDiagram);   // (i/o) item to move

long
DLLX TMoveField    (DbxContext* pContext,     // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
                    long        dy,           // (i) delta y to move
                    TField*     pTField);     // (i/o) item to move

long
DLLX TMoveItem     (DbxContext* pContext,     // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
                    long        dy,           // (i) delta y to move
                    TItem*      pTItem);      // (i/o) item to move

long
DLLX TMoveLine     (DbxContext* pContext,     // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
                    long        dy,           // (i) delta y to move
                    TLine*      pTLine);      // (i/o) item to move

long
DLLX TMovePad      (DbxContext* pContext,     // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
                    long        dy,           // (i) delta y to move
                    TPad*       pTPad);       // (i/o) item to move

long
DLLX TMovePin      (DbxContext* pContext,     // (i/o) dbx context info
                    long        dx,           // (i) delta x to move
```

```
                    long        dy,           // (i) delta y to move
                    TPin*       pTPin);       // (i/o) item to move

long
DLLX TMovePoint     (DbxContext* pContext,     // (i/o) dbx context info
                     long        dx,           // (i) delta x to move
                     long        dy,           // (i) delta y to move
                     TPoint*     pTPoint);     // (i/o) item to move

long
DLLX TMovePort      (DbxContext* pContext,     // (i/o) dbx context info
                     long        dx,           // (i) delta x to move
                     long        dy,           // (i) delta y to move
                     TPort*      pTPort);      // (i/o) item to move

long
DLLX TMoveSymbol    (DbxContext* pContext,     // (i/o) dbx context info
                     long        dx,           // (i) delta x to move
                     long        dy,           // (i) delta y to move
                     TSymbol*    pTSymbol);      // (i/o) item to move

long
DLLX TMoveTable     (DbxContext* pContext,     // (i/o) dbx context info
                     long        dx,           // (i) delta x to move
                     long        dy,           // (i) delta y to move
                     TTable*     pTTable);     // (i/o) item to move

long
DLLX TMoveText      (DbxContext* pContext,     // (i/o) dbx context info
                     long        dx,           // (i) delta x to move
                     long        dy,           // (i) delta y to move
                     TText*      pTText);      // (i/o) item to move

long
DLLX TMoveVia       (DbxContext* pContext,     // (i/o) dbx context info
                     long        dx,           // (i) delta x to move
                     long        dy,           // (i) delta y to move
                     TVia*       pTVia);       // (i/o) item to move

long
DLLX TMoveWire      (DbxContext* pContext,     // (i/o) dbx context info
                     long        dx,           // (i) delta x to move
                     long        dy,           // (i) delta y to move
                     TWire*      pTWire);      // (i/o) item to move

long
DLLX TMoveMetaFile (DbxContext*     pContext,    // (i/o) dbx context info
                     long         dx,           // (i) delta x to move
                     long         dy,           // (i) delta y to move
                     TMetaFile*   pTMetaFile); // (i/o) item to move


long
DLLX TOpenDesign (long language,
                  long version,
                  const char* pDesignName,
                  DbxContext* pContext);
```

```
//
// TOpenDesign - (PCB/Sch) Establishes communication channel with
// -----------             the specified P-CAD application
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  language      long         Calling program language
//                             (constant DBX_LANGUAGE)
//  version       long         DBX program version
//                             (constant DBX_VERSION)
//  pAppName      char*        Input application name ("pcb" or "sch")
//  pContext      DbxContext*  Output DBX converstation data
//
//
//  returns       long         DBX completion status
//

long
DLLX TOpenComponent   (DbxContext*   pContext,
                       const char*   pCompType);


long
DLLX  TOpenLibrary(long language,            // (i) language used (C or VB)
                   long version,             // (i) dbx user version number
                   const char* pLibraryName, // (i) Library Name
                   DbxContext* pContext);    // (o) dbx context data, including
                                                hConv handle


long
DLLX TOutputPrintJobByName (DbxContext * pContext,  // (i/o) dbx context info
                            const char * pJobName); // (i)   print job name.
//
//  TOutputPrintJobByName - (PCB/SCH) Generate output of a print job
//                           by the job name.
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pJobName      char*        Name of PCB Print Job or SCH Sheet
//
//  returns       long         DBX completion status
//


long
DLLX TOutputSelectedPrintJobs (DbxContext * pContext);      // (i/o) dbx
                                                               context info
//
//  TOutputSelectedPrintJobs - (PCB/SCH) Generate all the print
//                              jobs selected for output.
//
//
```

```
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//
//  returns       long         DBX completion status
//


long
DLLX TPlaceArc (DbxContext* pContext,          // (i/o) dbx context info
                TArc*       pTArc);            // (i/o) item to place
//
//  TPlaceArc - (PCB) Place the input DBX arc
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTArc         TArc*        DBX item to be modified
//                             Valid fields for Place:
//                                  - width
//                                  - radius
//                                  - centerPt
//                                  - startAng
//                                  - sweepAng
//                                  - layerId
//
//  returns       long         DBX completion status
//                             and pTArc updated to reflect changes
//


long
DLLX TPlaceAttribute(DbxContext* pContext,     // (i/o) dbx context info
                     TAttribute* pTAttr);      // (i/o) item to place
//
//  TPlaceAttribute - (PCB) Place the input DBX design attribute
//                           (Use TAddCompAttribute for component attributes)
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTAttr        TAttribute*  DBX item to be modified
//                             Valid fields for Place:
//                                  - value
//                                  - refPoint
//                                  - textStyleId
//                                  - justPoint
//                                  - isVisible
//                                  - layerId
//
//  returns       long         DBX completion status
//                             and pAttr updated to reflect changes
```

```
//

long
DLLX TPlaceBus      (DbxContext* pContext,     // (i/o) dbx context info
                     TBus*       pTBus);       // (i/o) item to place
//
//  TPlaceAttribute - (SCH) Place the input DBX bus object
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTBus         TBus*        DBX item to be modified
//                             Valid fields for Place:
//                                  - Starting point
//                                  - ending point
//                                  - bus name
//                                  - layerId
//
//  returns       long         DBX completion status
//                             and pTBus updated to reflect changes
//

long
DLLX TPlaceComponent(DbxContext* pContext,     // (i/o) dbx context info
                     TComponent* pTComponent); // (i/o) item to place
//
//  TPlaceComponent - (PCB) Place the input DBX Component
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTComponent   TComponent*  DBX item to be modified
//                             Valid fields for Place:
//                                  - compType
//                                  - libraryName (Must be open. If none
//                                    specified, search the open libraries)
//                                  - refDes
//                                  - value
//                                  - refPoint (component location)
//
//  returns       long         DBX completion status
//                             and pTComponent updated to reflect changes


long
DLLX TPlaceField    (DbxContext* pContext,     // (i/o) dbx context info
                     TField* pTField);         // (i/o) item to place
//
//  TPlaceField - (SCH) Place the input DBX Field
//
//
//  Parameter     Type/Description
//  ---------     ----------------
```

```
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTField       TField*      DBX item to be placed
//                             Valid fields for Place:
//                                  - key type
//                                  - location
//                                  - style
//                                  - justification
//                                  - visibility
//                                  - layerid
//
//  returns       long         DBX completion status
//                             and pTField updated to reflect changes


long
DLLX TPlaceLine (DbxContext* pContext,     // (i/o) dbx context info
                 TLine*      pTLine);      // (i/o) item to place
//
//  TPlaceLine - (PCB) Place the input DBX Line
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTLine        TLine*       DBX item to be modified
//                             Valid fields for Place:
//                                  - width
//                                  - startPt
//                                  - endPt
//                                  - layerId
//
//  returns       long         DBX completion status
//                             and pTLine updated to reflect changes


long
DLLX TPlacePad  (DbxContext* pContext,     // (i/o) dbx context info
                 TPad*       pTPad);       // (i/o) item to place
//
//  TPlacePad - (PCB) Place the input DBX Pad
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTPad         TPad*        DBX item to be modified
//                             Valid fields for Place:
//                                  - styleId
//                                  - location
//                                  - padNum
//
//  returns       long         DBX completion status
//                             and pTPad updated to reflect changes
```

```
long
DLLX TPlacePin  (DbxContext* pContext,     // (i/o) dbx context info
                 TPin*       pTPin);       // (i/o) item to place
//
//  TPlacePin - (SCH) Place the input DBX Pin
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTPin         TPin*        DBX item to be modified
//                             Valid fields for Place:
//                                  - location
//                                  - style
//                                  - layer id
//                                  - pin number
//                                  - default pin des
//
//  returns       long         DBX completion status
//                             and pTPin updated to reflect changes



long
DLLX TPlacePoint(DbxContext* pContext,     // (i/o) dbx context info
                 TPoint*     pTPoint);     // (i/o) item to place
//
//  TPlacePoint - (PCB) Place the input DBX Point
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTPoint       TPoint*      DBX item to be modified
//                             Valid fields for Place:
//                                  - x
//                                  - y
//                                  - pointType
//                                  - location
//                                  - infoText (for Info Point)
//
//  returns       long         DBX completion status
//                             and pTPoint updated to reflect changes

long
DLLX TPlacePort (DbxContext* pContext,     // (i/o) dbx context info
                 TPort*      pTPort);      // (i/o) item to place
//
//  TPlacePort - (SCH) Place the input DBX Port
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
```

```
//  pTPort        TPort*       DBX item to be modified
//                             Valid fields for Place:
//                                  - net name
//                                  - port type
//                                  - pin count
//                                  - location
//                                  - pin length
//                                  - rotation
//
//  returns       long         DBX completion status
//                             and pTPort updated to reflect changes

long
DLLX TPlaceSymbol (DbxContext* pContext,     // (i/o) dbx context info
                   TSymbol*    pTSymbol);    // (i/o) item to place
//
//  TPlaceSymbol - (SCH) Place the input DBX Symbol
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTSymbol      TSymbol*     DBX item to be modified
//                             Valid fields for Place:
//                                  - symbol name
//                                  - refdes
//                                  - number pins
//                                  - part number
//                                  - alternate type
//                                  - ref point
//                                  - layer id
//                                  - library name
//
//  returns       long         DBX completion status
//                             and pTSymbol updated to reflect changes

long
DLLX TPlaceText (DbxContext* pContext,     // (i/o) dbx context info
                 TText*      pTText);      // (i/o) item to place
//
//  TPlaceText - (PCB) Place the input DBX Text
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTText        TText*       DBX item to be modified
//                             Valid fields for Place:
//                                  - text
//                                  - refPoint
//                                  - textStyleId
//                                  - justPoint
//                                  - isVisible
//                                  - layerId
//
```

```
//  returns       long         DBX completion status
//                             and pTText updated to reflect changes


long
DLLX TPlaceVia  (DbxContext* pContext,     // (i/o) dbx context info
                 TVia*       pTVia);       // (i/o) item to place
//
//  TPlaceVia - (PCB) Place the input DBX Via
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTVia         TVia*        DBX item to be modified
//                             Valid fields for Place:
//                                  - styleId
//                                  - location
//
//  returns       long         DBX completion status
//                             and pTVia updated to reflect changes

long
DLLX TPlaceWire (DbxContext* pContext,     // (i/o) dbx context info
                 TWire*      pTWire);      // (i/o) item to place
//
//  TPlaceVia - (SCH) Place the input DBX Wire
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pTWire        TWire*       DBX item to be modified
//                             Valid fields for Place:
//                                  - styleId
//                                  - starting point
//                                  - ending point
//
//  returns       long         DBX completion status
//                             and pTWire updated to reflect changes

//
//  Rotate Functions - (PCB/SCH) Rotate the input DBX Item
//
//    All functions have the following parameters
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  angle         long         rotation angle, in degrees * 10
//                             -3600 <= angle <= 3600
//  pTCoord       TCoord*      Point about which to rotate the item.
//                             if pTCoord values = (-1,-1) the default
```

```
//                              item origin will be used as the rotation point
//  p<item>       <DbxItem>*    DBX item, or TItem to be rotated
//
//
//  returns       long          DBX completion status
//                              and p<item> updated to reflect new orientation
//
// Note: TRotateComponent only applies for PCB components,
//       and fails for SCH components
//
long
DLLX TRotateArc    (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TArc*       pTArc);      // (i/o) item to flip

long
DLLX TRotateAttribute(DbxContext* pContext,  // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TAttribute* pTAttr);     // (i/o) item to flip

long
DLLX TRotateBus    (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TBus*       pTBus);      // (i/o) item to flip

long
DLLX TRotateComponent(DbxContext* pContext,  // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TComponent* pTComponent);// (i/o) item to flip

long
DLLX TRotateField  (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TField*     pTField);    // (i/o) item to flip

long
DLLX TRotateItem   (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TItem*      pTItem);     // (i/o) item to flip

long
DLLX TRotateLine   (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TLine*      pTLine);     // (i/o) item to flip

long
DLLX TRotatePad    (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TPad*       pTPad);      // (i/o) item to flip
```

```
long
DLLX TRotatePin    (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TPin*       pTPin);      // (i/o) item to flip

long
DLLX TRotatePort   (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TPort*      pTPort);     // (i/o) item to flip

long
DLLX TRotateSymbol (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TSymbol*    pTSymbol);   // (i/o) item to flip

long
DLLX TRotateTable  (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TTable*     pTTable);    // (i/o) item to flip

long
DLLX TRotateText   (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TText*      pTText);     // (i/o) item to flip

long
DLLX TRotateVia    (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TVia*       pTVia);      // (i/o) item to flip

long
DLLX TRotateWire   (DbxContext* pContext,    // (i/o) dbx context info
                    long        angle,       // (i) rotation angle
                    TCoord*     pTCoord,     // (i) rotation point
                    TWire*      pTWire);     // (i/o) item to flip


long
DLLX TSaveComponent   (DbxContext*   pContext);


long
DLLX TSaveDesign (DbxContext * pContext);          // (i/o) dbx context info
//
//  TSaveDesign - (PCB/SCH) Saves the current design to file.
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
```

```
//  pContext      DbxContext*  Input DBX conversation data
//
//  returns       long         DBX completion status
//


long
DLLX TSelectAllPrintJobs (DbxContext * pContext);   // (i/o) dbx context info
//
//  TSelectAllPrintJobs - (PCB/SCH) Selects all the print jobs for output.
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//
//  returns       long         DBX completion status
//


long
DLLX TSelectPrintJob (DbxContext * pContext,        // (i/o) dbx context info
                      const char * pJobName);       // (i)   print job name.
//
//  TSelectPrintJob - (PCB/SCH) Select the named print job for output.
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  pJobName      char*        Name of PCB Print Job or SCH Sheet
//
//  returns       long         DBX completion status
//


//
//  UnHighlight Functions - (PCB/SCH) Unhighlight the input DBX item
//
//
//    All functions have the following parameters
//
//
//  Parameter     Type/Description
//  ---------     ----------------
//
//  pContext      DbxContext*  Input DBX conversation data
//  p<item>       <DbxItem>*   DBX item to be unhighlighted
//
//  returns       long         DBX completion status
//                             and p<item> updated to reflect highlight status
//
// Note: TUnHighlightComponent only applies for PCB components,
//       and fails for SCH components
//
```

```
long
DLLX TUnHighlightAll(DbxContext* pContext);     // (i/o) dbx context info

long
DLLX TUnHighlightArc (DbxContext* pContext,   // (i/o) dbx context info
                      TArc*       pTArc);     // (i/o) item to unhighlight

long
DLLX TUnHighlightAttribute (DbxContext* pContext, // (i/o) dbx context info
                            TAttribute* pTAttr);  // (i/o) item to unhighlight

long
DLLX TUnHighlightBus       (DbxContext* pContext, // (i/o) dbx context info
                            TBus*       pTBus);   // (i/o) item to unhighlight

long
DLLX TUnHighlightComponent(DbxContext* pContext,        // (i/o) dbx context
                                                    info
                           TComponent* pTComponent);    // (i/o) item to
                                                         unhighlight

long
DLLX TUnHighlightDetail (DbxContext* pContext,      // (i/o) dbx context info
                         TDetail*    pTDetail);     // (i/o) item to
                                                         unhighlight

long
DLLX TUnHighlightDiagram (DbxContext* pContext,      // (i/o) dbx context info
                          TDiagram*   pTDiagram);    // (i/o) item to
                                                         unhighlight

long
DLLX TUnHighlightField (DbxContext* pContext,   // (i/o) dbx context info
                        TField*     pTField);   // (i/o) item to unhighlight

long
DLLX TUnHighlightItem(DbxContext* pContext,     // (i/o) dbx context info
                      TItem*      pTItem);      // (i/o) item to unhighlight

long
DLLX TUnHighlightLine(DbxContext* pContext,     // (i/o) dbx context info
                      TLine*      pTLine);      // (i/o) item to unhighlight

long
DLLX TUnHighlightMetaFile (DbxContext* pContext,      // (i/o) dbx context info
                           TMetaFile*  pTMetaFile);   // (i/o) item to
                                                         unhighlight

long
DLLX TUnHighlightNet (DbxContext* pContext,     // (i/o) dbx context info
                      TNet*       pTNet);       // (i/o) net to unhighlight

long
DLLX TUnHighlightPad (DbxContext* pContext,     // (i/o) dbx context info
                      TPad*       pTPad);       // (i/o) item to unhighlight
```

```
long
DLLX TUnHighlightPin (DbxContext* pContext,     // (i/o) dbx context info
                      TPin*       pTPin);       // (i/o) item to unhighlight

long
DLLX TUnHighlightPoint(DbxContext* pContext,    // (i/o) dbx context info
                       TPoint*     pTPoint);    // (i/o) item to unhighlight

long
DLLX TUnHighlightPort (DbxContext* pContext,    // (i/o) dbx context info
                       TPort*      pTPort);     // (i/o) item to unhighlight

long
DLLX TUnHighlightRoom (DbxContext* pContext,    // (i/o) dbx context info
                       TRoom*      pTRoom);     // (i/o) object to unhighlight

long
DLLX TUnHighlightSymbol (DbxContext* pContext,  // (i/o) dbx context info
                         TSymbol*    pTSymbol); // (i/o) item to unhighlight

long
DLLX TUnHighlightTable(DbxContext* pContext,    // (i/o) dbx context info
                       TTable*     pTTable);    // (i/o) item to unhighlight

long
DLLX TUnHighlightText(DbxContext* pContext,     // (i/o) dbx context info
                      TText*      pTText);      // (i/o) item to unhighlight

long
DLLX TUnHighlightVia (DbxContext* pContext,     // (i/o) dbx context info
                      TVia*       pTVia);       // (i/o) item to unhighlight

long
DLLX TUnHighlightWire(DbxContext* pContext,     // (i/o) dbx context info
                      TWire*      pTWire);      // (i/o) item to unhighlight
```