3 UPI Functions Reference

This section provides the reference to the L-Edit User-Programmable Interface (UPI) and its C-language functions and datatypes. Functions are arranged in three primary categories—interface functions, database functions, and typedefs.

Interface Functions (page 4-57) allow you to create dialog boxes and other interface elements and to register UPI macros.

Database Functions (page 4-104) allow you to create and manipulate a design database. Subcategories of database functions include:

- **File Functions** (page 4-105)
- **Cell Functions** (page 4-154)
- **Instance Functions** (page 4-191)
- **Entity Functions** (page 4-215)
- **Object Functions** (page 4-232)

Typedefs (page 4-509) allow you to create and manipulate data structures.

Interface Functions

The first group of functions listed below are used to create graphical interface elements such as dialog and message boxes.

LDialog_MsgBox (page 4-60)	**LDialog_InputBox** (page 4-66)
LDialog_MultiLineMsgBox (page 4-61)	**LDialog_MultiLineInputBox** (page 4-68)
LDialog_AlertBox (page 4-63)	**LDialog_PickList** (page 4-71)
LDialog_YesNoBox (page 4-64)	

Functions in the second group are used for tasks such as finding the current mouse position, displaying a message in the status bar, or getting and setting the visible cell in L-Edit's layout window.

LCursor_GetPosition (page 4-74)	**LCell_HomeView** (page 4-80)
LCursor_GetSnappedPosition (page 4-75)	**LCell_GetVisible** (page 4-81)
LCursor_GetPositionEx99 (page 4-76)	**LCell_GetLastVisible** (page 4-82)

LDisplay_Refresh (page 4-78)	**LCell_MakeVisible** (page 4-83)
LStatusBar_SetMsg (page 4-79)	**LCell_MakeVisibleNoRefresh** (page 4-84)

Functions in the third group are used for registering and binding UPI macros and getting window parameters.

LMacro_Register (page 4-85)	**LMacro_BindToMenu** (page 4-87)
LMacro_BindToHotKey (page 4-86)	**LWindow_GetParameters** (page 4-88)

Functions in the fourth group are used for finding the L-Edit serial number, choosing the selection tool, and inserting a menu item separator.

LUpi_GetSerialNumber (page 4-90)	**LUpi_SetSelectionTool** (page 4-93)
LUpi_SetQuietMode (page 4-91)	**LUpi_SetDrawingTool** (page 4-94)
LUpi_InQuietMode (page 4-92)	**LUpi_InsertMenuItemSeparator** (page 4-95)

Functions in the fifth group are used to manipulate multiple windows in L-Edit:

- **LWindow_GetVisible** (page 4-96)
- **LWindow_MakeVisible** (page 4-97)
- **LWindow_Close** (page 4-98)
- **LWindow_CloseAll** (page 4-99)
- **LWindow_Get Type** (page 4-100)
- **LWindow_Get Cell** (page 4-101)
- **LFile_DisplayCellBrowser** (page 4-102)
- **LFile_OpenCell** (page 4-103)

LDialog_MsgBox

```
void LDialog_MsgBox(char *msg);
```

Description

Produces a single-line message box.

Parameters

msg	Specifies the message to be displayed.

Example

```
LDialog_MsgBox("Hello World");
```

See Also

Interface Functions (page 4-57)

LDialog_MultiLineMsgBox

```
void LDialog_MultiLineMsgBox(char *messages[], int
    total_entries);
```

Description

Produces a multiple-line message dialog.

Parameters

messages	Multiple-line message displayed in the dialog.
total_entries	Number of message lines.

Example

```
/*This example displays a message box with two message
   lines*/

/*Declare a message buffer to hold both the messages*/
char *msg_buf[2] = {
          “Message 1”,
          “Message 2”
};

/*Display the multi line message box*/
LDialog_MultiLineMsgBox(msg_buf, 2);
```

See Also

Interface Functions (page 4-57)

LDialog_AlertBox

`void LDialog_AlertBox(char *`*`msg`*`);`

Description

Produces a warning dialog.

Parameters

msg	Warning displayed in the dialog.

Example

```
LDialog_AlertBox("An Error has Occured");
```

See Also

Interface Functions (page 4-57)

LDialog_YesNoBox

```
int LDialog_YesNoBox(char *msg);
```

Description

Produces a query dialog. One of two choices is clicked in response to the query.

Return Values

If **Yes** is clicked, the function returns 1; if **No** is clicked, it returns zero.

Parameters

msg	Query to be displayed in the dialog.

Example

```
if ( LDialog_YesNoBox("Do you want to Continue") ){
     /*Yes is clicked - the program continues*/
}
else {
     /*No is clicked - the program exits*/
}
```

See Also

Interface Functions (page 4-57)

LDialog_InputBox

int LDialog_InputBox(char ****title*, char ****msg***, char ****ibuf***);**

Description

Produces an input dialog. The value entered by the user is returned as a string. If another datatype is needed, the string must be converted to the appropriate type.

Return Values

If **OK** is clicked, the function returns 1; if **Cancel** is clicked, it returns zero.

Parameters

title	Title of the dialog box.
msg	Prompt displayed in the dialog.
ibuf	A buffer used to return the value entered at the prompt.

Example

```
/*Allocate a buffer to store the return value*/
char value_buffer[50];

/*Initialize buffer to display a default value*/
strcpy(value_buffer, “pcell”);

/*Display an input box with Cell Name Query as the title*/
if ( LDialog_InputBox(“Cell Name Query”, “Enter Name of the
   cell to be instanced”, value_buffer) == 0)
     return;
```

See Also

Interface Functions (page 4-57)

LDialog_MultiLineInputBox

```
int LDialog_MultiLineInputBox(char *title, LDialogItem
    ibuf[], int total_entries);
```

Description

Produces a multiple-line input dialog. Several values are entered in response to prompts.

Return Values

If **OK** is clicked, the function returns 1; if **Cancel** is clicked, then it returns zero.

Parameters

title	Title of the dialog box.
ibuf	Prompts displayed and the values entered.
total_entries	Number of values expected.

Example

```
/*Declare an array of dialog items to be displayed*/
LDialogItem  Dialog_Items [ 3 ] =
     { { "Inner Radius", "175" },
          { "Outer Radius", "200" },
          { "Teeth Count ", "15 " } };

long R_Inner, R_Outer, Teeth_Count;
float Teeth_Width;

/*Display a multi line dialog box to display the default
   values of gear properties and get the modified
   properties*/
if ( LDialog_MultiLineInputBox ( “Gear Properties”,
   Dialog_Items, 3 ) ){
     /*A OK was hit by the user, so get the property value
   from the Dialog_Items buffer*/
```

```
        /*Get the value of Inner Radius*/
        R_Inner = atol ( Dialog_Items [ 0 ].value );
        /*Get the value of Outer Radius*/
        R_Outer = atol ( Dialog_Items [ 1 ].value );
        /*Get the value of Teeth Count*/
        Teeth_Count = atol ( Dialog_Items [ 2 ].value );
        /*Calculate Teeth Width*/
        Teeth_Width = 6.283185307 * R_Inner / ( 2 * Teeth_Count
      );
}
```

See Also

LDialogItem (page 4-514), **Interface Functions** (page 4-57)

LDialog_PickList

```
int LDialog_PickList(char *title, char *list[], int
    total_entries, int default_choice);
```

Description

Produces an input dialog. One of a list of possibilities is chosen either by highlighting the desired item and clicking **OK**, or by double-clicking the desired item.

Return Values

If **OK** is clicked (or the highlighted item is double-clicked), the function returns the index of the highlighted item; if **Cancel** is clicked, it returns –1.

Parameters

title	Title of the dialog.
list	Items listed.
total_entries	Number of items listed.
default_choice	Index of the item shown highlighted when the dialog first appears.

Example

```
/*This example displays a pick list with three members to
   choose from*/

/*Declare a buffer to hold all elements of the pick list*/
char *Pick_List [ ] =
     {"Inverter",
     "Op-Amp",
     "Transistor"};

/*Number of elements in the pick list*/
int Pick_Count = 3;

/*Index of the item picked by user*/
int Picked;
```

```
/*Display the pick list with Inverter as the default
   selection*/
Picked = LUPI_PickList (“Element Selection”, Pick_List,
   Pick_Count, 0);
```

See Also

Interface Functions (page 4-57)

LCursor_GetPosition

```
LPoint LCursor_GetPosition(void);
```

Description

Gets the current cursor (mouse) position.

Return Value

Returns the current cursor (mouse) position.

See Also

LPoint (page 4-527), **Interface Functions** (page 4-57)

LCursor_GetSnappedPosition

```
LPoint LCursor_GetSnappedPosition(void)
```

Description

Gets the current snapped cursor (mouse) position.

Return Values

Returns the current snapped cursor (mouse) position as a LPoint.

See Also

LPoint (page 4-527), **Interface Functions** (page 4-57)

LCursor_GetPositionEx99

```
LPoint LCursor_GetPositionEx99(int iSnapped, int
    iPauseForInput, const char* szMessage)
```

Description

Gets the current cursor (mouse) position. Optionally the cursor position can be snapped to the current snap grid settings in **Setup > Design—Grid**. **Upi_Lcursor_GetPositionEx99** gets the current cursor position and immediately returns. One can optionally pause for user input, allowing the user to press the left mouse button to indicate the cursor position.

Return Values

Returns the current cursor (mouse) position.

Parameters

iSnapped	Snap the cursor position to the current snap settings. (1 - True, 0 False)
iPauseForInput	Pause so the user can press the mouse left button to indicate the cursor position. (1 - True, 0 False)
szMessage	Displays the message when pausing for user input. If szMessage is NULL, then it displays "Please pick a point.".

See Also

Interface Functions (page 4-57)

LDisplay_Refresh

```
void LDisplay_Refresh(void);
```

Description

Updates the display to show layout modifications produced by UPI calls.

See Also

Interface Functions (page 4-57)

LStatusBar_SetMsg

```
void LStatusBar_SetMsg(char *msg);
```

Description

Displays a message in the status bar. To clear the status bar, set ***msg*** to “” (an empty string).

Parameters

msg	Message to be displayed.

See Also

Interface Functions (page 4-57)

LCell_HomeView

```
LStatus LCell_HomeView(LCell cell);
```

Description

Displays the home view of a given cell.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Specified cell.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Interface Functions** (page 4-57)

LCell_GetVisible

```
LCell LCell_GetVisible(void);
```

Description

Gets the visible (active) cell in the layout window.

Return Values

Returns a pointer to the active cell; otherwise NULL.

See Also

LCell (page 4-521), **Interface Functions** (page 4-57)

LCell_GetLastVisible

```
LCell LCell_GetLastVisible(LFile file);
```

Description

Gets the cell last open in the specified file.

Return Values

Returns a pointer to the last open cell in the specified file, or NULL on error.

Parameters

file	Specified file.

See Also

LCell (page 4-521), **Interface Functions** (page 4-57)

LCell_MakeVisible

```
LStatus LCell_MakeVisible(LCell cell);
```

Description

Makes the specified cell the current one and updates the display.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Specified cell.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Interface Functions** (page 4-57)

LCell_MakeVisibleNoRefresh

```
LStatus LCell_MakeVisibleNoRefresh(LCell cell);
```

Description

Makes the specified cell the current one without updating the display.

Return Values

It an error occurs, it returns LBadCell; otherwise returns LStatusOK.

Parameters

cell	Specified cell.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Interface Functions** (page 4-57)

LMacro_Register

```
void LMacro_Register(char *macro_desc, char *function);
```

Description

Registers a user-defined macro in L-Edit.

Parameters

macro_desc	Macro name that should be displayed in the **Macros list.**
function	Name of the macro function.

See Also

Interface Functions (page 4-57), **Creating an Interpreted Macro** (page 4-23)

LMacro_BindToHotKey

```
void LMacro_BindToHotKey(int keycode, char *macro_desc, char
    *function);
```

Description

Establishes a relationship between a user-defined macro and a keyboard shortcut ("hot key") so that user can invoke the macro by pressing the hot key. Supported key combinations (keycodes) are defined in **<*install_dir*>\include\lupi_usr.h**.

Parameters

keycode	Keyboard shortcut (for example, **KEY_F2** for the **F2** key).
macro_desc	String displayed in the **Macros** list of the **Run Macro** dialog.
function	Macro function name.

See Also

Interface Functions (page 4-57), **Binding Macros to Hot Keys** (page 4-36)

LMacro_BindToMenu

```
void LMacro_BindToMenu(char *menu, char *macro_desc, char
    *function);
```

Description

Establishes a relationship between a user-defined macro and a menu command.

Parameters

menu	Menu title (for example, **Tools).**
macro_desc	String displayed in the **Macros** list of the **Run Macro** dialog**.**
function	Macro function name.

See Also

Interface Functions (page 4-57), **Binding Macros to Menu Items** (page 4-38)

LWindow_GetParameters

```
void LWindow_GetParameters (void **phAppInst, void
    **phParentWnd, void **phUserDll)
```

Description

Gets the parameters of the L-Edit application window. These parameters can be used with Windows API. This call is only supported for compiled DLL macros.

In the example below, the UPI_Entry_Point function gets the L-Edit application window parameters. These parameters are used by MainFunction to interface with Windows API.

Parameters

HINSTANCE *phAppInst	Pointer to application instance.
HWND *phParentWnd	Pointer to parent window handle.
HINSTANCE *phUserDll	Pointer to DLL handle.

Example

```
HINSTANCE hInst=NULL;
HWND hWnd=NULL;
HINSTANCE hLib=NULL;
LWindow_GetParameters( (void**)&hInst, (void**)&hWnd,
    (void**)&hLib);
```

See Also

Interface Functions (page 4-57)

LUpi_GetSerialNumber

```
long LUpi_GetSerialNumber(void);
```

Description

Gets the serial number of L-Edit.

Return Values

Returns the serial number or -1 on error.

See Also

Interface Functions (page 4-57), **Creating an Interpreted Macro** (page 4-23)

LUpi_SetQuietMode

```
LStatus LUpi_SetQuietMode(int val);
```

Description

Sets the quiet mode. When the quiet mode is on, the alert boxes are suppressed. The use of quiet mode is required for batch processing.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

Interface Functions (page 4-57)

LUpi_InQuietMode

```
int LUpi_InQuietMode(void);
```

Description

Gets the quiet mode. When the quiet mode is on, the alert boxes are suppressed. The use of quiet mode is required for batch processing.

Return Values

1 if quiet mode is on, 0 if quiet mode if off

See Also

Interface Functions (page 4-57)

LUpi_SetSelectionTool

```
void LUpi_SetSelectionTool(void);
```

Description

Selects the selection tool in L-Edit.

See Also

Interface Functions (page 4-57), **Creating an Interpreted Macro** (page 4-23)

LUpi_SetDrawingTool

```
UPIDrawingToolType LUpi_SetDrawingTool(UPIDrawingToolType
    eTool)
```

Description

Selects the specified drawing tool.

Return Values

Returns the previously selected drawing tool.

Parameters

eTool	The drawing tool to select.

See Also

Interface Functions (page 4-57)

LUpi_InsertMenuItemSeparator

```
void LUpi_InsertMenuItemSeparator(char *menu);
```

Description

Appends a separator in the specified L-Edit menu. This function can be used for separating menu items.

Parameters

menu	Name of menu where separator is to be inserted.

See Also

Interface Functions (page 4-57), **Creating an Interpreted Macro** (page 4-23)

LWindow_GetVisible

```
LWindow LWindow_GetVisible(void);
```

Description

Gets the active window.

Return Values

Returns a pointer to the active window; otherwise NULL.

See Also

LWindow (page 4-578), **Interface Functions** (page 4-57)

LWindow_MakeVisible

```
LStatus LWindow_MakeVisible(LWindow wnd);
```

Description

Sets the active window.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

wnd	Pointer to a window.

See Also

LStatus (page 4-512), **LWindow** (page 4-578), **Interface Functions** (page 4-57)

LWindow_Close

```
LStatus LWindow_Close(LWindow wnd);
```

Description

Closes the specified window.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

wnd	Window to be closed.

See Also

LStatus (page 4-512), **LWindow** (page 4-578), **LWindow_CloseAll** (page 4-99), **Interface Functions** (page 4-57)

LWindow_CloseAll

```
LStatus LWindow_CloseAll(void);
```

Description

Closes all open windows.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **LWindow_Close** (page 4-98), **Interface Functions** (page 4-57)

LWindow_Get Type

```
LWindowType LWindow_GetType(LWindow wnd);
```

Description

Gets the window type.

Return Values

Returns the type of the specified window.

Parameters

wnd	Window to be closed.

See Also

LWindowType (page 4-579), **LWindow** (page 4-578), **Interface Functions** (page 4-57)

LWindow_Get Cell

```
LCell LWindow_GetCell(LWindow wnd);
```

Description

Gets the cell open in the specified layout window.

Return Values

Returns a pointer to the cell open in specified layout window; returns NULL on error.

Parameters

wnd	Window to be closed.

See Also

LCell (page 4-521), **LWindow** (page 4-578), **Interface Functions** (page 4-57)

LFile_DisplayCellBrowser

```
void LFile_DisplayCellBrowser(LFile file);
```

Description

Displays the cell browser for the specified file.

Parameters

file	Specified file.

See Also

LFile (page 4-515), **Interface Functions** (page 4-57)

LFile_OpenCell

```
LWindow LFile_OpenCell(LFile file, char *cell_name);
```

Description

Opens a layout window for the specified cell in the specified file.

Return Values

Returns a pointer to the newly created window; otherwise NULL.

Parameters

file	Specified file.
cell_name	Name of the specified cell.

See Also

LWindow (page 4-578), **LFile** (page 4-515), **Interface Functions** (page 4-57)

Database Functions

TDB (Tanner Database) files are design files in a Tanner Research proprietary format. A TDB file is the highest level of the L-Edit database hierarchy. It is composed of linearly linked lists of cells and layers.

A TDB file is the highest level of the L-Edit database hierarchy. A single TDB file usually contains the complete design for a chip or MCM, but it may also consist of a library of cells or a partial design to be merged with other design files.

File Functions

The file functions below allow the user to manipulate an L-Edit design file.

(continued)

LFile_GetSetupVersion (page 4-133)

LFile_SetSetupVersion (page 4-134)

LFile_GetInfoText (page 4-136)

LFile_SetInfoText (page 4-138)

LFile_GetEnvironment (page 4-139)

LFile_SetEnvironment (page 4-140)

LFile_GetGrid (page 4-141)

LFile_SetGrid (page 4-142)

LFile_GetSelectionParam (page 4-143)

LFile_SetSelectionParam (page 4-144)

LFile_GetUserData (page 4-146)

LFile_SetUserData (page 4-147)

LFile_DeleteUserData (page 4-148)

LFile_SetLastCurrent (page 4-149)

LFile_GetDesignRuleFlags (page 4-150)

LFile_SetDesignRuleFlags (page 4-152)

Some subcategories of file functions include:

Cell Functions (page 4-154)

Layer Functions (page 4-351)

Technology Setup Functions (page 4-418)

Import/Export Functions (page 4-441)

LFile_New

```
LFile LFile_New(LFile setup_file, char* name);
```

Description

Creates a new, empty layout file with a technology setup copied from the specified file.

Return Values

Returns a pointer to the new file, or NULL on error.

Parameters

setup_file	File whose setup is to be used (if NULL, then the setup of the current file is used).
name	Name of the new file.

See Also

File Functions (page 4-105)

LFile_Open

```
LFile LFile_Open(const char* name, LFileType type);
```

Description

Opens a TDB, CIF, or GDS II file.

Return Values

A pointer to the file, or NULL on error.

Parameters

name	Name of the file to open.
type	Format of the file (LTdbFile, LCifFile, or LGdsFile).

See Also

LFile (page 4-515), **LFileType** (page 4-516), **File Functions** (page 4-105)

LFile_Save

```
LStatus LFile_Save(LFile file);
```

Description

Saves the specified file into a TDB file of the same name (with extension .tdb).

Return Values

LStatusOK if successful. If an error occurs LStatus contains the error value.

Parameters

file	Pointer to the file to be saved.

See Also

LStatus (page 4-512), **LFile_SaveAs** (page 4-110), **File Functions** (page 4-105)

LFile_SaveAs

```
LStatus LFile_SaveAs(LFile file, const char* name, LFileType
    type);
```

Description

Saves a file as a different file with the specified name and file type.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	File to be saved.
name	Name under which the file is to be saved.
type	Format in which the file is to be saved (LTdbFile, LCifFile, or LGdsFile).

See Also

LStatus (page 4-512), **LFileType** (page 4-516), **LFile_Save** (page 4-109), **File Functions** (page 4-105)

LFile_Close

```
LStatus LFile_Close(LFile file);
```

Description

Closes the specified file without checking for changes.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	File to be closed.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **File Functions** (page 4-105)

LFile_Find

```
LFile LFile_Find(const char* name);
```

Description

Finds a file in a list of open files whose name matches the specified string.

Return Values

Returns a pointer to the file, if found; otherwise returns NULL.

Parameters

name	Name (without filename extension) of the file to be searched.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_GetList

```
LFile LFile_GetList(void)
```

Description

Gets a list of currently open files.

Return Values

Returns the head of the list of open files. Returns NULL if no files are open.

See Also

LFile_GetNext (page 4-115), **LCell_GetList** (page 4-162), **LInstance_GetList** (page 4-203), **File Functions** (page 4-105)

LFile_GetNext

```
LFile LFile_GetNext(LFile file);
```

Description

Gets the next file in the list of open files after the specified file.

Return Values

Returns a pointer to the next file in the currrently opened file list. If no next file exists, it returns a NULL.

Parameters

file	Pointer to a file.

Example

```
/*This example demonstrates a simple way of traversing all
   the loaded files*/
```

```
/*Declare a L-Edit file variable*/
LFile file;

/*Get a list of all the currently loaded files and traverse
   the list*/
for(file = LFile_GetList(); file != NULL; file =
   LFile_GetNext() {
     /*Do processing specific to a file*/
}
```

See Also

LFile_GetList (page 4-114), **LCell_GetNext** (page 4-163), **LInstance_GetNext** (page 4-204), **File Functions** (page 4-105)

LFile_GetLock

```
int LFile_GetLock(LFile file);
```

Description

Checks whether a file is locked or not.

Return Values

Returns zero if the specified file is unlocked; otherwise returns a nonzero value.

Parameters

file	File to be checked.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_SetLock

```
int LFile_SetLock(LFile file, int set);
```

Description

Locks or unlocks the specified file. If **set** is nonzero, the file is locked; if **set** is zero, the file is unlocked.

Return Values

A nonzero value if the file is locked.

Parameters

file	File to be locked or unlocked.
set	Value that determines the file's new status: zero unlocks; anything else locks.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_IsChanged

```
int LFile_IsChanged(LFile file);
```

Description

Checks the specified file to determine if it has been changed since it was last saved.

Return Values

The function returns 1 if the file has been changed or zero if it has not.

Parameters

file	File to be checked.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_GetName

```
char* LFile_GetName(LFile file, char* name, const int
    maxlen);
```

Description

Gets the text of the name field in the file information summary.

Return Values

Returns a pointer to the string ***name***; returns NULL if unsuccessful.

Parameters

file	File whose name is to be retrieved.
name	String containing the name text (the name buffer).
maxlen	Maximum length allowed for ***name.***

See Also

LCell_GetName (page 4-166), **LInstance_GetName** (page 4-205), **File Functions** (page 4-105)

LFile_GetAuthor

```
char* LFile_GetAuthor(LFile file, char* author, const int
    maxlen);
```

Description

Gets the text of the ***author*** field in the information summary for the specified file.

Return Values

Returns a pointer to the string ***author*** if successful; otherwise returns NULL.

Parameters

file	File whose author is to be retrieved.
author	String containing the author text.
maxlen	Maximum length allowed for ***author***.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_SetAuthor

```
char* LFile_SetAuthor(LFile file, char* author);
```

Description

Sets the text of the author field in the information summary for the specified file.

Return Values

Returns a pointer to the string ***author*** if successful; otherwise NULL.

Parameters

file	File whose ***author*** text is to be set.
author	String containing the ***author*** text.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_GetFabricationCell

```
LCell LFile_GetFabricationCell(LFile file);
```

Description

Gets the cell marked as the “top” or “root” cell (the fabrication cell) of the specified file for foundry fabrication.

Return Values

Returns a pointer to the fabrication cell if found; otherwise NULL.

Parameter

file	Specified file.

See Also

LCell (page 4-521), **LFile** (page 4-515), **File Functions** (page 4-105)

LFile_SetFabricationCell

```
LStatus LFile_SetFabricationCell(LFile file, LCell cell);
```

Description

Marks the specified cell as the “top” or “root” cell (the fabrication cell) of the specified file for foundry fabrication, conforming to CIF and GDS II conventions.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
cell	Cell to be set as the fabrication cell.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LFile** (page 4-515), **File Functions** (page 4-105)

LFile_GetOrganization

```
char* LFile_GetOrganization(LFile file, char* org, const int
    maxlen);
```

Description

Gets the text of the organization field in the information summary for the specified file.

Return Values

Returns a pointer to the organization string if successful; otherwise returns NULL.

Parameters

file	File whose organization is to be retrieved.
org	String containing the organization text.
maxlen	Maximum length allowed for ***org.***

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_SetOrganization

```
char* LFile_SetOrganization(LFile file, char* org);
```

Description

Sets the text of the organization field in the information summary for the specified file.

Return Values

Returns a pointer to the file organization buffer if successful; otherwise returns NULL.

Parameters

file	File whose organization is to be set.
org	String containing the organization text.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_GetLayoutVersion

```
void LFile_GetLayoutVersion(LFile file, long* major, long*
    minor);
```

Description

Gets the major and minor layout version numbers of the specified file.

Parameters

file	File whose layout version numbers are to be retrieved.
major	Pointer to the major layout version number.
minor	Pointer to the minor layout version number.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_SetLayoutVersion

```
void LFile_SetLayoutVersion(LFile file, long* major, long*
    minor);
```

Description

Sets the major and minor layout version numbers of the specified file.

Parameters

file	file whose layout version numbers are to be set.
major	Pointer to the major layout version number.
minor	Pointer to the minor layout version number.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_GetSetupVersion

```
void LFile_GetSetupVersion(LFile file, long* major, long*
    minor);
```

Description

Gets the major and minor setup numbers of the specified file.

Parameters

file	File whose setup version numbers are to be retrieved.
major	Pointer to the major setup version number.
minor	Pointer to the minor setup version number.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_SetSetupVersion

```
void LFile_SetSetupVersion(LFile file, long* major, long*
    minor);
```

Description

Sets the major and minor setup version numbers of the specified file.

Return Values

Returns NULL on error.

Parameters

file	File whose setup version numbers are to be set.
major	Pointer to the major setup version number.
minor	Pointer to the minor setup version number.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_GetInfoText

```
char* LFile_GetInfoText(LFile file, char* info, const int
    maxlen);
```

Description

Gets the text of the information field in the file information summary for the specified file.

Return Values

Returns a pointer to the string **info** if successful; otherwise returns NULL.

Parameters

file	File whose information is to be retrieved.
info	String containing the information text.
maxlen	Maximum length allowed for ***info.***

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_SetInfoText

```
char* LFile_SetInfoText(LFile file, char* info);
```

Description

Sets and returns the text of the information field in the file information summary for the specified file. A NULL value may be given.

Return Values

Returns a pointer to the string ***info*** if successful; otherwise returns NULL.

Parameters

file	File whose information is to be set.
info	String containing the information text.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_GetEnvironment

```
LEnvironment *LFile_GetEnvironment(LFile file, LEnvironment
    *env);
```

Description

Gets the environment setting of the specified file.

Return Values

Returns a pointer to the string ***env*** if successful; otherwise returns NULL.

Parameters

file	File whose environment setting is to be retrieved.
env	Pointer to the file environment structure.

See Also

LEnvironment (page 4-517), **LFile** (page 4-515), **File Functions** (page 4-105)

LFile_SetEnvironment

```
LStatus LFile_SetEnvironment(LFile file, LEnvironment *env);
```

Description

Sets the environment of the specified file according to the parameters defined in LEnvironment.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	File whose environment is to be set.
env	Pointer to the file environment structure.

See Also

LEnvironment (page 4-517), **LFile** (page 4-515), **File Functions** (page 4-105)

LFile_GetGrid

```
LGrid *LFile_GetGrid(LFile file, LGrid *grid);
```

Description

Gets the grid setting of the specified file.

Return Values

Returns a pointer to the grid structure if successful; otherwise returns NULL.

Parameters

file	File whose grid setting is to be retrieved.
grid	Pointer to the grid structure.

See Also

LGrid (page 4-520), **LFile** (page 4-515), **File Functions** (page 4-105)

LFile_SetGrid

```
LStatus LFile_SetGrid(LFile file, LGrid *grid);
```

Description

Sets the grid of the specified file.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	File whose grid is to be set.
grid	Pointer to the grid structure.

See Also

LStatus (page 4-512), **LGrid** (page 4-520), **LFile** (page 4-515), **File Functions** (page 4-105)

LFile_GetSelectionParam

```
LSelectionParam *LFile_GetSelectionParam(LFile file,
    LSelectionParam *param);
```

Description

Gets the selection parameters of the specified file.

Return Values

Returns a pointer to the selection structure if successful; otherwise returns NULL.

Parameters

file	File whose selection parameter are to be found.
param	Pointer to the selection parameter structure.

See Also

LSelectionParam (page 4-541), **File Functions** (page 4-105)

LFile_SetSelectionParam

```
LStatus LFile_SetSelectionParam(LFile file, LSelectionParam
    *param);
```

Description

Sets the selection parameters of the given file.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	File whose selection parameters are to be set.
param	Pointer to the selection parameter structure.

See Also

LStatus (page 4-512), **LSelectionParam** (page 4-541), **File Functions** (page 4-105)

LFile_GetUserData

```
void* LFile_GetUserData(LFile file);
```

Description

Gets a pointer to user-defined data associated with the specified cell.

Return Values

Returns a pointer to the user data if successful; otherwise returns NULL.

Parameter

file	File whose user-defined data is needed.

See Also

LFile (page 4-515), **File Functions** (page 4-105)

LFile_SetUserData

```
LStatus LFile_SetUserData(LFile file, void* dataPointer);
```

Description

Uses a data pointer within a file to associate user-defined data with the file. Data can be integer, string, or any other type.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameter

file	File which will contain the user-defined data.
dataPointer	User-defined data.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **File Functions** (page 4-105)

LFile_DeleteUserData

```
LStatus LFile_DeleteUserData(LFile file);
```

Description

Deletes the user-defined expansion pointer in the specified file.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **File Functions** (page 4-105)

LFile_SetLastCurrent

```
LStatus LFile_SetLastCurrent(LFile file, LCell cell);
```

Description

Sets the last open cell in the specified file.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
cell	Specified cell.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **LCell** (page 4-521), **File Functions** (page 4-105)

LFile_GetDesignRuleFlags

```
LStatus LEDITAPI LFile_GetDesignRuleFlags(LFile file,
   LDesignRuleFlags *pDRCFlags);
```

Description

Gets DRC flags.

Return Values

Returns LStatusOK if successful or LBadParameter if an error occurred.

Parameters

file	Current file.
pDRCFlags	Pointer to LDesignRuleFlags.

Example

```
LAmbiguousFillType
   GetActionOnPolygonsWithAmbiguousFills(LFile file)
```

```
{
LDesignRuleFlags drcFlags;/* LDesignRuleFlags structure */
LFile_GetDesignRuleFlags(file, &drcFlags);
/* get current flags */
return drcFlags.PolygonsWithAmbiguousFills; /* return one of
   the values */
}
```

See Also

LDesignRuleFlags (page 4-558), **LAmbiguousFillType** (page 4-584), **LFile_SetDesignRuleFlags** (page 4-152).

LFile_SetDesignRuleFlags

```
LStatus LEDITAPI LFile_SetDesignRuleFlags(LFile file,
   LDesignRuleFlags *pDrcFlags);
```

Description

Sets DRC flags.

Return Values

Returns LStatusOK if successful or LBadParameter if an error occurred.

Parameters

file	Specified file.
pDRCFlags	Pointer to LDesignRuleFlags.

Example

```
void SetFlagIgnoredObject(LFile file, LBoolean flagIgnored)
{
```

```
LDesignRuleFlagsdrcFlags;/* LDesignRuleFlags structure */
LFile_GetDesignRuleFlags(file, &drcFlags); /* get current
   flags */
drcFlags.FlagIgnoredObjects = flagIgnored; /* change one of
   the flags */
LFile_SetDesignRuleFlags(file, &drcFlags); /* modify current
   flags */
}
```

See Also

LDesignRuleFlags (page 4-558), **LAmbiguousFillType** (page 4-584), **LFile_GetDesignRuleFlags** (page 4-150).

Cell Functions

Cell functions allow the user to manipulate an individual cell in an L-Edit design file.

(continued)

LCell_Flatten (page 4-185)	**LCell_SetUserData** (page 4-188)
LCell_GetUserData (page 4-186)	**LCell_DeleteUserData** (page 4-190)

Subcategories of cell functions include:

- **Instance Functions** (page 4-191)
- **Object Functions** (page 4-232)
- **Selection Functions** (page 4-320)

LCell_New

```
LCell LCell_New(LFile file, char* name);
```

Description

Creates a new cell in the specified file.

Return Values

Returns a pointer to the newly created cell if successful; otherwise returns NULL.

Parameters

file	File where new cell need to be created.
name	Name of the new cell.

See Also

LCell (page 4-521), **LFile** (page 4-515), **Cell Functions** (page 4-154)

LCell_Delete

```
LStatus LCell_Delete(LCell cell);
```

Description

Deletes the specified cell from the current file.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameter

cell	Cell to be deleted.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Cell Functions** (page 4-154)

LCell_Copy

```
LStatus LCell_Copy(LFile sourceFile, LCell sourceCell, LFile
    destFile, char* destCellName);
```

Description

Copies a cell from one file (the “source” file) to another (the “destination” file—possibly the same) with a new name. If a cell with the new name already exists in the destination file, it is overwritten.

Return Values

Returns **LStatusOK** if no name collision occurs, **LCellOverWritten** if there is a collision. Returns **LBadParameters** if null parameters are passed, the source cell does not exist, or if ***sourceCell*** does not belong to ***sourceFile***. Returns **LLayerMapsDifferent** if the layer maps in ***sourceFile*** and ***destFile*** are not the same.

Parameters

sourceFile	Source file.
sourceCell	Cell to be copied.
destFile	Destination file.
destCellName	Name of the new cell.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LFile** (page 4-515), **Cell Functions** (page 4-154)

LCell_Find

```
LCell LCell_Find(LFile file, const char* name);
```

Description

Searches for a cell of the specified name in the specified file.

Return Values

Returns a pointer to the cell if found; otherwise returns NULL.

Parameters

file	File to search.
name	Cell name to search for.

See Also

LCell (page 4-521), **LFile** (page 4-515), **Cell Functions** (page 4-154)

LCell_GetFile

```
LFile LCell_GetFile(LCell cell);
```

Description

Returns a pointer to the parent file of the specified cell.

Return Values

Returns a pointer to the file if found; otherwise returns NULL.

Parameters

cell	Specified cell.

See Also

LFile (page 4-515), **LCell** (page 4-521), **Cell Functions** (page 4-154)

LCell_GetList

```
LCell LCell_GetList(LFile file);
```

Description

Gets a list of cells in the specified file.

Return Values

Returns a pointer to the head of the cell list if successful; otherwise returns NULL.

Parameters

file	Specified file.

See Also

LCell_GetNext (page 4-163), **LFile_GetList** (page 4-114), **LInstance_GetList** (page 4-203), **Cell Functions** (page 4-154)

LCell_GetNext

```
LCell LCell_GetNext(LCell cell);
```

Description

Gets the next cell in the current file's list of cells after the specified cell.

Return Values

Returns a pointer to the next cell if successful; otherwise returns NULL.

Parameters

cell	Specified cell.

See Also

LCell_GetList (page 4-162), **LFile_GetNext** (page 4-115), **LInstance_GetNext** (page 4-204), **Cell Functions** (page 4-154)

LCell_GetLock

```
int LCell_GetLock(LCell cell);
```

Description

Finds out if a cell is locked or not.

Return Values

Returns zero if the specified cell is unlocked; otherwise returns a nonzero value.

Parameters

cell	Cell to be checked.

See Also

LCell (page 4-521), **Cell Functions** (page 4-154)

LCell_SetLock

```
int LCell_SetLock(LCell cell, int set);
```

Description

Locks or unlocks the specified cell.

Return Values

Returns zero if the specified cell has been unlocked; otherwise returns a nonzero value.

Parameters

cell	Cell to be locked or unlocked.
set	Value that determines the cell's new status: zero unlocks; anything else locks.

See Also

LCell (page 4-521), **Cell Functions** (page 4-154)

LCell_GetName

```
char* LCell_GetName(LCell cell, char* name, const int
    maxlen);
```

Description

Gets the name of the specified cell.

Return Values

Returns a pointer to the string ***name*** if successful; otherwise returns NULL.

Parameters

cell	Cell whose name is to be retrieved.
name	String containing the name text.
maxlen	Maximum length allowed for ***name*****.**

See Also

LCell (page 4-521), **LFile_GetName** (page 4-120), **LInstance_GetName** (page 4-205), **Cell Functions** (page 4-154)

LCell_SetName

```
LStatus LCell_SetName(LFile file, LCell cell, const char*
    name);
```

Description

Sets the name of the specified cell in the specified file.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	File whose cell is being renamed.
cell	Cell to be (re)named.
name	New cell.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LFile** (page 4-515), **Cell Functions** (page 4-154)

LCell_GetAuthor

```
char* LCell_GetAuthor(LCell cell, char* author, const int
    maxlen);
```

Description

Gets the text of the string ***author*** for the specified cell.

Return Values

Returns a pointer to the string ***author*** if successful; otherwise returns NULL.

Parameters

cell	Cell whose author is to be retrieved.
author	String containing the author text.
maxlen	Maximum length allowed for ***author*.**

See Also

LCell (page 4-521), **LFile** (page 4-515), **Cell Functions** (page 4-154)

LCell_SetAuthor

```
char* LCell_SetAuthor(LCell cell, char* author);
```

Description

Sets the text of the string ***author*** for the specified cell.

Return Values

Returns a pointer to the structure containing the string ***author*** if successful; otherwise returns NULL.

Parameters

cell	Cell whose author is to be set.
author	String containing the ***author text.***

See Also

LCell (page 4-521), **Cell Functions** (page 4-154)

LCell_GetOrganization

```
char* LCell_GetOrganization(LCell cell, char* org, const int
    maxlen);
```

Description

Gets the organization text associated with the specified cell.

Return Values

Returns a pointer to the cell organization buffer if successful; otherwise returns NULL.

Parameters

cell	Cell whose organization is to be retrieved.
org	String containing the organization text.
maxlen	Maximum length allowed for ***org*.**

See Also

LCell (page 4-521), **Cell Functions** (page 4-154)

LCell_SetOrganization

```
char* LCell_SetOrganization(LCell cell, char* org);
```

Description

Sets the text of the organization field in the information summary of the specified cell. A NULL value may be given.

Return Values

Returns a pointer to the string containing the organization text if successful; otherwise returns NULL.

Parameters

cell	Cell whose organization is to be set.
org	String containing the organization text.

See Also

LCell (page 4-521), **Cell Functions** (page 4-154)

LCell_GetInfoText

```
char* LCell_GetInfoText(LCell cell, char* info, const int
    maxlen);
```

Description

Gets the text of the information field in the information summary of the specified cell.

Return Values

Returns a pointer to the cell info buffer if successful; otherwise returns NULL.

Parameters

cell	Cell whose information is to be retrieved.
info	String containing the information text.
maxlen	Maximum length allowed for ***info.***

See Also

LCell (page 4-521), **Cell Functions** (page 4-154)

LCell_SetInfoText

```
char* LCell_SetInfoText(LCell cell, char* info);
```

Description

Sets the text of the information field in the information summary of the specified cell. A NULL value may be given.

Return Values

Returns a pointer to the string ***info*** if successful; otherwise returns NULL.

Parameters

cell	Cell whose information is to be set.
info	String containing the new information text.

See Also

LCell (page 4-521), **Cell Functions** (page 4-154)

LCell_IsChanged

```
int LCell_IsChanged(LCell pCell)
```

Description

Checks the specified cell to determine if it has been changed since it was last saved.

Return Values

The function returns 1 if the cell has been changed or 0 if it has not.

Parameters

pCell	Cell to be checked.

See Also

LCell (page 4-521), **Cell Functions** (page 4-154)

LCell_GetView

```
LRect LCell_GetView(LCell cell);
```

Description

Gets the coordinates of the rectangle that defines the current view of the specified cell.

Return Values

Returns the coordinates of the viewing rectangle if successful; otherwise returns a rectangle whose coordinates are all zeros.

Parameters

cell	Cell whose viewing rectangle is needed.

See Also

LRect (page 4-528), **LCell** (page 4-521), **Cell Functions** (page 4-154)

LCell_SetView

```
LStatus LCell_SetView(LCell cell, LRect view);
```

Description

Sets the coordinates of the rectangle that defines the current view of the specified cell.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Cell whose viewing rectangle needs to be set.
view	New viewing rectangle.

See Also

LStatus (page 4-512), **LRect** (page 4-528), **LCell** (page 4-521), **Cell Functions** (page 4-154)

LCell_GetMbb

```
LRect LCell_GetMbb(LCell cell);
```

Description

Gets the coordinates of the rectangle representing the minimum bounding box (Mbb) of the specified cell, not including port text.

Return Values

Returns the Mbb if successful; otherwise returns a rectangle whose coordinates are all zeros.

Parameters

cell	Cell whose Mbb is to be found.

See Also

LRect (page 4-528), **LCell** (page 4-521), **Cell Functions** (page 4-154)

LCell_GetMbbAll

```
LRect LCell_GetMbbAll(LCell cell);
```

Description

Gets the coordinates of the rectangle representing the minimum bounding box (Mbb) of the specified cell, including port text.

Return Values

Returns the MbbAll rectangle if successful; otherwise returns a rectangle whose coordinates are all zeros.

Parameters

cell	Cell whose MbbAll is to be found.

See Also

LRect (page 4-528), **LCell** (page 4-521), **Cell Functions** (page 4-154)

LCell_Flatten

```
LCell LCell_Flatten(LCell cell);
```

Description

Flattens the specified cell.

Return Values

Returns a pointer to the flattened cell if successful; otherwise returns NULL.

Parameters

cell	Cell to be flattened.

See Also

LCell (page 4-521), **Cell Functions** (page 4-154)

LCell_GetUserData

`void* LCell_GetUserData(LCell` *`cell`***`);`**

Description

Gets a pointer to user-defined data associated with the specified cell.

Return Values

Returns a pointer to the user data if successful; otherwise returns NULL.

Parameter

cell	Cell whose user-defined data is needed.

Example

```
/*Declare user-defined data to be stored in a cell*/
typedef struct {
     int x;
     double y;
     float z;
} CellUserDataRec;

CellUserDataRec cd, *pd=NULL;

/*The Cell Pointer*/
LCell cell;

/*Get a pointer to the currently open cell*/
cell = LCell_GetVisible();

/*Fill in data into CellUserDataRec*/
cd.x = 1; cd.y = 2.0; cd.z = 1.5;

/*Store cd into cell's data pointer*/
LCell_SetUserData( cell, (void *) (&cd));

/*Get the data back from cell's data pointer into pd*/
pd = (CellUserDataRec *) LCell_GetUserData( cell );

/*pd now points to the user-defined data*/
```

See Also

LCell (page 4-521), **Cell Functions** (page 4-154)

LCell_SetUserData

`LStatus LCell_SetUserData(LCell` *`cell`***`, void*`** *`dataPointer`***`);`**

Description

Uses a data pointer within a cell to associate user-defined data with the cell. Data can be integer, string, or any other type.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameter

cell	Cell which will contain the user-defined data.

Example

```
/*Declare user-defined data to be stored in a cell*/
typedef struct {
     int x;
     double y;
     float z;
} CellUserDataRec;

CellUserDataRec cd;

/*The Cell Pointer*/
LCell cell;

/*Get a pointer to the currently open cell*/
cell = LCell_GetVisible();

/*Fill in data into CellUserDataRec*/
cd.x = 1; cd.y = 2.0; cd.z = 1.5;

/*Store cd into cell's data pointer*/
LCell_SetUserData( cell, (void *) (&cd));
```

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Cell Functions** (page 4-154)

LCell_DeleteUserData

```
LStatus LCell_DeleteUserData(LCell cell);
```

Description

Deletes the user-defined expansion pointer in the specified cell.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Specified cell.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Cell Functions** (page 4-154)

Instance Functions

An instance is a reference to a cell (the instanced cell) from within another cell (the instancing cell). Each instancing cell maintains a list of instances in an **LInstance** data structure.

Instance functions allow the user to manipulate an instance of a cell.

LInstance_New

```
LInstance LInstance_New(LCell cell, LCell instance_cell,
   LTransform transform, LPoint repeat_cnt, LPoint delta);
```

Description

Creates a new instance or array of instances in the specified cell. (An array is a geometrically regular two-dimensional arrangement of instances of a single cell.)

The array repeat count specified in ***repeat_cnt*** and array spacing offset specified in ***delta*** specify how an array of instances will be created.

The parameters ***cell*** and ***instance_cell*** must be in the same file.

Return Values

Returns a pointer to the newly created instance or array if successful; otherwise returns NULL.

Parameters

cell	Instancing cell.
instance_cell	Instanced cell.
transform	Translation and rotation of the new instance.
repeat_cnt	Ordered pair specifying the dimensions of the array. The first number in the pair specifies rows; the second number specified columns. Minimum value is 1,1.
delta	Ordered pair specifying the spacing offset of the array.

See Also

LInstance (page 4-522), **LCell** (page 4-521), **LTransform** (page 4-532), **LPoint** (page 4-527), **Instance Functions** (page 4-191)

LInstance_New_Ex99

```
LInstance LInstance_New_Ex99(LCell cell, LCell
    instance_cell, LTransform_Ex99 transform, LPoint
    repeat_cnt, LPoint delta);
```

Description

Creates a new instance or array of instances in the specified cell. (An array is a geometrically regular two-dimensional arrangement of instances of a single cell.)

The array repeat count specified in ***repeat_cnt*** and array spacing offset specified in ***delta*** specify how an array of instances will be created.

The parameters ***cell*** and ***instance_cell*** must be in the same file.

Return Values

Returns a pointer to the newly created instance or array if successful; otherwise returns NULL.

Parameters

cell	Instancing cell.
instance_cell	Instanced cell.
transform	Translation and rotation of the new instance.
repeat_cnt	Ordered pair specifying the dimensions of the array. The first number in the pair specifies rows; the second number specified columns. Minimum value is 1,1.
delta	Ordered pair specifying the spacing offset of the array.

See Also

LInstance_New (page 4-192), **LInstance** (page 4-522), **LCell** (page 4-521), **LTransform** (page 4-532), **LPoint** (page 4-527), **Instance Functions** (page 4-191)

LInstance_Delete

```
LStatus LInstance_Delete(LCell cell, LInstance instance);
```

Description

Deletes the specified instance from the specified cell.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Instancing cell.
instance	Instance to be deleted.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LInstance** (page 4-522), **Instance Functions** (page 4-191)

LInstance_Set

```
LStatus LInstance_Set(LCell cell, LInstance instance,
   LTransform transform, LPoint repeat_cnt, LPoint delta);
```

Description

Modifies the specified instance or array of instances in the specified cell with new values for translation, rotation, dimension, and offset.

Return Values

Returns **LStatusOK** if successful. If an error occurs, ***LStatus*** contains the error value.

Parameters

cell	Instancing cell.
instance	Instance to be modified.
transform	Translation, rotation, and magnification of the instance.
repeat_cnt	Dimensions of the array.
delta	Spacing offset of the array.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LInstance** (page 4-522), **LTransform** (page 4-532), **LPoint** (page 4-527), **Instance Functions** (page 4-191)

LInstance_Set_Ex99

```
LStatus LInstance_Set_Ex99(LCell cell, LInstance instance,
    LTransform_Ex99 transform, LPoint repeat_cnt, LPoint
    delta);
```

Description

Modifies the specified instance or array of instances in the specified cell with new values for translation, rotation, dimension, and offset.

Return Values

Returns **LStatusOK** if successful. If an error occurs, ***LStatus*** contains the error value.

Parameters

cell	Instancing cell.
instance	Instance to be modified.
transform	Translation, rotation, and magnification of the instance.
repeat_cnt	Dimensions of the array.
delta	Spacing offset of the array.

See Also

Instance Functions (page 4-191), **LStatus** (page 4-512), **LCell** (page 4-521), **LInstance** (page 4-522), **LPoint** (page 4-527), **LTransform** (page 4-532)

LInstance_Find

```
LInstance LInstance_Find(LCell cell, const char* name);
```

Description

Searches for an instance of the specified name in the specified cell.

Return Values

Returns a pointer to the instance if successful; otherwise returns NULL.

Parameters

cell	Instancing cell to search for instances.
name	Name of instance to search for.

See Also

LCell (page 4-521), **LInstance** (page 4-522), **Instance Functions** (page 4-191)

LInstance_FindNext

```
LInstance LInstance_FindNext(LInstance instance, const char*
    name);
```

Description

Continues the search for an instance of the specified name (proceeding from the last such instance).

Return Values

Returns a pointer to the next instance if successful; otherwise returns NULL.

Parameters

instance	Most recently found instance.
name	Name of instance to search for.

See Also

LInstance (page 4-522), **Instance Functions** (page 4-191)

LInstance_GetList

```
LInstance LInstance_GetList(LCell cell);
```

Description

Gets the first instance in the specified cell’s list of instances.

Return Values

Returns a pointer to the instance list if successful; otherwise returns NULL.

Parameters

cell	Instancing cell.

See Also

LInstance_GetNext (page 4-204), **LFile_GetList** (page 4-114), **LCell_GetList** (page 4-162), **Instance Functions** (page 4-191)

LInstance_GetNext

```
LInstance LInstance_GetNext(LInstance instance);
```

Description

Gets the next instance after the specified instance in the current cell's list of instances.

Return Values

Returns a pointer to the next instance if successful; otherwise returns NULL.

Parameters

instance	Specified instance.

See Also

LInstance (page 4-522), **LInstance_GetList** (page 4-203), **LFile_GetNext** (page 4-115), **LCell_GetNext** (page 4-163), **Instance Functions** (page 4-191)

LInstance_GetName

```
char* LInstance_GetName(LInstance instance, char* name,
   const int maxlen);
```

Description

Gets the name of the specified instance as a string (up to a maximum string length).

Return Values

Returns a pointer to the instance name buffer if successful; otherwise returns NULL.

Parameters

instance	Instance whose name is to be retrieved.
name	String (buffer) containing the name text.
maxlen	Maximum length allowed for ***name*****.**

See Also

LInstance (page 4-522), **LFile_GetName** (page 4-120), **LCell_GetName** (page 4-166), **Instance Functions** (page 4-191)

LInstance_SetName

```
LStatus LInstance_SetName(LCell cell, LInstance instance,
    char* name);
```

Description

Sets the name of the specified instance in the specified cell.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Cell containing the instance.
instance	Instance to be (re)named.
name	New name of the instance.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LInstance** (page 4-522), **Instance Functions** (page 4-191)

LInstance_GetCell

```
LCell LInstance_GetCell(LInstance instance);
```

Description

Gets the parent (instanced) cell of the specified instance.

Return Values

Returns a pointer to the parent cell if successful; otherwise returns NULL.

Parameter

instance	Specified instance.

See Also

LCell (page 4-521), **LInstance** (page 4-522), **Instance Functions** (page 4-191)

LInstance_GetTransform

```
LTransform LInstance_GetTransform(LInstance instance);
```

Description

Gets the transformation of the specified instance.

Return Values

Returns the translation, magnification, and rotation of the specified instance; returns a zero transform on error.

Parameters

instance	Specified instance.

See Also

LTransform (page 4-532), **LInstance** (page 4-522), **Instance Functions** (page 4-191)

LInstance_GetTransform_Ex99

```
LTransform_Ex99 LInstance_GetTransform_Ex99(LInstance
    instance);
```

Description

Gets the transformation of the specified instance.

Return Values

Returns the translation, magnification, and rotation of the specified instance; returns a zero transform on error.

Parameters

instance	Specified instance.

See Also

LInstance_GetTransform (page 4-210), **LInstance** (page 4-522), **LTransform** (page 4-532), **Instance Functions** (page 4-191)

LInstance_GetRepeatCount

```
LPoint LInstance_GetRepeatCount(LInstance instance);
```

Description

Gets the repeat count of an instance.

Return Values

Returns the array dimensionality of the specified instance as an ordered pair, or (1,1) for non-array instances; returns (0,0) on error.

Parameters

instance	Specified instance.

See Also

LPoint (page 4-527), **LInstance** (page 4-522), **Instance Functions** (page 4-191)

LInstance_GetDelta

```
LPoint LInstance_GetDelta(LInstance instance);
```

Description

Gets the array spacing of the specified instance as an ordered pair.

Return Values

Returns the array spacing of the specified instance as an ordered pair; returns (0,0) on error

Parameters

instance	Specified instance.

See Also

LPoint (page 4-527), **LInstance** (page 4-522), **Instance Functions** (page 4-191)

LInstance_GetMbb

```
LRect LInstance_GetMbb(LInstance instance);
```

Description

Gets the Mbb of an instance.

Return Values

Returns the coordinates of the rectangle representing the minimum bounding box (Mbb) of the specified instance; on error returns a rectangle whose coordinates are all zeros.

Parameters

instance	Specified instance.

See Also

LRect (page 4-528), **LInstance** (page 4-522), **Instance Functions** (page 4-191)

Entity Functions

An entity is an **LFile**, **LCell**, **LLayer**, or **LObject** (**LBox**, **LPolygon**, **LWire**, **LPort**, **LCircle**, and **LInstance**). Any entity can have properties. **LFile**, **LCell**, **LLayer**, and **LObject** must be cast to **LEntity** for use with the Entity Functions.

(continued)

LEntity_DeleteProperty (page 4-223)

LEntity_LoadBlobPropertyFromFile (page 4-230)

LEntity_SaveBlobPropertyToFile (page 4-231)

LEntity_PropertyExists

```
LStatus LEntity_PropertyExists (const LEntity entity, const
    char* name)
```

Description

Determines whether a property exists.

Return Values

Returns LStatusOK if the property is found. If an error occurs, LStatus contains the error value.

Parameters

entity	A pointer to an LEnitity.
name	The path of the property.

LEntity_GetPropertyType

```
LStatus LEntity_GetPropertyType (const LEntity entity, const
    char* name, LPropertyType* type)
```

Description

Retrieves the property’s type.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

entity	A pointer to an LEnitity.
name	The path of the property.
type	A pointer to the property type.

LEntity_GetPropertyValueSize

```
unsigned int LEntity_GetPropertyValueSize (const LEntity
    entity, const char* name)
```

Description

Retrieves the size of a property's value.

Return Values

Returns the size of the value if the property is found and it has a value; otherwise, returns zero.

Parameters

entity	A pointer to an LEnitity.
name	The path of the property.

LEntity_GetPropertyValue

```
LStatus LEntity_GetPropertyValue (const LEntity entity,
   const char* name, void* value, unsigned int max_size)
```

Description

Retrieves a property's value.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error.

Parameters

entity	A pointer to an LEnitity.
name	The path of the property.
value	A pointer to the value.
max_size	The maximum size of the buffer pointed to by the value.

LEntity_AssignProperty

```
LStatus LEntity_AssignProperty (LEntity entity, const char*
    name, LPropertyType type, const void* value)
```

Description

Creates a new property and assigns a type and value, or changes or removes the value of an existing property. An existing property's type cannot be changed.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error.

Parameters

entity	A pointer to an LEnitity.
name	The path of the property.
value	A pointer to the value. If NULL, no value is assigned to a new property, or the current value of an existing property is removed.
type	The property's type.

LEntity_AssignBlobProperty

```
LStatus LEntity_AssignBlobProperty (LEntity entity, const
   char* name, const void* value, unsigned int size)
```

Description

Creates a new blob property and a value, changes or removes the value of an existing property.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error.

Parameters

entity	A pointer to an LEnitity.
name	The path of the property.
value	A pointer to the value. If NULL, no value is assigned to a new property, or the current value of an existing property is removed.
size	The size of the value.

LEntity_DeleteProperty

```
LStatus LEntity_DeleteProperty (LEntity entity, const char*
    name)
```

Description

Deletes a property.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error.

Parameters

entity	A pointer to an LEnitity.
name	The path of the property.

LEntity_DeleteAllProperties

LStatus LEntity_DeleteAllProperties (LEntity *entity***)**

Description

Deletes all properties on an entity.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error.

Parameters

entity	A pointer to an LEnitity.
name	The path of the property.

LEntity_CopyAllProperties

```
LStatus LEntity_CopyAllProperties (LEntity target_entity,
    const LEntity source_entity)
```

Description

Copies all of one entity's properties to another entity overwriting the other entity's properties.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error.

Parameters

target_entity	A pointer to the target entity.
source_entity	A pointer to the source entity.

LEntity_GetFirstProperty

```
const char* LEntity_GetFirstProperty (const LEntity entity)
```

Description

Retrieves the first property of an entity.

Return Values

Returns the name first property on an entity or NULL if the entity has no properties.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error.

Parameters

entity	A pointer to an LEnitity.

LEntity_GetNextProperty

```
const char* LEntity_GetNextProperty ()
```

Description

Retrieves the next property of an entity or NULL if there are no more properties on the entity.

Note: If the current property is deleted or renamed, the next call will return NULL, unless an appropriate call to **`LEntity_SetCurrentProperty`** is made first.

Return Values

Returns the name of the next property on an entity or NULL if the entity has no more properties.

Parameters

entity	A pointer to an LEnitity.

LEntity_SetCurrentProperty

```
void LEntity_SetCurrentProperty ()
```

Description

Sets the name of the current property in the traversal of the property tree.

Return Values

Returns the name of the path of the current property on an entity.

Parameters

name	The full path of the property.

LEntity_BrowseProperties

```
LStatus LEntity_BrowseProperties (LEntity entity)
```

Description

Invokes the standard property browser.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error.

Parameters

entity	The entity on which to invoke the browser.

LEntity_LoadBlobPropertyFromFile

```
LStatus LEntity_LoadBlobPropertyFromFile (LEntity entity,
   const char* name, const char* file_name)
```

Description

Sets a blob property from a file.

Return Values

Returns the name of the next property on an entity or NULL if the entity has no more properties.

Parameters

entity	A pointer to an LEnitity.
name	The path of the property
file_name	The name of the file containing the value.

LEntity_SaveBlobPropertyToFile

```
LStatus LEntity_SaveBlobPropertyToFile (const LEntity
    entity, const char* name, void* value, const char*
    file_name)
```

Description

Saves a blob property's value to a file.

Return Values

Returns the name of the next property on an entity or NULL if the entity has no more properties.

Parameters

entity	A pointer to an LEnitity.
name	The path of the property
file_name	The name of the file containing the value.

Object Functions

An object is a fundamental geometric shape. Objects include points, boxes, circles, wires, polygons, and ports.

Object functions allow the user to manipulate an object in a cell.

LObject_Area (page 4-243)

LObject_Perimeter (page 4-244)

LObject_GetLayer (page 4-245)

LVertex_Delete (page 4-266)

Subcategories of object functions include:

Box Functions (page 4-267)

Circle Functions (page 4-273)

Wire Functions (page 4-281)

Polygon Functions (page 4-292)

Port Functions (page 4-299)

LObject_Delete

```
LStatus LObject_Delete(LCell cell, LObject object);
```

Description

Removes object from cell.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Cell containing the object to be deleted.
object	Object to be deleted.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Object Functions** (page 4-232)

LObject_GetList

```
LObject LObject_GetList(LCell cell, LLayer layer);
```

Description

Gets a list of objects in the specified cell on the specified layer.

Return Values

Returns a pointer to the head object in the current object list if successful; otherwise returns NULL.

Parameters

cell	Specified cell.
layer	Layer on which objects are drawn.

See Also

LCell (page 4-521), **LLayer** (page 4-542), **Object Functions** (page 4-232)

LObject_GetNext

```
LObject LObject_GetNext(LObject object);
```

Description

Gets the next object which follows the specified object.

Return Values

Returns a pointer to the next object in the object list if successful; otherwise returns NULL.

Parameters

object	Specified object.

See Also

LObject (page 4-523), **Object Functions** (page 4-232)

LObject_Transform

```
void LObject_Transform(LObject object, LTransform
    transform);
```

Description

Transforms an object.

Parameters

object	Specified object.
transform	Specified transform.

See Also

LObject (page 4-523), **LTransform** (page 4-532), **Object Functions** (page 4-232)

LObject_Transform_Ex99

```
void LObject_Transform_Ex99(LObject object, LTransform_Ex99
    transform);
```

Description

Transforms an object.

Parameters

object	Specified object.
transform	Specified transform.

See Also

LObject_Transform (page 4-237), **LObject** (page 4-523), **LTransform** (page 4-532), **Object Functions** (page 4-232)

LObject_GetMbb

```
LRect LObject_GetMbb(LObject object);
```

Description

Gets the minimum bounding box of an object.

Return Values

Returns the coordinates of the Mbb rectangle if successful; otherwise returns a rectangle whose coordinates are all zeros.

Parameters

object	Specified object.

See Also

LObject (page 4-523), **Object Functions** (page 4-232)

LObject_GetShape

```
LShapeType LObject_GetShape(LObject pObject)
```

Description

Gets the shape of an object.

Return Values

Returns the shape of object as an LShapeType enum. Possible values include box, circle, wire, polygon, torus, pie wedge, instance, port, ruler, or other.

Parameters

pObject	Specified object.

See Also

LShapeType (page 4-524), **LObject** (page 4-523), **Object Functions** (page 4-232)

LObject_GetGeometry

```
LGeomType LObject_GetGeometry(LObject object);
```

Description

Gets the geometry specifications of an object.

Return Values

Returns the geometric constraint of object as an LGeomType enum. Geometry types include orthogonal, 45-degree angle, and all-angle.

Parameters

object	Specified object.

See Also

LGeomType (page 4-525), **LObject** (page 4-523), **Object Functions** (page 4-232)

LObject_GetVertexList

LVertex LObject_GetVertexList (LObject *object***)**

Description

Retrieves the first vertex of an object. This works only on **LPolygon** and **LWire**.

Return Values

Returns a pointer to the first vertex in a polygon or wire object's vertex list or NULL if no vertices exist for the object.

Parameters

object	The specified object.

LObject_Area

```
double LObject_Area(LObject pObject)
```

Description

Calculates the area of a box, polygon, wire, circle, pie wedge or torus.

Return Values

The area of the object in Internal Units squared.

Parameters

pObject	Specified object.

See Also

LObject_Perimeter (page 4-244), **LObject** (page 4-523), **Object Functions** (page 4-232)

LObject_Perimeter

```
double LObject_Perimeter(LObject pObject)
```

Description

Calculates the perimeter of a box, polygon, wire, circle, pie wedge or torus.

Return Values

The perimeter of the object in Internal Units.

Parameters

pObject	Specified object.

See Also

LObject_Area (page 4-243), **LObject** (page 4-523), **Object Functions** (page 4-232)

LObject_GetLayer

```
LLayer LObject_GetLayer(LCell pCell, LObject pObject)
```

Description

Retrieves the layer of the specified object.

Return Values

Returns the layer of the object if successful; otherwise returns NULL.

Parameters

pCell	Specified cell containing the object.
pObject	Specified object.

See Also

LObject (page 4-523), **Object Functions** (page 4-232), **LLayer** (page 4-542)

LObject_ChangeLayer

LStatus LObject_ChangeLayer(LCell *pCell*, **LObject** *pObject*,
    LLayer *pNewLayer***)**

Description

Changes the layer of an object to a different layer. After this function is called, the pObject pointer is no longer valid for getting the next object (LObject_GetNext) (see example below).

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

LBadCell - *pCell* is NULL.

LBadObject - *pObject* is NULL, *pObject* is an instance, or object is corrupted.

LBadLayer - *pLayer* is NULL.

Parameters

pCell	Specified cell containing the object.
pObject	Object to change the layer of.
pNewLayer	New layer.

Example

```
LCell  pCell = LCell_GetVisible(); // The current cell.
if(NotAssigned(pCell))
{
        LDialog_AlertBox("ERROR:  Could not find a Visible
    Cell.");
        return;
}

LFile  pTDBFile = LCell_GetFile(pCell);// The TDB current
    file.
if(NotAssigned(pTDBFile))
{
        LDialog_AlertBox("ERROR:  Could not get the TDB file
    from the Visible Cell.");
        return;
}
```

```
// Change all objects on Poly to Metal1.
LLayer pPoly = LLayer_Find(pTDBFile, "Poly");
LLayer pMetal1 = LLayer_Find(pTDBFile, "Metal1");
if(Assigned(pPoly) && Assigned(pMetal1))
{
       LObject pObject = NULL, pNextObject = NULL;
       for(pObject = LObject_GetList(pCell, pPoly);
   Assigned(pObject); pObject = pNextObject)
       {
              // After the LObject_ChangeLayer call, the
   pObject pointer will no longer be valid.
              //            So get the next object before you
   change the layer.
              pNextObject = LObject_GetNext(pObject);
              if(LObject_ChangeLayer(pCell, pObject,
   pMetal1) == LStatusOK)
              {
                     // The layer was changed to Metal1.
              }
       } // endfor(pObject = LObject_GetList(pCell, pPoly);
   Assigned(pObject); pObject = pNextObject)
}  // endif(Assigned(pLayer))
```

Version

Available in L-Edit 8.3 and later versions.

See Also

Object Functions (page 4-232), **LStatus** (page 4-512), **LCell** (page 4-521), **LObject** (page 4-523), **LLayer** (page 4-542), **LObject_GetLayer** (page 4-245), **LVertex_GetNext** (page 4-260).

LObject_GetGDSIIDataType

```
short LObject_GetGDSIIDataType(LObject pObject)
```

Description

Retrieves the GDSII data type of an object.

Return Values

The GDSII data type if successful, **-1** if an error occurred such as pObject is NULL or pObject is an instance.

Parameters

pObject	The specified object.

Example

```
LFile pFile = LFile_GetVisible();
if(Assigned(pFile))
{
        // Get the GDSII Data Type of the first object in the
    selected.
        LSelection pSelection = LSelection_GetList();
        if(Assigned(pSelection))
        {
                LObject pObject =
    LSelection_GetObject(pSelection);
                if(Assigned(pObject))
                {
                        short iObjectDataType =
    LObject_GetGDSIIDataType(pObject);
                        // More Processing
                        // ...
                }
        }
}
        LDialog_AlertBox("ERROR:  Could not find a Visible
    Cell.");
        return;
}

LFile  pTDBFile = LCell_GetFile(pCell);// The TDB current
    file.
if(NotAssigned(pTDBFile))
{
        LDialog_AlertBox("ERROR:  Could not get the TDB file
    from the Visible Cell.");
        return;
}
```

```
// Change all objects on Poly to Metal1.
LLayer pPoly = LLayer_Find(pTDBFile, "Poly");
LLayer pMetal1 = LLayer_Find(pTDBFile, "Metal1");
if(Assigned(pPoly) && Assigned(pMetal1))
{
        LObject pObject = NULL, pNextObject = NULL;
        for(pObject = LObject_GetList(pCell, pPoly);
    Assigned(pObject); pObject = pNextObject)
        {
                // After the LObject_ChangeLayer call, the
    pObject pointer will no longer be valid.
                //          So get the next object before you
    change the layer.
                pNextObject = LObject_GetNext(pObject);
                if(LObject_ChangeLayer(pCell, pObject,
    pMetal1) == LStatusOK)
                {
                        // The layer was changed to Metal1.
                }
        } // endfor(pObject = LObject_GetList(pCell, pPoly);
    Assigned(pObject); pObject = pNextObject)
}  // endif(Assigned(pLayer))
```

Version

Available in L-Edit 8.2 and later versions.

See Also

LObject_SetGDSIIDataType (page 4-254), **Object Functions** (page 4-232), Edit Objects, **LObject** (page 4-523), **LStatus** (page 4-512).

LObject_SetGDSIIDataType

```
LStatus LObject_SetGDSIIDataType(LObject pObject, short
    GDSIIDataType)
```

Description

Sets the GDSII data type of an object.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value with possible values:

LBadObject - where pObject is NULL or pObject is an instance.

Parameters

pObject	Specified object.
GDSIIDataType	GDSII data type.

Example

```
LFile pFile = LFile_GetVisible();
if(Assigned(pFile))
{
      LCell pCell = LCell_Find(pFile, "MyCell");
      LLayer pLayer = LLayer_Find(pFile, "Poly");
      if(Assigned(pCell) && Assigned(pLayer))
      {
            LObject pObject;
            // Set the GDSII data type of each object on
   Poly in cell MyCell to 12.
            for(pObject = LObject_GetList(pCell, pLayer);
                   Assigned(pObject);
                   pObject = LObject_GetNext(pObject))
            {
                  if(LObject_SetGDSIIDataType(pObject, 12)
   != LStatusOK)
                  {
                        // Some problem occurred.
                        break;
```

```
                    }
                }
            }
        }
```

See Also

Object Functions (page 4-232), **LObject_GetGDSIIDataType** (page 4-250), Edit Objects, **LStatus** (page 4-512).

LVertex_GetCount

```
long LVertex_GetCount(LObject object);
```

Description

Gets the number of vertices in a polygon or wire.

Return Values

Returns the number of vertices in object of type polygon or wire; on error returns -1.

Parameters

object	Specified object.

See Also

LObject (page 4-523), **Object Functions** (page 4-232)

LVertex_GetArray

```
long LVertex_GetArray(LObject object, LPoint point_arr[],
    const int maxpoints);
```

Description

Fills an array with the vertices stored in an object. If the number of vertices is greater than ***maxpoints***, the extra vertices are ignored.

Return Values

Returns the number of vertices in object of type polygon or wire; on error returns -1.

Parameters

cell	Specified object.
point_arr	Array of vertices.
maxpoints	Maximum number of vertices allowed.

See Also

LObject (page 4-523), **LPoint** (page 4-527), **Object Functions** (page 4-232)

LVertex_GetNext

`LVertex LVertex_GetNext (LVertex` *`vertex`***`)`**

Description

Gets the next vertex of an object.

Return Values

Returns a pointer to the next vertex in a polygon or wire object's vertex list or NULL if no vertices exist for the object.

Parameters

vertex	The previous vertex.

Example

```
/* for each vertex of the polygon */
for (LVertex Vertex = LObject_GetVertexList (MyPolygon);
     vertex !=NULL;
```

```
          Vertex = LVertex_GetNext(Vertex);
{
          /* do something with the current vertex */
}
```

LVertex_GetPoint

```
LPoint LVertex_GetPoint (LVertex vertex)
```

Description

Gets the x and y coordinates for a vertex.

Return Values

Returns a point structure containing the coordinates. If the vertex pointer was invalid, the return value is not defined.

Parameters

vertex	A specified vertex.

LVertex_SetPoint

LStatus LVertex_SetPoint (LVertex *vertex***, LPoint** *point***)**

Description

Sets the x and y coordinates for a vertex.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

vertex	A specified vertex.
point	A point structure with the x and y coordinates.

Example

```
/* get the point information associated with MyVertex */
LPoint Point = LVertex_GetPoint(MyVertex)
```

```
/* change the position of the point */
Point.y +=10;
Point.x -=20;

/* update the position of the MyVertex */
LVertex_SetPoint(MyVertex, point);
```

LVertex_Add

`LVertex LVertex_Add (LObject` *`object`***`, const LVertex`**
    *`prev_vertex`***`, LPoint`** *`point`***`)`**

Description

Adds a vertex to the object. The object can be an LPolygon or LWire. *`prev_vertex`* is a pointer to the previous vertex. If *`prev_vertex`*=NULL, the vertex is added to the head of the list.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

object	Object to add vertex to.
prev_vertex	The previous vertex.
point	A point structure with the x and y coordinate.

LVertex_Delete

```
LStatus LVertex_Delete (LObject object, LVertex vertex)
```

Description

Deletes the vertex from the object. The object can be an LPolygon or LWire. This function will delete only if more than three vertices exist.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

object	The object to delete the vertex from.
vertex	The vertex to be deleted.

Box Functions

LBox_New

```
LObject LBox_New(LCell cell, LLayer layer, LCoord x0, LCoord
    y0, LCoord x1, LCoord y1);
```

Description

Creates a new box object in ***cell*** on ***layer*** with the given coordinates.

Return Values

Returns a pointer to the newly created box if successful; otherwise returns NULL.

Parameters

cell	Cell where box will be drawn.
layer	Layer on which the box will be drawn.
x0	Lower left x-coordinate of box.
y0	Lower left y-coordinate of box.
x1	Upper right x-coordinate of box.
y1	Upper right y-coordinate of box.

See Also

LObject (page 4-523), **LCell** (page 4-521), **LLayer** (page 4-542), **Box Functions** (page 4-267)

LBox_Set

`LStatus LBox_Set(LCell` *`cell`***`, LObject`** *`object`***`, LRect`** *`box`***`);`**

Description

Modifies the coordinates of the object in ***cell*** according to the specification contained in ***box.***

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Cell that contains the box.
object	Pointer to the box object.
box	New coordinates of the box.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LObject** (page 4-523), **LRect** (page 4-528), **Box Functions** (page 4-267)

LBox_GetRect

```
LRect LBox_GetRect(LObject object);
```

Description

Returns the minimum bounding box (MBB) of the specified box.

Return Values

If successful, an LRect structure containing the minimum bounding box (MBB) of the specified box; on error, a rectangle whose coordinates are all zeros.

Parameters

object	Specified box object.

See Also

LRect (page 4-528), **LObject** (page 4-523), **Box Functions** (page 4-267)

Circle Functions

LCircle_New

```
LObject LCircle_New(LCell cell, LLayer layer, LPoint center,
    LCoord radius);
```

Description

Creates a new circle in ***cell*** on ***layer*** with the center and radius specified by ***center*** and ***radius***.

Return Values

Returns a pointer to the newly created circle if successful; otherwise returns NULL.

Parameters

cell	Cell where the new circle is to be drawn.
layer	Layer on which circle is to be drawn.
center	x- and y- coordinates of the center.
radius	Radius of the circle.

See Also

LObject (page 4-523), **LCell** (page 4-521), **LLayer** (page 4-542), **LPoint** (page 4-527), **LCoord** (page 4-526), **Circle Functions** (page 4-273)

LCircle_Set

```
LStatus LCircle_Set(LCell cell, LObject object, LPoint
    center, LCoord radius);
```

Description

Modifies object in ***cell*** to the new ***center*** and ***radius*.**

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Cell where the circle is drawn.
object	Circle object.
center	New x- and y- coordinates of the center.
radius	New circle radius.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LObject** (page 4-523), **LPoint** (page 4-527), **LCoord** (page 4-526), **Circle Functions** (page 4-273)

LCircle_GetCenter

```
LPoint LCircle_GetCenter(LObject object);
```

Description

Gets the coordinates of the center of a circle.

Return Values

Returns the center point of object, or (0,0) on error

Parameters

object	Circle object.

See Also

LPoint (page 4-527), **LObject** (page 4-523), **Circle Functions** (page 4-273)

LCircle_GetRadius

```
LCoord LCircle_GetRadius(LObject object);
```

Description

Gets the radius of a circle.

Return Values

Returns the radius of object, or (0,0) on error

Parameters

cell	Circle object.

See Also

LCoord (page 4-526), **LObject** (page 4-523), **Circle Functions** (page 4-273)

LCircle_GetRect

```
LRect LCircle_GetRect(LObject object);
```

Description

Returns the minimum bounding box (MBB) of the specified circle.

Return Values

If successful, an LRect structure containing the minimum bounding box (MBB) of the specified circle; on error, a rectangle whose coordinates are all zeros.

Parameters

object	Specified circle object.

See Also

LRect (page 4-528), **LObject** (page 4-523), **Circle Functions** (page 4-273)

Wire Functions

LWire_New

```
LObject LWire_New(LCell cell, LLayer layer, LWireConfig*
    config, LWireConfigBits bits, Lpoint point_arr[], const
    int npoints);
```

Description

Creates a new wire in ***cell*** on ***layer***. The new wire will have ***npoints*** set to the values in the array ***point_arr***. If ***config*** is NULL or ***bits*** is zero, the wire will have the default width, join, and end styles of the corresponding layer. If ***bits*** is set to a mask of LWireConfigBits enum values, then values from the structure ***config*** will be used to override the defaults for the settings of ***bits***.

Return Values

Pointer to the newly created wire if successful; NULL otherwise.

Parameters

cell	Cell which will contain the wire.
layer	Wire layer.
config	Pointer to the wire configuration structure.
bit	Wire configuration bits.
point_arr	Array of wire vertices.
npoints	Number of wire vertices.

See Also

LObject (page 4-523), **LLayer** (page 4-542), **LWireConfig** (page 4-535), **LWireConfigBits** (page 4-536), **LPoint** (page 4-527), **Wire Functions** (page 4-281)

LWire_GetWidth

```
LCoord LWire_GetWidth(LObject object);
```

Description

Gets the wire width.

Return Values

Returns the width setting of the object, or zero on error.

Parameters

object	Specified wire object.

See Also

LCoord (page 4-526), **LObject** (page 4-523), **Wire Functions** (page 4-281)

LWire_GetCapType

```
LCapType LWire_GetCapType(LObject object);
```

Description

Gets the wire cap type.

Return Values

Returns the wire cap style of object. The return value is undefined on error.

Parameters

object	Wire object.

See Also

LCapType (page 4-537), **LObject** (page 4-523), **Wire Functions** (page 4-281)

LWire_GetJoinType

```
LJoinType LWire_GetJoinType(LObject object);
```

Description

Gets the wire join type

Return Values

Returns the wire join style of object—miter, round, or bevel. The return value is undefined on error.

Parameters

object	Wire object.

See Also

LJoinType (page 4-538), **LObject** (page 4-523), **Wire Functions** (page 4-281)

LWire_GetMiterAngle

```
short LWire_GetMiterAngle(LObject Object);
```

Description

Gets the wire miter angle

Return Values

Returns the miter angle of object. It returns -1 on error.

Parameters

object	Wire object.

See Also

LObject (page 4-523), **Wire Functions** (page 4-281)

LWire_GetLength

```
double LWire_GetLength(LObject pObject)
```

Description

Calculates the centerline length of the wire including end styles.

Return Values

The centerline length of the wire in Internal Units.

Parameters

pObject	Specified object.

See Also

LWire_GetSquares (page 4-289), **LWire_GetResistance** (page 4-290), **Wire Functions** (page 4-281)

LWire_GetSquares

double LWire_GetSquares(LObject *pObject*)

Description

Calculates the number of squares of an orthogonal wire including end styles. In the calculation of the number of squares, corners are counted as ½ a square.

Return Values

The number of squares of an orthogonal wire in Internal Units. If the object is not an orthogonal wire, then zero.

Parameters

pObject	Specified object.

See Also

LWire_GetLength (page 4-288), **LWire_GetResistance** (page 4-290), **Wire Functions** (page 4-281)

LWire_GetResistance

`double LWire_GetResistance(LObject` ***`pObject`*** **`)`**

Description

Calculates the resistance of an orthogonal wire including end styles. This uses the Resistivity on the Setup Layers dialog and the number of squares of the wire. In the calculation of the number of squares, corners are counted as ½ a square.

Return Values

The resistance of an orthogonal wire in Ohms. If the object is not an orthogonal wire, then zero.

Parameters

pObject	Specified object.

See Also

LWire_GetLength (page 4-288), **LWire_GetSquares** (page 4-289), **Wire Functions** (page 4-281)

Polygon Functions

LPolygon_New

```
LObject LPolygon_New(LCell cell, LLayer layer, LPoint
    point_arr[], const int npoints);
```

Description

Creates a new polygon object in ***cell*** on ***layer***. The new polygon will have ***npoints*** vertices at locations specified in ***point_arr***.

Return Values

Returns a pointer to the newly created polygon if successful; NULL otherwise.

Parameters

cell	Cell which will contain the polygon.
layer	Wire layer.
point_arr	Array of polygon vertices.
npoints	Number of polygon vertices.

See Also

LObject (page 4-523), **LCell** (page 4-521), **LLayer** (page 4-542), **LPoint** (page 4-527), **Polygon Functions** (page 4-292)

LPolygon_WireToPolygon

```
LObject LPolygon_WireToPolygon(LCell cell, LLayer layer,
    LObject object);
```

Description

Converts a wire object to a polygon object.

Return Values

Returns a pointer to the newly converted polygon if successful; NULL otherwise.

Parameters

cell	Cell containing the wire object.
layer	Wire layer.
object	Wire object.

See Also

LObject (page 4-523), **LCell** (page 4-521), **LLayer** (page 4-542), **Polygon Functions** (page 4-292)

LPolygon_CircleToPolygon

```
LObject LPolygon_CircleToPolygon(LCell cell, LLayer layer,
    LObject object, int NumSides);
```

Description

Converts a circle to a polygon with the given number of sides.

Return Values

Returns a pointer to the newly converted polygon if successful; NULL otherwise.

Parameters

cell	Cell containing the circle.
layer	Circle layer.
object	Circle object.
NumSides	Number of sides in the new polygon.

See Also

LObject (page 4-523), **LCell** (page 4-521), **LLayer** (page 4-542), **Polygon Functions** (page 4-292)

Port Functions

LPort_New

```
LPort LPort_New(LCell cell, LLayer layer, char* text, LCoord
    x0, LCoord y0, LCoord x1, LCoord y1);
```

Description

Creates a new port in ***cell*** on ***layer*** with the specified text and rectangle (location) coordinates x0, y0, x1, y1.

Return Values

Pointer to the newly created port if successful; NULL otherwise.

Parameters

cell	Cell that will contain the port.
layer	Port layer.
text	Port text string.
x0	Lower left x-coordinate of port rectangle.
y0	Lower left y-coordinate of port rectangle.
x1	Upper right x-coordinate of port rectangle.
y1	Upper right y-coordinate of port rectangle.

See Also

LPort (page 4-539), **LCell** (page 4-521), **LLayer** (page 4-542), **LCoord** (page 4-526), **Port Functions** (page 4-299)

LPort_Delete

```
LStatus LPort_Delete(LCell cell, LPort port);
```

Description

Deletes the specified port from the given cell.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Cell containing the port.
port	Port to be deleted.

See Also

LStatus (page 4-512), **LPort** (page 4-539), **LCell** (page 4-521), **Port Functions** (page 4-299)

LPort_Find

```
LPort LPort_Find(LCell cell, const char* name);
```

Description

Finds the first port in ***cell*** with the name specified in ***name***.

Return Values

Pointer to the port if successful; NULL otherwise.

Parameters

cell	Cell containing the port.
name	Port string to search for.

See Also

LPort (page 4-539), **LCell** (page 4-521), **Port Functions** (page 4-299)

LPort_FindNext

```
LPort LPort_FindNext(LPort port, const char* name);
```

Description

Finds the next port after ***port*** that has the name specified in ***name.***

Return Values

Pointer to the port if successful; NULL otherwise.

Parameters

port	Specified port.
name	Port string to search for.

See Also

LPort (page 4-539), **Port Functions** (page 4-299)

LPort_GetList

```
LPort LPort_GetList(LCell cell);
```

Description

Gets a pointer to the first port in the given cell.

Return Values

Pointer to the head of the port list if successful; NULL otherwise.

Parameters

cell	Specified cell.

See Also

LPort (page 4-539), **LCell** (page 4-521), **Port Functions** (page 4-299)

LPort_GetNext

```
LPort LPort_GetNext(LPort port);
```

Description

Gets a pointer to the port immediately following ***port*** in the port list.

Return Values

Pointer to the next element in the port list if successful; NULL otherwise.

Parameters

port	Specified port.

See Also

LPort (page 4-539), **Port Functions** (page 4-299)

LPort_GetText

```
char* LPort_GetText(LPort port, char* name, const int
    maxlen);
```

Description

Gets the text of a port. It the port text is longer than ***maxlen***, the extra characters are ignored.

Return Values

Pointer to the port text buffer if successful; NULL otherwise.

Parameters

port	Port whose text is required.
name	String (buffer) containing the port text.
maxlen	Maximum length allowed for port text.

See Also

LPort (page 4-539), **Port Functions** (page 4-299)

LPort_SetText

```
char* LPort_SetText(LCell cell, LPort port, char* text,
    LCoord textsize);
```

Description

Sets the text of a port.

Return Values

Pointer to the port text string if successful; NULL otherwise.

Parameters

cell	Cell containing the port.
port	Port whose text is being modified.
text	String (buffer) containing the port text.
textsize	Port text size.

See Also

LPort (page 4-539), **LCell** (page 4-521), **LCoord** (page 4-526), **Port Functions** (page 4-299)

LPort_GetTextSize

```
LCoord LPort_GetTextSize(LPort port);
```

Description

Gets the port text size.

Return Values

The port text size if successful; zero on error

Parameters

port	Specified port.

See Also

LCoord (page 4-526), **LPort** (page 4-539), **Port Functions** (page 4-299)

LPort_GetLayer

```
LLayer LPort_GetLayer(LPort port);
```

Description

Gets the layer that a port is drawn on.

Return Values

Pointer to the port layer if successful; NULL otherwise.

Parameters

port	Specified port.

See Also

LLayer (page 4-542), **LPort** (page 4-539), **Port Functions** (page 4-299)

LPort_GetMbb

```
LRect LPort_GetMbb(LPort port);
```

Description

Gets the minimum bounding box (Mbb) of a port.

Return Values

The minimum bounding box if successful, or on error a rectangle whose coordinates are all zeros.

Parameters

port	Specified port.

See Also

LRect (page 4-528), **LPort** (page 4-539), **Port Functions** (page 4-299)

LPort_GetRect

```
LRect LPort_GetRect(LPort port);
```

Description

Returns the rectangle (location) of the port.

Return Values

If successful, an LRect structure containing the location of the port; on error, a rectangle whose coordinates are all zeros.

Parameters

port	Specified port.

See Also

LRect (page 4-528), **LPort** (page 4-539), **Port Functions** (page 4-299)

LPort_Set

```
LStatus LPort_Set(LCell cell, LPort port, LRect rect);
```

Description

Modifies the rectangle (location) of the specified port in the specified cell according to the value specified in ***rect*.**

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Cell containing the port.
port	Port to be modified.
rect	New location of the port.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LPort** (page 4-539), **LRect** (page 4-528), **Port Functions** (page 4-299)

LPort_SetTextSize

```
LStatus LPort_SetTextSize(LPort pPort, LCoord lcTextSize);
```

Description

Sets the text size of a port. Available in L-Edit 8.2 and later versions.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value with the following possible values:

```
LBadParameters - pPort is NULL
```

Parameters

pPort	Specified port.
lcTextSize	Text size in Internal Units.

Example

```
LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
        LPort pPort = LPort_Find(pCell, "Gnd");
        if(Assigned(pPort))
        {
                if(LPort_SetTextSize(pPort,
    LFile_LocUtoIntU(LCell_GetFile(pCell), 2.5)) ==
    LStatusOK)
                {
                        // More Processing
                        // ...
                } // endif(LPort_SetTextSize(pPort,
    LFile_LocUtoIntU(LCell_GetFile(pCell), 2.5)) ==
    LStatusOK)
        } // endif(Assigned(pPort))
} // endif(Assigned(pCell))
```

See Also

Port Functions on page 4-299, LPort_GetTextSize on page 4-311, LStatus on page 4-512, LPort on page 4-539, LCoord on page 4-526.

Selection Functions

Selected objects may be those selected with the mouse, those falling into a drawn box, all objects on a particular layer, or all objects of a particular cell. Once selected, they are entered into an internal selection list. Several functions may be applied to objects found in the selection list.

Selection functions allow the user to manipulate a selection in L-Edit.

(continued)

LSelection_Cut

```
LStatus LSelection_Cut (void);
```

Description

Removes all objects in the selection and copies them into the paste buffer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_Copy

```
LStatus LSelection_Copy(void);
```

Description

Copies all objects in the selection to the paste buffer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_Paste

```
LStatus LSelection_Paste(void);
```

Description

Pastes the contents of the paste buffer into the Work Area.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_PasteToLayer

```
LStatus LSelection_PasteToLayer(LLayer layer);
```

Description

Pastes the contents of the paste buffer to the given layer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

layer	Layer on which paste buffer contents are to be pasted.

See Also

LLayer (page 4-542), **LStatus** (page 4-512), **Selection Functions** (page 4-320)

LSelection_Clear

```
LStatus LSelection_Clear(void);
```

Description

Removes all objects in the current selection.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_SelectAll

```
LSelection LSelection_SelectAll(void);
```

Description

Selects all objects in the current cell.

Return Values

Returns a pointer to the head of the selection list if successful; NULL otherwise.

See Also

Selection Functions (page 4-320)

LSelection_DeselectAll

```
void LSelection_DeselectAll(void);
```

Description

Deselects all objects in the current cell.

See Also

Selection Functions (page 4-320)

LSelection_AddObject

```
LStatus LSelection_AddObject(LObject obj);
```

Description

Adds an object to the selection list.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

obj	Object to be added to the selection list.

See Also

LStatus (page 4-512), **LObject** (page 4-523), **Selection Functions** (page 4-320)

LSelection_RemoveObject

```
LStatus LSelection_RemoveObject(LObject obj);
```

Description

Removes an object from the selection list.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

obj	Object to be removed from the selection list.

See Also

LStatus (page 4-512), **LObject** (page 4-523), **Selection Functions** (page 4-320)

LSelection_GetObject

```
LObject LSelection_GetObject(LSelection selection);
```

Description

Gets the object associated with a selection element.

Return Values

Pointer to the selection object if successful; NULL otherwise.

Parameters

selection	Pointer to the selection element.

See Also

LObject (page 4-523), **LSelection** (page 4-540), **Selection Functions** (page 4-320)

LSelection_AddAllObjectsOnLayer

```
LStatus LSelection_AddAllObjectsOnLayer(LLayer layer);
```

Description

Adds all objects on ***layer*** to the selection list.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

layer	Layer whose objects are to be added to the selection.

See Also

LStatus (page 4-512), **LLayer** (page 4-542), **Selection Functions** (page 4-320)

LSelection_RemoveAllObjectsOnLayer

```
LStatus LSelection_RemoveAllObjectsOnLayer(LLayer layer);
```

Description

Removes all objects on ***layer*** from the selection list.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

layer	Layer whose objects are to be removed from the selection.

See Also

LStatus (page 4-512), **LLayer** (page 4-542), **Selection Functions** (page 4-320)

LSelection_AddAllObjectsInRect

```
LStatus LSelection_AddAllObjectsInRect(LRect *box);
```

Description

Adds all objects in rectangle ***box*** to the selection list.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

box	Pointer to an LRect that specifies the coordinates of the box.

See Also

LStatus (page 4-512), **LRect** (page 4-528), **Selection Functions** (page 4-320)

LSelection_RemoveAllObjectsInRect

```
LStatus LSelection_RemoveAllObjectsInRect(LRect *box);
```

Description

Removes all objects in rectangle ***box*** from the selection list.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

box	Pointer to an LRect that specifies the coordinates of the box.

See Also

LStatus (page 4-512), **LRect** (page 4-528), **Selection Functions** (page 4-320)

LSelection_GetList

```
LSelection LSelection_GetList(void);
```

Description

Gets the pointer to the first element in the selection list.

Return Values

Pointer to the head of the selection list if successful; NULL otherwise.

See Also

LSelection (page 4-540), **Selection Functions** (page 4-320)

LSelection_GetNext

```
LSelection LSelection_GetNext(LSelection selection);
```

Description

Gets a pointer to the next element in the selection list.

Return Values

Pointer to the next element in the selection list if successful; NULL otherwise.

Parameters

selection	Pointer to a selection element.

See Also

LSelection (page 4-540), **Selection Functions** (page 4-320)

LSelection_GetLayer

```
LLayer LSelection_GetLayer (LSelection selection);
```

Description

Gets the layer of a given selection element.

Return Values

Pointer to the layer if successful; NULL otherwise.

Parameters

selection	Pointer to the selection element.

See Also

LLayer (page 4-542), **LSelection** (page 4-540), **Selection Functions** (page 4-320)

LSelection_ChangeLayer

```
LStatus LSelection_ChangeLayer(LLayer srcLayer, LLayer
    dstLayer);
```

Description

Changes the layer of all objects in the selection on ***srcLayer*** to ***dstLayer*.**

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

srcLayer	Source layer.
dstLayer	Destination layer.

See Also

LStatus (page 4-512), **LLayer** (page 4-542), **Selection Functions** (page 4-320)

LSelection_Move

```
LStatus LSelection_Move(long dx, long dy);
```

Description

Moves the selection by displacements ***dx*** (in the x-direction) and ***dy*** (in the y-direction).

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

dx	Displacement value in x-direction.
dy	Displacement value in y-direction.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_Duplicate

```
LStatus LSelection_Duplicate(void);
```

Description

Duplicates the contents of the current selection. The duplicate is placed next to the original.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_Group

```
LStatus LSelection_Group(char *group_cell_name );
```

Description

Creates a new cell containing the currently selected objects and an instance of the new cell in the current cell.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

group_cell_name	Name of the group cell.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_UnGroup

```
LStatus LSelection_UnGroup(void);
```

Description

Ungroups (flattens one level of hierarchy of) the curent selection.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_Flatten

```
LStatus LSelection_Flatten(void);
```

Description

Flattens all levels of hierarchy (down to basic objects) in the current selection.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_Merge

```
LStatus LSelection_Merge(void);
```

Description

Merges all objects in the current selection which share the same layer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_FlipHorizontal

```
LStatus LSelection_FlipHorizontal(void);
```

Description

Flips all objects in current selection horizontally (left/right).

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_FlipVertical

```
LStatus LSelection_FlipVertical(void);
```

Description

Flips all objects in current selection vertically (up/down).

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

LSelection_SliceHorizontal

```
LStatus LSelection_SliceHorizontal(LPoint *point);
```

Description

LSelection_SliceHorizontal slices horizontally all objects in current selection at the specified point.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

point	Pointer to an LPoint structure that contains an (x,y) point on the horizontal slice line.

See Also

LStatus (page 4-512), **LPoint** (page 4-527), **Selection Functions** (page 4-320)

LSelection_SliceVertical

```
LStatus LSelection_SliceVertical(LPoint *point);
```

Description

Slices vertically all objects in current selection at the specified point.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

point	Pointer to an LPoint structure that contains an (x,y) point on the vertical slice line.

See Also

LStatus (page 4-512), **LPoint** (page 4-527), **Selection Functions** (page 4-320)

LSelection_Rotate

```
LStatus LSelection_Rotate(void);
```

Description

Rotates all objects in current selection counterclockwise.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **Selection Functions** (page 4-320)

Layer Functions

There are four categories of UPI layer functions.

Design Layer Functions (page 4-352) allow the user to assign resistance or capacitance values or wire setup information to a layer. These functions also allow the user to make a layer hidden or visible in a display.

Generated Layer Functions (page 4-376) allow the user to manipulate layers generated from other layers according to equations defined by the user.

Special Layer Functions (page 4-398) allow the user to control the appearance of L-Edit constructs such as the grid, origin, and drag boxes.

Rendering Functions (page 4-402) are used to edit the information that defines how L-Edit displays a design layer.

Design Layer Functions

L-Edit supports an unlimited number of design layers. Layers may be assigned a capacitance value, a resistance value, and wire setup information. Layers may also be hidden or visible in the display.

Design layer functions allow the user to create and manipulate design layers in a file.

LLayer_New

```
LStatus LLayer_New(LFile file, LLayer layer, char *name);
```

Description

Creates a new layer in the specified file. All layers in a file are arranged in a list. The newly created layer is added to the layer list directly after the specified layer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	File where new layer is to be added.
layer	Layer after which the new layer is to be added.
name	Name of the new layer.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **LLayer** (page 4-542), **Design Layer Functions** (page 4-352)

LLayer_Delete

```
LStatus LLayer_Delete(LFile file, LLayer layer);
```

Description

Deletes the specified layer from the specified file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	File containing the layer.
layer	Layer to be deleted.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **LLayer** (page 4-542), **Design Layer Functions** (page 4-352)

LLayer_Find

```
LLayer LLayer_Find(LFile file, const char* name);
```

Description

Searches the layer list of the specified file for a layer with the given name.

Return Values

Pointer to the matching layer if successful; NULL otherwise.

Parameters

file	File whose layer list is to be searched.
name	Layer name to look for.

Example

The following example searches for the layer named Metal1 in the file layout.tdb:

```
/*This example  opens layout.tdb and checks to see if it
   contains a layer called Metal1*/

LFile file;
LLayer layer;

/*Open layout.tdb file*/
file = LFile_Open(“layout”, TdbFile);

/*Search for layer Metal1 in this file*/
layer = LLayer_Find(file, “Metal1”);

if ( layer == NULL ){
       /*Layer not found*/
}
else {
       /*layer found*/
}
```

The above example will return an opaque pointer layer to the layer Metal1. It thus saves the time required to write code for browsing through all the layers using LLayer_GetList and LLayer_GetNext.

See Also

LLayer (page 4-542), **LFile** (page 4-515), **Design Layer Functions** (page 4-352)

LLayer_GetList

```
LLayer LLayer_GetList(LFile file);
```

Description

Gets a pointer to the first layer in the layer list of file.

Return Values

Pointer to the head of the layer list if successful; NULL otherwise.

Parameters

file	Specified file.

See Also

LLayer (page 4-542), **LFile** (page 4-515), **Design Layer Functions** (page 4-352)

LLayer_GetNext

```
LLayer LLayer_GetNext(LLayer layer);
```

Description

Gets a pointer to the layer immediately following a given layer in the layer list.

Return Values

Pointer to the next element in the layer list if successful; NULL otherwise.

Parameters

layer	Specified layer.

See Also

LLayer (page 4-542), **Design Layer Functions** (page 4-352)

LLayer_PrecedingLayer

```
LLayer LLayer_PrecedingLayer(LFile pFile, char* szName,
    LLayer pReserved)
```

Description

Finds the layer that precedes the specified layer's name. Argument *pReserved* should set to NULL when calling this function.

Return Values

Pointer to the preceding layer if successful; NULL otherwise.

Parameters

pFile	File whose layers are to be searched.
szName	Name of the layer whose preceding layer is required.
pReserved	Reserved variable. Set to NULL when calling this function.

See Also

LLayer (page 4-542), **LFile** (page 4-515), **Design Layer Functions** (page 4-352)

LLayer_PrecedingLayerEx99

```
LLayer LLayer_PrecedingLayerEx99(LFile pFile, LLayer pLayer)
```

Description

Finds the layer that precedes the specified layer.

Return Values

Pointer to the preceding layer if successful; NULL otherwise.

Parameters

pFile	File whose layers are to be searched.
pLayer	Specified layer.

See Also

LLayer (page 4-542), **LFile** (page 4-515), **Design Layer Functions** (page 4-352)

LLayer_GetName

```
char* LLayer_GetName(LLayer layer, char* name, const int
    maxlen);
```

Description

Gets the name of a layer and fills the buffer ***name*** with the name of the layer. If the layer name is longer than ***maxlen***, the extra characters are ignored.

Return Values

Pointer to the layer name buffer if successful; NULL otherwise.

Parameters

layer	Layer whose name is to be retrieved.
name	String (buffer) containing the name of the layer.
maxlen	Maximum length allowed for **name.**

See Also

LLayer (page 4-542), **Design Layer Functions** (page 4-352)

LLayer_SetName

```
LStatus LLayer_SetName(LLayer layer, const char *name);
```

Description

Changes the name of a layer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

layer	Layer whose name is to be changed.
name	String (buffer) that contains the new name of the layer.

See Also

LStatus (page 4-512), **LLayer** (page 4-542), **Design Layer Functions** (page 4-352)

LLayer_GetParameters

```
LLayerParam *LLayer_GetParameters(LLayer layer, LLayerParam
    *param);
```

Description

Gets the properties of ***layer***.

Return Values

Pointer to the layer parameter structure if successful; NULL otherwise.

Parameters

layer	Specified layer.
param	Pointer to a layer parameter structure. This structure will be used for returning data.

See Also

LLayer (page 4-542), **LLayerParam** (page 4-545), **Design Layer Functions** (page 4-352)

LLayer_SetParameters

```
LStatus LLayer_SetParameters(LLayer layer, LLayerParam
    *param);
```

Description

Sets the parameters of ***layer*.**

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

layer	Specified layer.
param	Pointer to a layer parameter structure containing the new layer parameters.

See Also

LStatus (page 4-512), **LLayer** (page 4-542), **LLayerParam** (page 4-545), **Design Layer Functions** (page 4-352)

LLayer_GetCap

```
double LLayer_GetCap(LLayer layer);
```

Description

Gets the capacitance of ***layer***.

Return Values

The capacitance value of ***layer***. It returns -1 on error.

Parameters

layer	Specified layer.

See Also

LLayer (page 4-542), **Design Layer Functions** (page 4-352)

LLayer_SetCap

```
LStatus LLayer_SetCap(LLayer layer, double cap);
```

Description

Changes the capacitance value of ***layer***.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

layer	Specified layer.
cap	Capacitance value.

See Also

LStatus (page 4-512), **LLayer** (page 4-542), **Design Layer Functions** (page 4-352)

LLayer_GetRho

```
double LLayer_GetRho(LLayer layer);
```

Description

Gets the resistance value of ***layer*****.**

Return Values

The resistance value of ***layer***. It returns -1 on error.

Parameters

layer	Specified layer.

See Also

LLayer (page 4-542), **Design Layer Functions** (page 4-352)

LLayer_SetRho

```
LStatus LLayer_SetRho(LLayer layer, double rho);
```

Description

Changes the resistance of ***layer***.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

layer	Specified layer.
rho	New resistance value.

See Also

LStatus (page 4-512), **LLayer** (page 4-542), **Design Layer Functions** (page 4-352)

LLayer_GetCurrent

```
LLayer LLayer_GetCurrent(LFile file);
```

Description

Gets a pointer to the current layer in the specified file.

Return Values

Pointer to the current layer if successful; NULL otherwise.

Parameters

file	Specified file.

See Also

LLayer (page 4-542), **LFile** (page 4-515), **Design Layer Functions** (page 4-352)

LLayer_SetCurrent

```
LStatus LLayer_SetCurrent(LFile file, LLayer layer);
```

Description

Sets the current layer in the specified file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
layer	Specified layer.

See Also

LStatus (page 4-512), **LLayer** (page 4-542), **LFile** (page 4-515), **Design Layer Functions** (page 4-352)

Generated Layer Functions

Generated layers are generated from other layers according to an equation defined by the user. Generated layer definitions are assigned to each layer in the layer list and can be directly accessed and modified with the function calls below.

LLayer_GetDerivedList

```
LLayer LLayer_GetDerivedList(LFile file);
```

Description

Gets the list of generated layers in a file.

Return Values

Pointer to the head of the generated layer list if successful; NULL otherwise.

Parameters

file	Specified file.

See Also

LLayer (page 4-542), **LFile** (page 4-515), **Generated Layer Functions** (page 4-376)

LDerivationType

```
typedef enum
{
LDOT_Bool=0,
LDOT_Area=1,
      LDOT_Select=2,
}
LDerivationType;
```

Description

[ADD]

Return Values

Pointer to the head of the generated layer list if successful; NULL otherwise.

Parameters

file	Specified file.

Structure

```
typedef structure {
       short LRed
       short LBlue
       short LGreen
} LColor
```

See Also

LLayer (page 4-542), **LFile** (page 4-515), **Generated Layer Functions** (page 4-376)

LLayer_GetDerivedNext

```
LLayer LLayer_GetDerivedNext(LLayer layer);
```

Description

Gets the generated layer following a given generated layer.

Return Values

Pointer to the next element in the generated layer list if successful; NULL otherwise.

Parameters

layer	Specified layer.

See Also

LLayer (page 4-542), **Generated Layer Functions** (page 4-376)

LLayer_IsDerived

```
int LLayer_IsDerived(LLayer layer);
```

Description

Checks whether a layer is a generated layer or not.

Return Values

A nonzero value if the layer is a generated layer, or zero if the layer is not a generated layer.

Parameters

layer	Specified layer.

See Also

LLayer (page 4-542), **Generated Layer Functions** (page 4-376)

LLayer_EnableAllDerived

```
LStatus LLayer_EnableAllDerived(LFile file);
```

Description

Enables the generated layer definition for all layers in a file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **Generated Layer Functions** (page 4-376)

LLayer_DisableAllDerived

```
LStatus LLayer_DisableAllDerived(LFile file);
```

Description

Disables the generated layer definition for all layers in the specified file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **Generated Layer Functions** (page 4-376)

LLayer_GetDerivedParameters

```
LDerivedLayerParam *LLayer_GetDerivedParameters(LLayer
    layer, LDerivedLayerParam *param);
```

Description

Gets the parameters of a generated layer.

Return Values

Pointer to the generated layer parameter structure if successful; NULL otherwise.

Note: Note that this function is superseded by **LLayer_GetDerivedParametersEx00** (page 4-387).

Parameters

layer	Specified layer.
param	Pointer to a generated layer parameter structure.

See Also

LDerivedLayerParam (page 4-547), **LLayer** (page 4-542), **Generated Layer Functions** (page 4-376), **LLayer_GetDerivedParametersEx00** (page 4-387).

LLayer_GetDerivedParametersEx00

```
LDerivedLayerParamEx00*
   LLayer_GetDerivedParametersEx00(LLayer  layer,
   LDerivedLayerParamEx00 *param);
```

Description

Gets derivation parameters of the specified layer into the LDerivedLayerParamEx00 structure pointed to by the specified parameter value.

See Also

LDerivedLayerParamEx00 (page 4-585), **LDerivationType** (page 4-586), **LDerivedLayerBoolOperation** (page 4-588), **LDerivedLayerSelectOperation** (page 4-589), **LDerivedLayerAreaOperation** (page 4-591).

LLayer_SetDerivedParameters

```
LStatus LLayer_SetDerivedParameters(LFile file, LLayer
    layer, LDerivedLayerParam *param);
```

Description

Sets the generated layer parameters of a layer in a given file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Note: Note that this function is superseded by **LLayer_SetDerivedParametersEx00** (page 4-390)

Parameters

file	File containing the specified layer.
layer	Specified layer.
param	Pointer to a generated layer parameters structure that contains the new parameter values.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **LLayer** (page 4-542), **LDerivedLayerParam** (page 4-547), **Generated Layer Functions** (page 4-376)

LLayer_SetDerivedParametersEx00

```
LStatus LLayer_SetDerivedParametersEx00(LFile file, LLayer
    layer, LDerivedLayerParamEx00 *param)
```

Description

Sets derivation parameters of the specified layer to the values specified in the LDerivedLayerParamEx00 structure pointed to by the specified parameter value.

See Also

LDerivedLayerParamEx00 (page 4-585), **LDerivationType** (page 4-586), **LDerivedLayerBoolOperation** (page 4-588), **LDerivedLayerSelectOperation** (page 4-589), **LDerivedLayerAreaOperation** (page 4-591).

LLayer_DestroyDerivedParameter

LStatus *LLayer_DestroyDerivedParameter* (**LDerivedLayerParam*** *pDerivedLayerParam***)**

Description

Frees the memory associated with the derived layer parameter structure that was allocated by L-Edit during an **LLayer_GetDerivedParameters** call. Do not call **Layer_DestroyDerivedParameter** if **LDrcRule_GetParameter** has not been previously called with **pDesignRuleParam**.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value with the possible values **LBadParameters** - *pDerivedLayerParam* is NULL.

Parameters

pDerivedLayerParam	Pointer to a derived layer parameter structure.

Example

```
LFile pTDBFile = LFile_GetVisible();
if(Assigned(pTDBFile))
{
       LLayer pLayer = LLayer_Find(pTDBFile,
   "PolyCnt_And_NotPoly");
       if(Assigned(pLayer))
       {
              LDerivedLayerParam pDerivedLayerParam;

   if(Assigned(LLayer_GetDerivedParameters(pLayer,
   &pDerivedLayerParam)))
              {
                     long lGrow =
   pDerivedLayerParam.layer1_grow_amount;

                     // More Processing
                     // ...


   LLayer_DestroyDerivedParameter(&pDerivedLayerParam);
```

```
                    }
                } // endif(Assigned(pLayer))
            } // endif(Assigned(pTDBFile))
```

Version

Available in L-Edit 8.2 and later versions.

See Also

Generated Layer Functions (page 4-376), **LDerivedLayerParamEx00** (page 4-585), **LStatus** (page 4-512)

LCell_GenerateLayers

```
LStatus LCell_GenerateLayers(LCell cell, int bin_size);
```

Description

Generates layers in the specified cell.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Specified cell.
bin_size	Bin size.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Generated Layer Functions** (page 4-376)

LCell_GenerateLayersEx99

```
LStatus LCell_GenerateLayersEx99(LCell pCell, int iBinSize,
    LLayer pLayer)
```

Description

Generates the layer or layers in the specified cell. To generate all layers, set ***pLayer*** to NULL.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

pCell	Cell to generate the layers in.
iBinSize	Bin size for generating layers.
pLayer	Layer to generate. If pLayer is NULL then all layers are generated.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Generated Layer Functions** (page 4-376)

LCell_ClearGenerateLayers

```
LStatus LCell_ClearGenerateLayers(LCell cell);
```

Description

Clears all generated layers from a cell.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Specified cell.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Generated Layer Functions** (page 4-376)

Special Layer Functions

Each file employs seven layers for specific purposes involved with graphic display. Each special layer is assigned a layer selected from the file's layer list. (For more information on special layers, see Special Layers on page 1-182.)

LLayer_GetSpecial (page 4-399) **LLayer_SetSpecial** (page 4-400)

LLayer_GetSpecial

```
LLayer LLayer_GetSpecial(LFile file, LSpecialLayer
    specialLayer);
```

Description

Gets a particular type of special layer of a file.

Return Values

Pointer to the special layer if successful; NULL otherwise.

Parameters

file	Specified file.
specialLayer	Type of special layer.

See Also

LLayer (page 4-542), **LFile** (page 4-515), **LSpecialLayer** (page 4-548), **Special Layer Functions** (page 4-398)

LLayer_SetSpecial

```
LStatus LLayer_SetSpecial(LFile file, LSpecialLayer
    specialLayer, LLayer layer);
```

Description

Sets a special layer of a given file to a particular type.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
specialLayer	Type of special layer.
layer	New special layer.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **LSpecialLayer** (page 4-548), **LLayer** (page 4-542), **Special Layer Functions** (page 4-398)

Rendering Functions

The rendering pass list contains the layer rendering information that L-Edit uses to display a layer. Information found in this list include stipple pattern, color, pass type and write mode (set or clear). (For additional information, see Rendering Layer Parameters on page 1-161.)

LPass_New

```
LPass LPass_New(LPass precedingPass, LPass pass);
```

Description

Adds a new pass after the preceding pass.

Return Values

Pointer to the newly added pass if successful; NULL otherwise.

Parameters

precedingPass	Preceding pass. The new pass will be added after ***precedingPass.***
pass	Pass to be deleted.

See Also

LPass (page 4-549), **Rendering Functions** (page 4-402)

LPass_GetList

```
LPass LPass_GetList(LLayer layer, LPassType passType);
```

Description

Gets a list of particular pass types associated with a layer.

Return Values

Pointer to the head of the pass list if successful; NULL otherwise.

Parameters

layer	Specified layer.
passType	Type of pass.

See Also

LPass (page 4-549), **LLayer** (page 4-542), **LPassType** (page 4-551), **Rendering Functions** (page 4-402)

LPass_GetNext

```
LPass LPass_GetNext(LPass pass);
```

Description

Gets the next pass in the pass list.

Return Values

Pointer to the next element in the pass list if successful; NULL otherwise.

Parameters

pass	Specified pass.

See Also

LPass (page 4-549), **Rendering Functions** (page 4-402)

LPass_GetParameters

```
LPassParam *LPass_GetParameters(LPass pass, LPassParam
    *param);
```

Description

Gets the parameters of a pass.

Return Values

Pointer to the pass parameters structure if successful; NULL otherwise.

Parameters

pass	Specified pass.
param	Pointer to a pass structure. This buffer will be filled with the results.

See Also

LPassParam (page 4-552), **LPass** (page 4-549), **Rendering Functions** (page 4-402)

LPass_SetParameters

```
LStatus LPass_SetParameters(LPass pass, LPassParam *param);
```

Description

Sets the parameters of a pass.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

pass	Specified pass.
param	Pointer to a pass structure. This buffer contains the new parameter values.

See Also

LStatus (page 4-512), **LPassParam** (page 4-552), **LPass** (page 4-549), **Rendering Functions** (page 4-402)

LLayer_GetRenderingAttribute

```
LStatus LLayer_GetRenderingAttribute(LLayer layer,
    LRenderingAttributeIndex index, LLRenderingAttribute
    pRA);
```

Description

This function returns a rendering attribute.

Return Values

LStatusOK if successful or LBadParameters if not.

Parameters

type	*variable*	*meaning*
LLayer	layer	The layer
LRenderingAttributeIndex	index	The number of the rendering attribute to get.
LRenderingAttribute	pRA	A pointer to **LRendering Attribute** structure.

Example

```
unsigned int get_port_text_pass(LLayer layer)
{
        LRenderingAttribute ra;
        LLayer_GetRenderingAttribute(layer, raiPortText,
    &ra);
        return ra.mPass;
}
```

See Also

LRenderingAttributeIndex (page 4-593), **LRenderingAttribute** (page 4-594), **LLayer_GetRenderingObjectName** (page 4-415), **LLayer_SetRenderingAttribute** (page 4-412).

LLayer_SetRenderingAttribute

```
LStatus LLayer_SetRenderingAttribute(LLayer layer,
    LRenderingAttributeIndex index, LLRenderingAttribute
    pRA);
```

Description

This function sets a rendering attribute.

Return Values

LStatusOK if successful or LBadParameters if not.

Parameters

type	*variable*	*meaning*
LLayer	layer	The layer
LRenderingAttributeIndex	index	The number of the rendering attribute to set.
LRenderingAttribute	pRA	A pointer to **LRendering Attribute** structure.

Example

```
void make_outline_thin(LLayer layer)
{
        unsigned int n;
        LRenderingAttribute        ra;

        for(n=raiFirstRenderingAttribute;
    n<=raiLastRenderingAttribute; n++)
        {
                LLayer_GetRenderingAttribute(layer, n; &ra);
                ra.mOutlineThicknessUnits = utPixels;
                ra.mOutlineThickness = 1;
                LLayer_SetRenderingAttribute(layer, n; &ra);
        }
}
```

See Also

LRenderingAttributeIndex (page 4-593), **LRenderingAttribute** (page 4-594), **LLayer_GetRenderingObjectName** (page 4-415), **Special Layer Functions** (page 4-398).

LLayer_GetRenderingObjectName

```
LStatus LLayer_GetRenderingObjectName(LLayer layer,
    LRenderingAttributeIndex index, char *nameBuf, int
    nameBufSize);
```

Description

This function is mainly for debugging purposes. It returns the name of a rendering attribute.

Return Values

LStatusOK if successful or LBadParameters if not. The possible values of nameBuf after a successful call are: "Object," "PortBox," "PortText," "WireCenterline," "SelectedObject," "SelectedPortBox," "SelectedPortText," and "SelectedWireCenterline."

Parameters

type	*variable*	*meaning*
LLayer	layer	The layer
LRenderingAttributeIndex	index	The number of the rendering attribute to get.
LRenderingAttribute	pRA	A pointer to **LRendering Attribute** structure.

Example

```
void message_outline_thickness(LLayer layer)
{
      unsigned int n;
      LRenderingAttribute ra;
      char   nameBuf[64];
      char   msgBuf[NumberOfRenderingAttributes][128];

for(n=raiFirstRenderingAttribute;
   n<=raiLastRenderingAttribute; n++)
{
      LLayer_GetRenderingObjectName(layer, n, nameBuf,
   sizeof(nameBuf));
      LLayer_GetRenderingAttribute(layer, n; &ra);
```

```
        sprintf(msgbuf[n], "Outline thickness for %s is %u
    %s",
                    nameBuf,
                    ra.mOutlineThickness,
                    (ra.mOutlineThicknessUnits==utPixels)?
    "Pixels" : "LU");
        }

        LDialog_MultiLineMsgBox(msgBuf,
    NumberOfRenderingAttributes);
}
```

See Also

LRenderingAttribute (page 4-594), **LLayer_GetRenderingObjectName** (page 4-415), **LLayer_SetRenderingAttribute** (page 4-412).

Technology Setup Functions

Technology functions allow the user to manipulate the technology of a design file. Specifically, these functions allow the user to get, set, or change the technology setup or individual technology parameters.

LFile_GetTechnology

```
LTechnology LFile_GetTechnology(LFile file);
```

Description

Gets the technology setup of a file.

Return Values

The LTechnology structure filled with the values of the current technology setup.

Parameters

file	Specified file.

See Also

LFile (page 4-515), **Ltech_unit_type** (page 4-568), **Technology Setup Functions** (page 4-418)

LFile_SetTechnology

```
LStatus LFile_SetTechnology(LFile file, LTechnology
    *technology);
```

Description

Sets the technology setup of a file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
technology	Pointer to an LTechnology structure that contains the new technology setup.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **Ltech_unit_type** (page 4-568), **Technology Setup Functions** (page 4-418)

LFile_SetTechnologyName

```
char* LFile_SetTechnologyName(LFile file, char* name);
```

Description

Sets the technology name of file.

Return Values

Pointer to the technology name buffer if successful; NULL otherwise.

Parameters

file	Specified file.
name	New technology name.

See Also

LFile (page 4-515), **Technology Setup Functions** (page 4-418)

LFile_SetTechnologyUnitNum

```
LStatus LFile_SetTechnologyUnitNum(LFile file, long num);
```

Description

Sets the numerator of the technology unit mapping fraction in file to ***num*.**

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
num	Numerator value.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **Technology Setup Functions** (page 4-418)

LFile_SetTechnologyUnitDenom

```
LStatus LFile_SetTechnologyUnitDenom(LFile file, long
    denom);
```

Description

Sets the denominator of the technology unit mapping fraction in file to *denom*.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
denom	Denominator value.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **Technology Setup Functions** (page 4-418)

LFile_SetTechnologyLambdaNum

```
LStatus LFile_SetTechnologyLambdaNum(LFile file, long num);
```

Description

Sets the numerator of the technology lambda mapping fraction in file to ***num***.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
num	Numerator value.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **Technology Setup Functions** (page 4-418)

LFile_SetTechnologyLambdaDenom

```
LStatus LFile_SetTechnologyLambdaDenom(LFile file, long
    denom);
```

Description

Sets the denominator of the technology lambda mapping fraction in file to ***denom***.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
denom	Denominator value.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **Technology Setup Functions** (page 4-418)

Color Palette Functions

L-Edit color palette can contain 16, 32, 64, 128, or 256 colors. (For further information, see Color Parameters on page 1-107.)

These functions allow the user to manipulate the color palette of a layout.

LFile_GetColorPalette (page 4-429)

LFile_GetColorPaletteNumColors (page 4-431)

LFile_GetColorPaletteSortBy (page 4-433)

LFile_SetColorPalette (page 4-435)

LFile_SetColorPaletteNumColors (page 4-437)

LFile_SetColorPaletteSortBy (page 4-439)

LFile_GetColorPalette

```
LStatus LEDITAPI LFile_GetColorPalette(LFile file, LColor
    *pColor, int index);
```

Description

Gets a color from the palette.

Return Values

Returns LStatusOK if successful or LBadParameter if an error occurred.

Parameters

file	Current file.
pColor	Pointer to LColor.
index	Index number of the color to get. Must be non-negative and less than the number of colors in the palette.

Structure

```
typedef structure {
      short LRed
      short LBlue
      short LGreen
} LColor
```

See Also

LFile_SetColorPalette (page 4-435), **LFile_GetColorPaletteNumColors** (page 4-431).

LFile_GetColorPaletteNumColors

```
int LEDITAPI LFile_GetColorPaletteNumColors(LFile file);
```

Description

Gets the number of colors in the palette.

Return Values

Number of colors in the palette. Possible values are:

- 16
- 32
- 64
- 128
- 256

Returns null if there is an error.

Parameters

file	Current file.

See Also

LFile_SetColorPalette (page 4-435), **LFile_SetColorPaletteNumColors** (page 4-437).

LFile_GetColorPaletteSortBy

```
const char *LEDITAPI LFile_GetColorPaletteSortBy(LFile
    file);
```

Description

Sets the name of the palette sort option.

Return Values

The name of the palette sort option. Possible values are:

- “SortByIndex”
- “SortByNumBits”
- “SortByHue”
- “SortByBrightness”

Returns null if an error occurred.

Parameters

file	Current file.

See Also

LFile_SetColorPaletteSortBy (page 4-439).

LFile_SetColorPalette

```
LStatus LEDITAPI LFile_SetColorPalette(LFile file, const
    LColor *pcolor, int index);
```

Description

Sets a color specified by the index in the palette.

Return Values

Returns LStatusOK if successful or LBadParameter if an error occurred.

Parameters

file	Current file.
pcolor	Pointer to LColor.
index	Index of the color to set. Must be non-negative and less than the number of colors in the palette.

Structure

```
typedef structure {
       short LRed
       short LBlue
       short LGreen
} LColor
```

See Also

LColor (page 4-553), **LFile_SetColorPalette** (page 4-435), **LFile_GetColorPaletteNumColors** (page 4-431), Color Parameters on page 1-107.

LFile_SetColorPaletteNumColors

```
LStatus LEDITAPI LFile_SetColorPaletteNumColors(LFile file,
    int numcolors);
```

Description

Sets the number of colors in the palette. This number must be one of the following values:

- 16
- 32
- 64
- 128
- 256

Return Values

Returns LStatusOK if successful or LBadParameter if an error occurred.

Parameters

file	Current file.

See Also

LFile_GetColorPaletteSortBy (page 4-433).

LFile_SetColorPaletteSortBy

```
LStatus LEDITAPI LFile_SetColorPaletteSortBy(LFile file,
    const char *sortby);
```

Description

Sets a name of the palette sort option. Possible values are:

- “SortByIndex” (See Color Parameters on page 1-107 for a complete description of the color index.)
- “SortByNumBits”
- “SortByHue”
- “SortByBrightness”

Return Values

Returns LStatusOK if successful or LBadParameter if an error occurred.

Parameters

file	Current file.

See Also

LFile_GetColorPaletteNumColors (page 4-431).

Import/Export Functions

L-Edit can import a layout from GDS II and CIF files or export a layout to GDS II or CIF files.

CIF Setup Functions (page 4-442) allow the user to set CIF import/export parameters.

GDS II Setup Functions (page 4-447) allow the user to set GDS II import/export parameters.

CIF Setup Functions

LFile_GetCIFParameters

```
LCIFParam *LFile_GetCIFParameters(LFile file, LCIFParam
    *cifparam);
```

Description

Gets the CIF parameters of a file.

Return Values

Pointer to the CIF parameters structure if successful; NULL otherwise.

Parameters

file	Specified file.
cifparam	Pointer to a structure that will contain CIF parameters.

See Also

LCIFParam (page 4-560), **LFile** (page 4-515), **CIF Setup Functions** (page 4-442)

LFile_SetCIFParameters

```
LStatus LFile_SetCIFParameters(LFile file, LCIFParam
    *cifparam);
```

Description

Sets the CIF parameters of a file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
cifparam	Pointer to a structure that contains the new CIF parameter values.

See Also

LStatus (page 4-512), **LCIFParam** (page 4-560), **LFile** (page 4-515), **CIF Setup Functions** (page 4-442)

GDS II Setup Functions

LFile_GetGDSParameters (page 4-448)

LFile_SetGDSParameters (page 4-450)

LFile_GetGDSParameters

```
LGDSParam *LFile_GetGDSParameters(LFile file, LGDSParam
    *gdsparam);
```

Description

Gets GDS II parameters of a file.

Return Values

Pointer to theGDS II parameters structure if successful; NULL otherwise.

Parameters

file	Specified file.
gdsparam	Pointer to a structure that will contain GDS II parameters.

See Also

LGDSParam (page 4-561), **LFile** (page 4-515), **GDS II Setup Functions** (page 4-447)

LFile_SetGDSParameters

```
LStatus LFile_SetGDSParameters(LFile file, LGDSParam
    *gdsparam);
```

Description

Sets the current GDS II parameters of a file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
gdsparam	Pointer to a structure that contains GDS II parameters.

See Also

LStatus (page 4-512), **LGDSParam** (page 4-561), **LFile** (page 4-515), **GDS II Setup Functions** (page 4-447)

DRC Functions

DRC functions allow the user to manipulate the design rules of a layout file and run a design rule check.

LDrcRule_Add

```
LDrcRule LDrcRule_Add(LFile file, LDrcRule preceding_rule,
    LDesignRuleParam *param);
```

Description

Adds a new design rule to the file. The newly added design rule will be added after the specified ***preceding_rule*** and will have the specified parameters.

Return Values

Pointer to the newly added DRC rule if successful; NULL otherwise.

Parameters

file	Specified file.
preceding_rule	New rule will be added after this rule.
param	Pointer to a design rule parameter structure that specifes the details of the new rule.

See Also

LDrcRule (page 4-555), **LFile** (page 4-515), **LDesignRuleParam** (page 4-557), **DRC Functions** (page 4-452)

LDrcRule_Delete

```
LStatus LDrcRule_Delete(LFile file, LDrcRule rule);
```

Description

Deletes a design rule from a file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
rule	Rule to be deleted.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **LDrcRule** (page 4-555), **DRC Functions** (page 4-452)

LDrcRule_Find

```
LDrcRule LDrcRule_Find(LFile file, LDrcRuleType rule_type,
    char *layer1, char *layer2);
```

Description

Searches for a specific design rule involving two given layers.

Return Values

Pointer to the DRC rule if successful; NULL otherwise.

Parameters

file	Specified file.
rule_type	Type of DRC rule.
layer1	Source layer 1.
layer2	Source layer 2.

See Also

LDrcRule (page 4-555), **LFile** (page 4-515), **LDrcRuleType** (page 4-556), **DRC Functions** (page 4-452)

LDrcRule_GetList

```
LDrcRule LDrcRule_GetList(LFile file);
```

Description

Gets a list of DRC rules in a file.

Return Values

Pointer to the head of the DRC rule list if successful; NULL otherwise.

Parameters

file	Specified file.

See Also

LDrcRule (page 4-555), **LFile** (page 4-515), **DRC Functions** (page 4-452)

LDrcRule_GetNext

```
LDrcRule LDrcRule_GetNext(LDrcRule rule);
```

Description

Gets the design rule that follows a given design rule.

Return Values

Pointer to the next element in the DRC rule list if successful; NULL otherwise.

Parameters

rule	Specified design rule.

See Also

LDrcRule (page 4-555), **DRC Functions** (page 4-452)

LDrcRule_SetRuleSet

```
LStatus LDrcRule_SetRuleSet(LFile file, char *rule_set);
```

Description

Sets the name of the design rule set in a file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
rule_set	Name of the rule set.

See Also

LStatus (page 4-512), **LDrcRule** (page 4-555), **LFile** (page 4-515), **DRC Functions** (page 4-452)

LDrcRule_SetTolerance

```
LStatus LDrcRule_SetTolerance(LFile file, long tolerance);
```

Description

Sets the tolerance of the design rule set in a file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
tolerance	Tolerance of the design rule set.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **DRC Functions** (page 4-452)

LDrcRule_GetParameters

```
LDesignRuleParam *LDrcRule_GetParameters(LDrcRule rule,
   LDesignRuleParam *param);
```

Description

Gets the parameters of a DRC rule.

Return Values

Pointer to the DRC rule parameters structure if successful; NULL otherwise.

Parameters

rule	Specified design rule.
param	Pointer to a structure that will contain the parameters.

See Also

LDesignRuleParam (page 4-557), **LDrcRule** (page 4-555), **DRC Functions** (page 4-452)

LDrcRule_SetParameters

```
LStatus LDrcRule_SetParameters(LFile file, LDrcRule rule,
   LDesignRuleParam *param);
```

Description

Sets the parameters of a DRC rule.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file	Specified file.
rule	Specified design rule.
param	Pointer to a structure that contains the design rule parameters.

See Also

LStatus (page 4-512), **LFile** (page 4-515), **LDrcRule** (page 4-555), **LDesignRuleParam** (page 4-557), **DRC Functions** (page 4-452)

LDRC_Run

```
void LDRC_Run(LCell cell, LRect* onArea, char* errfile,int
    writeErrorPorts, int writeErrorObjects);
```

Description

Runs DRC on the specified area of a cell.

Parameters

cell	Cell on which DRC is to be run.
onArea	Pointer to a LRect structure that specifies a rectangle where DRC will be run.
errfile	Name of the error file.
writeErrorPorts	If 1, error ports will be drawn.
writeErrorObjects	If 1, error objects will be written to the output file.

See Also

LCell (page 4-521), **LRect** (page 4-528), **DRC Functions** (page 4-452)

Extract Functions

These functions are used for netlist extraction.

LExtract_Run (page 4-469)

LProp_AddExtractProp (page 4-471)

LExtract_Run_Ex98 (page 4-473)

LExtract_GetOptions_Ex98 (page 4-474)

LExtract_Run

```
LStatus LExtract_Run(LCell cell, char *extDefFile, char
    *spiceOutFile, int writeNodeName, int
    writeNodeCapacitance);
```

Description

Runs L-Edit/Extract on a given cell.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Cell that needs to be extracted.
ExtDefFile	Name of the extract definition file.
spiceOutFile	Name of the SPICE output file.
writeNodeName	If 1, the SPICE node names are written to the output file.
writeNodeCapacitance	If 1, node capacitances are written to the output file.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **Extract Functions** (page 4-468)

LProp_AddExtractProp

```
LStatus LProp_AddExtractProp( LCell Cell_New, char
    *propName, LPropType type, LPropVal value, LPropCount
    count, LPropAttrib attrib );
```

Description

Adds an extract property with specified characteristics to the specified cell.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell	Cell to attach the extract property to.
propName	Name of property.
type	Property type.
value	Property value.
count	Parameter not available yet. Set to 1.
Attrib	Parameter not available yet. Set to 1.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LPropType** (page 4-573), **LPropVal** (page 4-577), **LPropCount** (page 4-574), **LPropAttrib** (page 4-575), **LPropItem** (page 4-570), **LProp** (page 4-572), **Extract Functions** (page 4-468)

LExtract_Run_Ex98

```
LStatus LExtract_Run_Ex98(LCell topCell, LEtractOptions
    ExtOptions);
```

Description

Runs Extract on the ***topCell***, using the extract options specified in ***ExtOptions***.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LExtractOptions** (page 4-562)

LExtract_GetOptions_Ex98

```
LStatus LExtract_GetOptions_Ex98(LCell oCell,
   LExtractOptions *ExtOptions);
```

Description

Retrieves the L-Edit/Extract options for the given cell (topcell). The resulting extract options are stored in ExtOptions.

Note: To properly retrieve the **.include** statement, the data member szExtIncludeStmt of the structure ExtOptions must be dynamically allocated to a size big enough to hold the expected **.include** statement. The data member lMaxIncludeStmtLen must be set to the size of the allocated string szExtIncludeStmt. Failure to do so could result in a general protection fault.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

LStatus (page 4-512), **LCell** (page 4-521), **LExtractOptions** (page 4-562)

SPR Functions

The core is the "heart" of the design, where the functional logic is contained. It may be one large block containing all of the logic for the design, or it may be composed of several smaller blocks, which typically each have different functions within the design.

Core functions allow the user to manipulate the core of a layout file. The first function provide a way to check if a core exists. The other functions allow the user to get or set the layer-to-layer capacitance for a design's horizontal or vertical routing layer.

LCore_GetCore

```
LCore LCore_GetCore(LFile file);
```

Description

Gets the core of the specified file.

Return Values

Pointer to the core if successful; NULL otherwise.

Parameters

file	Specified file.

See Also

LCore (page 4-567), **LFile** (page 4-515), **SPR Functions** (page 4-476)

LCore_GetLLHCap

```
double LCore_GetLLHCap(LCore core);
```

Description

Gets the layer-to-layer capacitance for the horizontal routing layer of a core.

Return Values

The capacitance value (in aF/sq. micron), or -1 on error.

Parameters

core	Specified core.

See Also

LCore (page 4-567), **SPR Functions** (page 4-476)

LCore_SetLLHCap

```
LStatus LCore_SetLLHCap(LCore core, double LLHCap);
```

Description

Sets the layer-to-layer capacitance for the horizontal routing layer of a core.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

core	Specified core.
LLHCap	New capacitance value (in aF/sq. micron).

See Also

LStatus (page 4-512), **LCore** (page 4-567), **SPR Functions** (page 4-476)

LCore_GetLLVCap

```
double LCore_GetLLVCap(LCore core);
```

Description

Gets the layer-to-layer capacitance for the vertical routing layer of a core.

Return Values

The capacitance value (in aF/sq. micron), or -1 on error.

Parameters

core	Specified core.

See Also

LCore (page 4-567), **SPR Functions** (page 4-476)

LCore_SetLLVCap

```
LStatus LCore_SetLLVCap(LCore core, double LLVCap);
```

Description

Sets the layer-to-layer capacitance for the vertical routing layer of a core.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

core	Specified core.
LLHCap	New capacitance value (in aF/sq. micron).

See Also

LStatus (page 4-512), **LCore** (page 4-567), **SPR Functions** (page 4-476)

Utility Functions

There are three categories of utility functions.

Point Functions (page 4-483) allow the user to create or transform a point.

Rectangle Functions (page 4-489) allow the user to create or transform a rectangle.

Transformation Functions (page 4-494) allow the user to adjust the translation, orientation, or manipulation of an object.

Point Functions

LPoint_Set

```
LPoint LPoint_Set(LCoord x, LCoord y);
```

Description

Creates an LPoint type from two LCoord types with the values x and y.

Return Values

Returns the newly created LPoint.

Parameters

x	x-coordinate.
y	y-coordinate.

See Also

LPoint (page 4-527), **LCoord** (page 4-526)

LPoint_Add

```
LPoint LPoint_Add(LPoint ptA, LPoint ptB)
```

Description

Adds two points

Return Values

The resultant point.

Parameters

ptA	Point 1.
ptB	Point 2.

See Also

Point Functions (page 4-483)

LPoint_Subtract

```
LPoint LPoint_Subtract(LPoint ptA, LPoint ptB)
```

Description

Subtracts two points

Return Values

The resultant point.

Parameters

ptA	Point 1.
ptB	Point 2.

See Also

Point Functions (page 4-483)

LPoint_Transform

```
LPoint LPoint_Transform(LPoint point, LTransform transform);
```

Description

Applies ***transform*** to a ***point***.

Return Values

Values of a new point. The original point is not modified.

Parameters

point	Specified point.
transform	Specified transformation.

See Also

LPoint (page 4-527), **LTransform** (page 4-532)

LPoint_Transform_Ex99

```
LPoint LPoint_Transform_Ex99(LPoint point, LTransform_Ex99
    transform);
```

Description

Applies ***transform*** to a ***point***.

Return Values

Values of a new point. The original point is not modified.

Parameters

point	Specified point.
transform	Specified transformation.

See Also

LPoint_Transform (page 4-487), **LPoint** (page 4-527), **LTransform** (page 4-532)

Rectangle Functions

LRect_Set

LRect LRect_Set(LCoord *x0*, **LCoord** *y0*, **LCoord** *y0*, **LCoord** *y1***);**

Description

Creates an ***LRect*** type from the specified lower left and upper right coordinates. A rectangle can be defined by specifying its lower left and the upper right corners.

Return Values

Returns the newly created ***LRect***.

Parameter

x0	*x*- coordinate of the lower left point
y0	*y*- coordinate of the lower left point
X1	*x*- coordinate of the upper right point
x0	*y*- coordinate of the upper right point

See Also

LRect (page 4-528), **LCoord** (page 4-526), **Rectangle Functions** (page 4-489)

LRect_Transform

```
LRect LRect_Transform (LRect rect, LTransform transform);
```

Description

Applies ***transform*** to ***rect***.

Return Values

Returns a new transformed rectangle. Original ***rect*** is not modified.

Parameters

rect	Rectangle that needs to be transformed.
transform	Specified transformation.

See Also

LRect_Transform_Ex99 (page 4-493), **LRect** (page 4-528), **LTransform** (page 4-532), **Rectangle Functions** (page 4-489)

LRect_Transform_Ex99

```
LRect LRect_Transform_Ex99 (LRect rect, LTransform_Ex99
    transform);
```

Description

Applies ***transform*** to ***rect***.

Return Values

Returns a new transformed rectangle. Original ***rect*** is not modified.

Parameters

rect	Rectangle that needs to be transformed.
transform	Specified transformation.

See Also

Rectangle Functions (page 4-489), **LRect_Transform** (page 4-492), **LRect** (page 4-528), **LTransform** (page 4-532)

Transformation Functions

LTransform_Set

```
LTransform LTransform_Set(LCoord xtrans, LCoord ytrans,
    LOrientation orient, LMagnification mag);
```

Description

Sets a transformation structure.

Return Values

An **LTransform** structure containing the specified transformation.

Parameters

xtrans	Translation amount in the x-direction.
ytrans	Translation amount in the y-direction.
orient	Orientation.
mag	Magnification.

See Also

LTransform (page 4-532), **LCoord** (page 4-526), **LOrientation** (page 4-529), **LMagnification** (page 4-531), **Transformation Functions** (page 4-494)

LTransform_Set_Ex99

```
LTransform_Ex99 LTransform_Set_Ex99(LCoord xtrans, LCoord
    ytrans, LOrientation_Ex99 orient, LMagnification mag);
```

Description

Sets a transformation structure.

Return Values

An **LTransform_Ex99** structure containing the specified transformation.

Parameters

xtrans	Translation amount in the x-direction.
ytrans	Translation amount in the y-direction.
orient	Orientation as a real number
mag	Magnification.

See Also

LTransform_Set (page 4-495), **LTransform** (page 4-532), **LCoord** (page 4-526), **LOrientation** (page 4-529), **LMagnification** (page 4-531), **Transformation Functions** (page 4-494)

LTransform_Zero

```
LTransform LTransform_Zero(void);
```

Description

Makes an identity transformation.

Return Values

Returns the identity transformation.

See Also

LTransform (page 4-532), **Transformation Functions** (page 4-494)

LTransform_Zero_Ex99

```
LTransform_Ex99 LTransform_Zero_Ex99(void);
```

Description

Makes an identity transformation.

Return Values

Returns the identity transformation.

See Also

LTransform_Zero (page 4-499), **LTransform** (page 4-532), **Transformation Functions** (page 4-494)

LTransform_Add

```
LTransform LTransform_Add(LTransform transform_to_be_added,
    LTransform current_transform);
```

Description

Adds two transformations. A transform is the translation, orientation, or magnification of an object.

Return Values

Returns the sum of ***transform_to_be_added*** and ***current_transform*** as an **LTransform.**

Parameters

transform_to_be_added	Transformation structure 1.
current_transform	Transformation structure 2.

See Also

LTransform (page 4-532), **Transformation Functions** (page 4-494)

LTransform_Add_Ex99

```
LTransform_Ex99 LTransform_Add_Ex99(LTransform_Ex99
    transform_to_be_added, LTransform_Ex99
    current_transform);
```

Description

Adds two transformations. A transform is the translation, orientation, or magnification of an object.

Return Values

Returns the sum of ***transform_to_be_added*** and ***current_transform*** as an **LTransform**.

Parameters

transform_to_be_added	Transformation structure 1.
current_transform	Transformation structure 2.

See Also

LTransform_Add (page 4-501), **LTransform** (page 4-532), **Transformation Functions** (page 4-494)

LTransform_Subtract

```
LTransform LTransform_Subtract(LTransform
    transform_to_be_subtracted, LTransform
    current_transform);
```

Description

Subtracts ***transform_to_be_subtracted*** from ***current_transform***. A transform is the translation, orientation, or magnification of an object.

Return Values

The resulting ***transform*** if successful; zero ***transform*** otherwise.

Parameters

transform_to_be_subtracted	Transformation structure 1.
current_transform	Transformation structure 2.

See Also

LTransform (page 4-532), **Transformation Functions** (page 4-494)

LTransform_Subtract_Ex99

```
LTransform_Ex99 LTransform_Subtract_Ex99(LTransform_Ex99
    transform_to_be_subtracted, LTransform_Ex99
    current_transform);
```

Description

Subtracts one transform from another. A transform is the translation, orientation, or magnification of an object.

Return Values

The resulting ***transform*** if successful; zero ***transform*** otherwise.

Parameters

transform_to_be_subtracted	Transformation structure 1.
current_transform	Transformation structure 2.

See Also

LTransform_Subtract (page 4-505), **LTransform** (page 4-532), **Transformation Functions** (page 4-494)

Typedefs

(continued)

(continued)

LStatus

```
typedef enum {
     LStatusOK,
     LTooManyInits,
     LOpenError,
     LCloseError,
     LCreateError,
     LSaveError,
     LBadFile,
     LBadCell,
     LBadLayer,
     LBadParameters,
     LBadObject,
     LBadHierarchy,
     LUserDataError,
     LCellOverWritten,
     LLayerMapsDifferent,
     LNamedCellExists,
     LCopyProtViolation,
     LNoSelection
     LPropertyHasNoValue
     LPropertyTypeMismatch
     LBufferTooSmall
     LVertexNotFound
     LCantDeleteVertex
          } LStatus;
```

Description

LStatus is an enumeration of various error returns. A return value of zero indicates no errors; a value greater than zero indicates an error by its position in the list.

LDialogItem

```
typedef struct {
     char prompt[40];
     char value[21];
          } LDialogItem;
```

Description

Defines the ***prompt*** and ***value*** fields associated with a multiple-line input dialog. This structure is used by **LDialog_MultiLineInputBox.**

LFile

```
typedef struct _LFile *LFile;
```

Description

A pointer to an L-Edit layout file whose contents can only be accessed or modified through UPI functions.

LFileType

```
typedef enum {
     LTdbFile,
     LCifFile,
     LGdsFile
          } LFileType;
```

Description

Lists the three design formats supported by L-Edit: Tanner Database (TDB) format, Caltech Intermediate Form (CIF), and GDS II (stream) format. CIF and GDS II are standard machine-readable formats for representing IC layouts.

LEnvironment

```
typedef struct _LEnvironment {
     short MenuBackgroundColor;
     short MenuForegroundColor;
     short MenuSelectColor;
     short AlertBackgroundColor;
     long DefaultPortTextSize;
     int DropDownMenus;
     int ActivePushRubberbanding;
     int AutoPanning;
     int StatusBar;
     int HideInsides;
     short HorizontalPixels;
     short VerticalPixels;
          } LEnvironment;
```

Description

Used to get and set the environment of a design file. All colors take values between 0 and 15. The **int** quantities take values of either 0 or 1, equivalent to the off and on states of the corresponding switches in the **Setup Application** dialog**.**

Note: The color parameters are not applicable to L-Edit for Windows.

See Also

Application Parameters on page 1-111.

LCursorType

```
typedef enum {
     LSnapping,
     LSmooth
     } LCursorType;
```

Description

Lists the cursor's (mouse pointer's) modes of movement: bound to the mouse snap grid points ("snapping") or unconstrained ("smooth").

LGrid

```
typedef struct {
     long displayed_grid_size;
     long min_grid_pixels;
     long mouse_snap_grid_size;
     LCursorType cursor_type;
     long locator_scaling;
     } LGrid;
```

Description

Used to get and set the grid parameters of the design file. The fields appear as corresponding items in the **Setup Application—Grid**.

LCell

```
typedef struct _LCell *LCell;
```

Description

A pointer to an L-Edit cell whose contents can only be accessed or modified through UPI functions.

LInstance

```
typedef struct _LInstance *LInstance;
```

Description

A pointer to an L-Edit instance whose contents can only be accessed or modified through UPI functions.

LObject

```
typedef union _LObject *LObject;
```

Description

A pointer to an L-Edit object whose contents can only be accessed or modified through UPI functions.

LShapeType

```
typedef enum {
     LBox,
     LCircle,
     LWire,
     LPolygon,
     LTorus,
     LPie,
     LOtherObject,
     LObjInstance,
     LObjPort,
     LObjRuler
} LShapeType;
```

Description

An enumeration of the object type of an L-Edit object.

LGeomType

```
typedef enum {
     LOrthogonal,
     LFortyFive,
     LAllAngle,
     LNonGeometric,
     LManhattan = LOrthogonal,
     LBoston = LFortyFive,
          } LGeomType;
```

Description

An enumerated datatype indicating the geometry type of an L-Edit object.

LCoord

```
typedef long LCoord;
```

Description

The basic internal unit coordinate type for the L-Edit layout space.

LPoint

```
typedef struct {
     LCoord y, x;
          } LPoint;
```

Description

A point in the L-Edit two-dimensional layout space.

LRect

```
typedef struct {
     LCoord y0, x0;
     LCoord y1, x1;
          } LRect;
```

Description

The coordinates of a rectangle in layout space. Here, (x0, y0) is the lower left corner of a rectangle and (x1, y1) is the upper right corner.

LOrientation

```
typedef long int LOrientation;
     #define LNormalOrientation      0
     #define LRotate0                0
     #define LRotate90              90
     #define LRotate180            180
     #define LRotate270            270
     #define LRotate0MirrorX      -360
     #define LRotate90MirrorX      -90
     #define LRotate180MirrorX    -180
     #define LRotate270MirrorX    -270
```

Description

A rotation and/or mirror operation that may be applied to any L-Edit objects.

LOrientation_Ex99

```
typedef float LOrientation_Ex99;
     #define LNormalOrientation      0
     #define LRotate0                0
     #define LRotate90              90
     #define LRotate180            180
     #define LRotate270            270
     #define LRotate0MirrorX      -360
     #define LRotate90MirrorX      -90
     #define LRotate180MirrorX    -180
     #define LRotate270MirrorX    -270
```

Description

A rotation and/or mirror operation that may be applied to any L-Edit objects. Rotation can be specified as any real number.

See Also

LOrientation (page 4-529)

LMagnification

```
typedef struct LMagnification {
     LLen num;
     LLen denom;
          } LMagnification;
```

Description

Specifies the scaling of an object.

LTransform

```
typedef struct {
     LPoint translation;
     LOrientation orientation;
     LMagnification magnification;
          } LTransform;
```

Description

Specifies the translation, orientation, and magnification of an object. All objects, ports, and instances can be transformed.

See Also

LPoint (page 4-527), **LOrientation** (page 4-529), **LMagnification** (page 4-531), **LTransform_Ex99** (page 4-533)

LTransform_Ex99

```
typedef struct {
     LPoint translation;
     LOrientation_Ex99 orientation;
     LMagnification magnification;
          } LTransform_Ex99;
```

Description

Specifies the translation, orientation as a real number, and magnification of an object. All objects, ports, and instances can be transformed.

See Also

LPoint (page 4-527), **LMagnification** (page 4-531), **LTransform** (page 4-532)

LLen

```
typedef unsigned long LLen;
```

Description

The internal unit used for specifying the magnification ratio.

LWireConfig

```
typedef struct {
     LCoord width;
     LJoinType join;
     LCapType cap;
     LCoord miter_angle;
          } LWireConfig;
```

Description

Specifies the configuration of a wire. The configuration of a wire includes width, join type, cap type, and miter angle.

See Also

LCoord (page 4-526), **LJoinType** (page 4-538), **LCapType** (page 4-537)

LWireConfigBits

```
typedef enum {
     LSetWireWidth = 1 << 0,
     LSetWireJoin = 1 << 1,
     LSetWireCap = 1 << 2,
     LSetWireMiterLimit = 1 << 3,
     LSetWireAll = -1
          } LWireConfigBits;
```

Description

Used to mask out configuration properties that you do not wish to set.

See Also

LWireConfig (page 4-535)

LCapType

```
typedef enum {
     LCapButt,
     LCapRound,
     LCapExtend
          } LCapType;
```

Description

Defines the end style of a wire.

LJoinType

```
typedef enum {
     LJoinMiter,
     LJoinRound,
     LJoinBevel
          } LJoinType;
```

Description

Defines the join style of a wire.

LPort

```
typedef struct _LPort *LPort;
```

Description

A pointer to an L-Edit port whose contents can only be accessed or modified through UPI functions. A port is a text whose location in a cell is specified by a rectangle, and it is typically used for documentation or routing purposes. Each cell has a single list of ports. Each port has a layer, text, and a location associated with it.

LSelection

```
typedef struct _LSelection *LSelection;
```

Description

A pointer to the L-Edit selection list whose contents can only be accessed or modified through UPI functions.

LSelectionParam

```
typedef struct _LSelectionParam {
     long selection_range;
     long deselect_distance_2;
     long deselect_distance_1;
     long lambda_edit_range;
     long pixel_edit_range;
     int select_draws;
          } LSelectionParam;
```

Description

Used to get and set the selection setup of file. This structure is used to specify the selection range, deselection range, and the edit range. The switch **select_draws** determines if an object will be automatically selected after it is created.

LLayer

```
typedef struct _LLayer *LLayer;
```

Description

A pointer to an L-Edit layer whose contents can only be accessed or modified through UPI functions.

LWireParam

```
typedef struct {
     long defaultWireWidth;
     short defaultWireMiterAngle;
     LCapType capType;
     LJoinType joinType;
          } LWireParam;
```

Description

Specifies the default properties of a wire: wire width, miter angle, join style, and end style.

See Also

LCapType (page 4-537), **LJoinType** (page 4-538)

LLayerViewStatus

```
typedef enum {
     LHidden,
     LVisible
          } LLayerViewStatus;
```

Description

Used to make a layer visible or hidden.

LLayerParam

```
typedef struct {
     char CIFName [7];
     short GDSNumber;
     double cap;
     double rho;
     int lock;
     LLayerViewStatus viewStatus;
     LWireParam wireParam;
          } LLayerParam;
```

Description

A structure where layer information can be stored. It specifies the CIF name, GDS number, capacitance, and resistance of a layer.

Return Values

When lock is zero, a layer is locked.

Parameters

viewStatus	Indicates whether a layer is visible or hidden.
wireParam	Specifies the properties of a wire that can be drawn using this layer.

See Also

LLayerViewStatus (page 4-544), **LWireParam** (page 4-543)

LDerivedLayerParam

```
typedef struct _LDerivedLayerParam {
     int enable_evaluation; /*if 0 evaluation disabled else enabled*/
     char *name; /*Name of the derived layer*/
     char *src_layer1; /*Name of the first source layer*/
     char *src_layer2; /*Name of the second source layer*/
     char *src_layer3; /*Name of the third source layer*/
     int layer1_not_op; /*If NOT operator enabled for 1st source layer*/
     long layer1_grow_amount; /*grow amount for first source layer*/
     int layer2_not_op; /*If NOT operator enabled for 2nd source layer*/
     long layer2_grow_amount; /*grow amount for second source layer*/
     int layer3_not_op; /*If NOT operator is enabled for layer 3*/
     long layer3_grow_amount; /*grow amount for third source layer*/
     int layer1_bool_layer2; /*1=> AND, 0=> OR of 1st &2nd source layer*/
     int layer2_bool_layer3; /*1=> AND, 0=> OR of 1st &3rd source layer*/
} LDerivedLayerParam;
```

Description

Used to get and set the parameters of a generated layer.

LSpecialLayer

```
typedef enum {
     GridLayer,
     OriginLayer,
     CellOutlineLayer,
     ErrorLayer,
     IconLayer,
     FirstMaskLayer,
     DragBoxLayer
          } LSpecialLayer;
```

Description

An enumerated datatype that specifies the type of a special layer.

LPass

```
typedef struct _LPass *LPass;
```

Description

A pointer to a layer's pass list, whose contents can only be accessed or modified through UPI functions.

LPassMode

```
typedef unsigned char LStipple[8];
typedef enum {
     LSet=16,
     LClear=8
          } LPassMode;
```

Description

LPassMode is used to specify the write mode of a pass (set or clear).

LPassType

```
typedef enum {
     LObjectPass,
     LPortPass,
     LTextPass
          } LPassType;
```

Description

Specifies the type of a pass (object, port, or text).

LPassParam

```
typedef struct _LPassParam {
     unsigned char ColorIndex;
     LPassMode WriteMode;
     LStipple Stipple;
          } LPassParam;
```

Description

Specifies the properties of a pass, including its color index, pass mode, and stipple pattern.

See Also

LPassMode (page 4-550)

LColor

```
typedef struct {
     short LRed;
     short LBlue;
     short LGreen;
          } LColor;
```

Description

Defines a color to be used by L-Edit.

LPalette

```
typedef LColor LPalette[16];
```

Description

Gets and sets the color palette of a layout file.

LDrcRule

```
typedef struct _LDrcRule *LDrcRule;
```

Description

A pointer to an L-Edit design rule whose contents can only be accessed or modified through UPI functions.

LDrcRuleType

```
typedef enum {
     LMIN_WIDTH,
     LEXACT_WIDTH,
     LOVERLAP,
     LEXTENSION,
     LNOT_EXISTS,
     LSPACING,
     LSURROUND
          } LDrcRuleType;
```

Description

LDrcRuleType is an enumerated datatype used to specify the type of a design rule.

LDesignRuleParam

```
typedef struct _LDesignRuleParam {
     int enable; /*0=>disabled, 1=>enabled*/
     char *name; /*Name of the design rule*/
     LDrcRuleType rule_type; /*type of a design rule*/
     int ignore_coincidences; /*0=>false, 1=>true*/
     int ignore_intersections; /*0=> false, 1=>true*/
     int ignore_enclosures; /*0=> false, 1=>true*/
     int ignore_45_acute_angles; /*0=> false, 1=>true*/
     char *layer1; /*Name of the first layer involved in in design rule*/
     char *layer2; /*Name of the second layer involved in design rule*/
     long distance; /*Distance value associated with a rule*/
     int use_internal_units; /*0=> false, 1=>true :False=> use LAMBDA*/
     } LDesignRuleParam;
```

Description

This structure is used to get and set parameters of a design rule.

LDesignRuleFlags

```
typedef struct
{
      int FlagSelfIntersection,
      LAmbiguousFillType PolygonsWithAmbiguousFills,
      int FlagIgnoredObjects
      int FlagOffGridObjects
      double GridSize
      int UseLocatorUnits
}
LDesignRuleFlags;
```

Description

FlagSelfIntersection	Ignore polygons with ambiguous fills.
PolygonsWithAmbiguousFills	Flag polygons with ambiguous fills, fix polygons with ambiguous fills, or ignore them.
FlagIgnoredObjects	Flag objects not checked by DRC.
FlagOffGridObjects	Flags off-grid objects.

GridSize	Used to flag off-grid objects.
UselocatorUnits	Indicates whether GridSize is in technology units or locator units.

See Also

LAmbiguousFillType (page 4-584), **LFile_GetDesignRuleFlags** (page 4-150), **LFile_SetDesignRuleFlags** (page 4-152).

LCIFParam

```
typedef struct {
        int poly_to_rect;
        int port_rect;
                } LCIFParam;
```

Description

LCIFParam is used to get and set the CIF setup of a file. Rectangular polygons are read as boxes if ***poly_to_rect*** is 1. Port boxes are written out if ***port_rect*** is 1.

See Also

Importing Files (page 1-185)

LGDSParam

```
typedef struct {
     int upcase_cell_name;
     short circle_to_polygon_sides;
     int use_default_units;
          } LGDSParam;
```

Description

LGDSParam is used to get and set the GDSII setup of a file. If ***upcase_cell_name*** is 1, L-Edit will write out all cells with uppercase names. Circle are written out as *n*-sided polygons. The number of polygon sides is specified in ***circle_to_polygon_sides***. If ***use_default_units*** is 1, default GDSII units are used.

See Also

Exporting Files (page 1-191)

LExtractOptions

```
typedef struct _LExtractOptions
{
     char szExtDefnFile[256];
     char szExtOutFile[256];
     double dExtractBinSize;
     int iWriteNodeNames;
     int iWriteDeviceCoord;
     int iWriteShortedDevices;
     int iWriteParasiticCap;
     double dParasiticCutoff;
     int iWriteNodesAs;
     int iWriteSciNotation;
     int iWriteVerboseSPICE;
     char *szExtIncludeStmt;
     int iLabelAllDevices;
     LLayer oDeviceLabelLayer;
     int iSubCktRecognition;
     LLayer oSubCktRecogLayer;
     int iUseSubCktNetlistFmt;
     int iFlagImproperOverlaps;
     LLayer oIgnoreConnPortLayer;
     char szIgnoreConnPort[256];
     char szIgnoreCrossPort[256];
     long lMaxIncludeStmtLen;
} LExtractOptions;
```

Description

Used to get and set the extract options for a cell. The **int** quantities take values of either 0 or 1, equivalent to the off and on states of the corresponding switches in the Extract dialog. All options available in the extract dialog can be set with the above structure.

General Options

szExtDefnFile	Character string of the extract definition file. (256 characters max).
szExtOutFile	Character string of the extract SPICE output file. (256 characters max).
dExtractBinSize	Bin size in locator units.

Output Options

iWriteNodeNames	Write node names in comments. (0 - False, Otherwise True).
iWriteDeviceCoord	Write device coordinates in comments. (0 - False, Otherwise True).
iWriteShortedDevices	Write shorted devices in comments. (0 - False, Otherwise True).
iWriteParasiticCap	Write parasitic capacitances. (0 - False, Otherwise True).
dParasiticCutoff	Cutoff value for parasitic capacitors. (in Femtofarads).

iWriteNodesAs	Write nodes as (integers or names). (0 - Integers, Otherwise Names).
iWriteSciNotation	Write values in scientific notation. (0 - False, Otherwise True).
iWriteVerboseSPICE	Write R, L, C with verbose style (R=, L=, C=). (0 - False, Otherwise True).
szExtIncludeStmt	SPICE include statement.
iLabelAllDevices	Create ports for all devices. (0 - False, Otherwise True).
oDeviceLabelLayer	Place device labels on this layer.

Subcircuit Options

iSubCktRecognition	Recognize subcircuit instances. (0 - False, Otherwise True).
oSubCktRecogLayer	Subcircuit recognition layer.
iUseSubCktNetlistFmt	Write netlist as a subcircuit. (0 - False, Otherwise True).
iFlagImproperOverlaps	Flag improper overlaps. (0 - False, Otherwise True).
oIgnoreConnPortLayer	Ignore connection ports on this layer.
szIgnoreConnPort	Ignore connection ports with this name.
szIgnoreCrossPort	Ignore cross ports with this name.

Miscellaneous

lMaxIncludeStmtLen	Length of the **.include** statement string.

LCore

```
typedef struct _LCore *LCore;
```

Description

A pointer to an L-Edit standard cell place-and-route core, whose contents can only be accessed or modified through UPI functions. A core is generated by the standard cell place and route utility.

Ltech_unit_type

Description

Specifies the units of technology measurement.

```
typedef enum {
     MICRONS,
     MILLIMETERS,
     CENTIMETERS,
     MILS,
     INCHES,
     LAMBDA,
     OTHER
          } tech_unit_type;
```

LTechnology

Description

A structure where information about the technology of the design file can be stored.

```
typedef struct _LTechnology {
     const char* name; /*Technology name*/
     tech_unit_type unit_type; /*Unit of measurement*/
     const char* unit_name; /*Other unit name*/
     long num; /*Numerator of mapping*/
     long denom; /*Denominator of mapping*/
     long lambda_num; /*Numerator, lambda mapping*/
     long lambda_denom; /*Denominator, Lambda mapping*/
          } LTechnology;
```

LPropItem

```
typedef struct _LPropItem *LPropItem;
```

Description

A pointer to an L-Edit property item whose contents can only be accessed or modified through UPI functions.

LPropItemType

```
typedef enum {
     LPROP, LPROP_ITEM, LPROP_NONE
          } LPropItemType;
```

Description

Used to get and set the type of the property item.

LProp

```
typedef struct _LProp *LProp;
```

Description

A pointer to an L-Edit property whose contents can only be accessed or modified through UPI functions.

LPropType

```
typedef enum {
     LPT_INT, LPT_FLOAT, LPT_BOOL, LPT_STRING, LPT_EQN,
     LPT_LINK, LPT_BLOB, LPT_GROUP, LPT_UNKNOWN
          } LPropType ;
```

Description

An enumerated datatype indicating the type of an L-Edit property.

LPropCount

```
typedef short LPropCount;
```

Description

The basic type indicating property count for an L-Edit property.

LPropAttrib

```
typedef long LPropAttrib;
```

Description

The basic type indicating property attribute for an L-Edit property.

LBoolean

```
typedef enum {LFALSE,LTRUE} LBoolean;
```

Description

An enumerated datatype indicating the Boolean value of an L-Edit property value.

LPropVal

```
typedef union LpropertyVal {
     int LintVal;
     float LfloatVal;
     Lboolean LBoolVal;
     char* LstrVal;
          } LPropVal;
```

Description

A union datatype indicating value of an L-Edit property.

LWindow

```
typedef struct _LWindow *LWindow;
```

Description

A pointer to an L-Edit window whose contents can only be accessed or modified through UPI functions.

LWindowType

```
typedef enum {
     UNKNOWN = 0,
     CELL_BROWSER,
     TEXT,
     LAYOUT,
     CROSS_SECTION
          } LWindowType;
```

Description

Lists the window types that can be opened in L-Edit.

UPIDrawingToolType

```
typedef enum
{
     LSelectionTool = 0; /*Selection tool*/
     LBoxTool; /*Box tool*/
     LPolygon90Tool; /*Orthogonal polygon tool*/
     LPolygon45Tool; /*45 degree polygon tool*/
     LPolygonAATool; /*All angle polygon tool*/
     LWire90Tool; /*Orthogonal wire tool*/
     LWire45Tool; /*45 degree wire tool*/
     LWireAATool; /*All angle wire tool*/
     LCircleTool; /*Circle tool*/
     LPieWedgeTool; /*Pie wedge tool*/
     LTorusTool; /*Torus tool*/
     LPortTool; /*Port tool*/
     LRuler90Tool; /*Orthogonal ruler tool*/
     LRuler45Tool; /*45 degree tool*/
     LRulerAATool; /*All angle tool*/
     LInstanceTool; /*Instance tool (Not currently implemented*/
     LBPRRoute90Tool; /*Orthogonal BPR routing tool*/
     LBPRRoute45Tool; /*45 degree BPR routing tool*/
     LBPRRouteAATool; /*All angle BPR routing tool*/
} UPIDrawingToolType;
```

Description

An enumeration of the different drawing tools.

LObject_GetInstance

```
LObject LObject_GetInstance (LCell pCell, LObject pObject)
```

Description

Retrieves any instance of the specified object. Retrieves the instance object from an LObject, if possible.

Return Values

Returns the instance of the object if successful; otherwise returns NULL.

Parameters

pCell	Specified cell containing the object.
pObject	Specified object.

See Also

LObject (page 4-523), **Object Functions** (page 4-232), **LCell** (page 4-521)

LPropertyType

```
typedef enum {
     L_unassigned,
     L_none,
     L_int, /* int */
     L_real, /* double */
     L_bool, /* int, 0 false, otherwise true */
     L_string, /* char* */
     L_enum, /* not in use */
     L_byte, /* char */
     L_ptr, /* void* */
     L_blob, /* void* */
} LPropertyType;
```

Description

An enumerated datatype indicating the type of an L-Edit property.

LAmbiguousFillType

```
typedef enum
{
     LDo_Not_Flag = 0,
     LFlag,
     LFix
} LAmbiguousFillType;
```

Description

LDo_Not_Flag	Ignore polygons with ambiguous fills
LFlag	Flag polygons with ambiguous fills
LFix	Fix polygons with ambiguous fills

See Also

LDesignRuleFlags (page 4-558), **LFile_GetDesignRuleFlags** (page 4-150), **LFile_SetDesignRuleFlags** (page 4-152)

LDerivedLayerParamEx00

```
typedef struct _LDerivedLayerParamEx00
{
      int enable_evaluation; /*is derivation enabled? */
      LDerivationType  derivation_type; /* information
   about derivation type */
      LDerivedLayerOperation operation; /* derivation type
   specific information */
}
LDerivedLayerParamEx00;
```

Description

Contains information about layer derivation.

See Also

LDerivationType (page 4-586), **LDerivedLayerOperation** (page 4-587), **LLayer_GetDerivedParametersEx00** (page 4-387), **LLayer_SetDerivedParametersEx00** (page 4-390).

LDerivationType

```
typedef enum
{
	LDOT_Bool=0, /* Boolean derivation as in L-Edit 8.2
   */
	LDOT_Area=1, /* Area derivation */
	LDOT_Select=2 /* Select derivation */
}
LDerivationType;
```

See Also

LDerivedLayerParamEx00 (page 4-585), **LDerivedLayerOperation** (page 4-587).

LDerivedLayerOperation

```
typedef union _LDerivedLayerOperation
{
        LDerivedLayerBoolOperation boolean;
        LDerivedLayerSelectOperation select;
        LDerivedLayerAreaOperation area;
}
LDerivedLayerOperation;
```

Description

One of three possible derivations, as determined externally by the variable derivation_type in **LDerivedLayerParamEx00** (page 4-585).

See Also

LDerivationType (page 4-586), **LDerivedLayerBoolOperation** (page 4-588), **LDerivedLayerSelectOperation** (page 4-589), **LDerivedLayerAreaOperation** (page 4-591).

LDerivedLayerBoolOperation

```
typedef struct _ LDerivedLayerBoolOperation
{
     LLayer src_layer1;/*Name of the first source layer*/
     LLayer src_layer2;/*Name of the second source layer*/
     LLayer src_layer3;/*Name of the third source layer*/
     int layer1_not_op;/*If NOT operator enabled for 1st source layer*/
     long layer1_grow_amount;/*grow amount for first source layer*/
     int layer2_not_op;/*If NOT operator enabled for 2nd source layer*/
     long layer2_grow_amount;/*grow amount for second source layer*/
     int layer3_not_op;/*If NOT operator enabled for 3rd source layer*/
     long layer3_grow_amount;/*grow amount for third source layer*/
     int layer1_bool_layer2;/*1=> AND, 2=> OR of 1st & 2nd source layer*/
     int layer2_bool_layer3;/*1=> AND, 2=> OR of 1st & 3rd source layer*/
}
LDerivedLayerBoolOperation;
```

Description

Used to get and set the parameters of a Boolean generated layer.

See Also

LDerivationType (page 4-586), **LDerivedLayerParamEx00** (page 4-585).

LDerivedLayerSelectOperation

```
typedef enum
{
      LDOST_Inside=0,
      LDOST_Outside,
      LDOST_Hole,
      LDOST_Cut,
      LDOST_Touch,
      LDOST_Enclose,
      LDOST_Overlap,
      LDOST_Vertex,
      LDOST_Density
}
LSelectOperationRelationType;
typedef struct _LDerivedLayerSelectOperation
{
      LLayer layer1; /* name of the first source layer */
      LLayer layer2; /* name of the second source layer.
   not valid if relation_type is LDOST_Vertex */
      int not_op; /* 1=NOT */
      long vertex_range_n1; /* lower value for vertex
   range. Valid only if relation_type is LDOST_Vertex. */
      long vertex_range_n2; /* higher value for vertex
   range. Valid only if relation_type is LDOST_Vertex. */
      double density_range_n1; /* lower value for density
   range. Valid only if relation_type is LDOST_Density. */
```

```
      double density_range_n2; /* higher value for density
   range. Valid only if relation_type is LDOST_Density. */
LSelectOperationRelationType relation_type; /* indicates
   which relation type is used for derivation. */
}
LDerivedLayerSelectOperation;
```

Description

Used to get and set the parameters of a layer generated using Select operations.

See Also

LDerivationType (page 4-586), **LDerivedLayerParamEx00** (page 4-585).

LDerivedLayerAreaOperation

```
typedef enum
{
  LDOAT_Range=0,
  LDOAT_NE,
  LDOAT_EQ
}
LAreaOperationConditionType;

typedef struct _ LDerivedLayerAreaOperation
{
       LLayer layer1; /* name of the first source layer */
       LLayer layer2; /* name of the second source layer */
       int not_op; /* 1=NOT */
       long range_n1; /* lower value of area range */
       long range_n2; /* higher value of area range */
       int use_locator_units; /* 1=use locator units, 0=use
   technology units */
       LAreaOperationConditionType condition_type; /* the
   condition type */
}
LDerivedLayerAreaOperation;
```

Description

Used to get and set the parameters of a layer generated using Area operations.

See Also

LDerivationType (page 4-586), **LDerivedLayerParamEx00** (page 4-585).

LRenderingAttributeIndex

```
typedef enum _LRenderingAttributeIndex
{
        raiFirstRenderingAttribute = 0,
        raiObject = 0,
        raiPortBox = 1,
        raiPortText = 2,
        raiWireCenterline = 3,
        raiSelectedObject = 4,
        raiSelectedPortBox = 5,
        raiSelectedPortText = 6,
        raiSelectedWireCenterline = 7,
        raiLastRenderingAttribute = 7
}
LRenderingAttributeIndex;
```

Description

Lists the available rendering attributes.

LRenderingAttribute

```
typedef enum _LRenderingMode
{
        rmPaint = 0,
        rmAdd = 1,
        rmSubtract = 2
}
LRenderingMode;

typedef enum _LOutlineUnitType
{
        utPixels = 0,
        utLocatorUnits = 1
}
LOutlineUnitType;

typedef struct _LRenderingAttribute
{
        LRenderingMode      mMode; /* rmPaint=draw, rmAdd=OR
    with background, rmSubtract=AND with background */
        unsigned int        mPass; /* Pass number from 1 to
    10 */
        LStipple            mFillPattern; /* Pattern */
        unsigned int        mFillColorIndex; /* Color */
        LStipple            mOutlinePattern; /* Outline
    pattern */
```

```
        unsigned int          mOutlineColorIndex; /* Outline
    color */
        LOutlineStyle         mOutlineStyle; /* Outline style
    */
        LOutlineUnitType      mOutlineWidthUnits; /* utPixels
    or utLocatorUnits. utPixels is recommended for better
    performance */
        unsigned int          mOutlineWidth; /* 1 is
    recommended for best performance */
}
LRenderingAttribute;

typedef LRenderingAttribute *LLRenderingAttribute;
```

Description

Defines rendering attributes.

See Also

LOutlineStyle (page 4-596), **LStipple** (page 4-598), **LRenderingAttributeIndex** (page 4-593), **LLayer_GetRenderingObjectName** (page 4-415), **LLayer_GetRenderingAttribute** (page 4-409), **LLayer_SetRenderingAttribute** (page 4-412), **LFile_GetColorPalette** (page 4-429).

LOutlineStyle

```
typedef struct _LOutlineStyle
{
	unsigned short mLengthOfPattern; /* the number of 
   bits in the pattern */
	unsigned short mBitPattern; /* the pattern bits */
}
LOutlineStyle;
```

Predefined Outline Styles

```
LOutlineStyle OutlineStyleSolid = { 0x0001,  1 };
LOutlineStyle OutlineStyleDotted = { 0x0002,  2 };
LOutlineStyle OutlineStyleShort = { 0x000C,  4 };
LOutlineStyle OutlineStyleShortDot = { 0x001A,  5 };
LOutlineStyle OutlineStyleLongDot = { 0x007A,  7 };
LOutlineStyle OutlineStyleLong = { 0x00F0,  8 };
LOutlineStyle OutlineStyleLongDotDot = { 0x01EA,  9 };
LOutlineStyle OutlineStyleLongShortShort = { 0x07B6, 11 };
LOutlineStyle OutlineStyleLongLongShort  = { 0x1EF6, 13 };
```

Description

This structure defines outline style.

See Also

LRenderingAttribute (page 4-594), **LRenderingAttributeIndex** (page 4-593), **LLayer_GetRenderingAttribute** (page 4-409), **LLayer_SetRenderingAttribute** (page 4-412), **LLayer_GetRenderingObjectName** (page 4-415).

LStipple

```
typedef unsigned char LStipple[8];
```

Description

Each unsigned character in the array LStipple is a bitmap of a row in the 8x8 stipple pattern.

Stipple is an array of eight charaters, each corresponding to the bitmap of a row in the 8x8 stipple array (shown in the **Create Pattern** dialog in the Fill **Pattern** pull-down menu of the **Setup Layers** dialog.) If, for example, the stipple is {11000000, 00110000, 00001100, 00000011, 11000000, 00110000, 00001100, 00000011} in binary notation, the layer's color will show only in those pixels that are marked with x below:

```
xx......
..xx....
....xx..
......xx
xx......
..xx....
....xx..
......xx
```

See Also

LRenderingAttribute (page 4-594).