cadence

What's New in LEF 5.8 C/C++ Programming Interface

Product Version 5.8
May 2017

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Contents

About This Manual

This document provides information on new and changed features for version 5.8 of the C and C++ application programming interface (API) used to read and write Cadence® Library Exchange Format (LEF) files.

How This Document Is Organized

This *What's New* document is organized into the following chapters:

- New Features

  This chapter describes features that were added since version 5.7 of the LEF API. New features are those that introduce new functionality into the LEF API. Any enhancements made to existing statements to support a new feature are also described in this chapter.

- Changed Features

  This chapter discusses features that were changed since version 5.7 of the LEF API. Changed features include such things as changes in default behavior, changes in whether keywords and statements are required, and any other changes that do not reflect new functionality.

Related Documents

The following documents provide detailed information about LEF and DEF, and the LEF and DEF application programming interfaces.

- *DEF C/C++ Programming Interface (Open Licensing Program)*
- *LEF C/C++ Programming Interface (Open Licensing Program)*
- *LEF/DEF Language Reference*
- *What's New in DEF C/C++ Programming Interface*
- *What's New in LEF/DEF*

7/10/17

1

New Features

This chapter describes the new features that were added in this release of the LEF application programming interface.

General LEF Reader Changes

The following changes were made in the latest version of the parser:

- **Elimination of static data**: In the latest version, the parser architecture has changed from C style-based to C++ style-based. In the previous version of the parser, most of the parser data were stored in static variables and the data lifecycle was based on initializers and cleaners. The new architecture places data in data singletons and uses C++ constructors and destructors. The switch to the C++ architecture has improved the parser re-enterability, made the data flow more robust, and helped clean multiple memory leaks in the parser code.

- **Introduction of parsing sesions**: In the previous version, data were stored in static variables and, therefore, were retained across all parsing cycles. This meant that if a property was defined once, it continued to be defined in the next LEF file reads. In some applications, this feature was actively used. In others, it disturbed expected application behavior. To address this issue, the latest version of the parser introduces two modes of files processing - compatibility mode and session-based mode.

  - Compatibility mode (session-less mode) - This mode is compatible with the old parser behavior. You can call the parser initialization once with `lefrInit()`, adjust parsing settings and initialize the parser callbacks any time. The properties in `PROPERTYDEFINITIONS` sections will be active in all subsequent file reads.

  - Session-based mode - This mode introduces the concept of a parsing session – the parser configuration settings will be active during the session time and will be cleaned on its end.The parsing session also controls `PROPERTYDEFINITIONS` data. Property definitions remain active throughout the parsing session time and are cleaned at the end of the session. The session-based mode does not require calling callbacks and configuration unset functions - all callbacks and properties will be set to defaults by `lefrClear()` or the next session `lefrInitSession()` call.

  By default, the LEF parser works in the compatibility mode. To activate the session-based mode, you must use `lefrInitSession()` instead of `lefrInit()`.

  Note: Currently, the compatibility mode is used for all old applications for which code has not been adjusted. The `lef2oa` and `def2oa` translators have adjusted to use the session-based parsing mode.

  For more information, see "LEF Reader Working Modes" in the *LEF 5.8 C/C++ Programming Interface (Open Licensing Program)*.

- **Long DEF Files Support**: In this version, the LEF line counter switched to 64-bit integer type, making it possible to process files with more than two billion lines.

LEF Reader Setup and Control Routines

The following reader setup and class routines were added in this release:

- lefrInitSession
- lefrClear
- lefrRegisterLef58Type

For more information, see "LEF Reader Setup and Control Routines" in the *LEF 5.8 C/C++ Programming Interface (Open Licensing Program)*.

LEF Reader Callback Routines

The following reader callback routines were added in this release:

- lefrFixedMaskCbk
- lefrMacroFixedMaskCbk
- lefrSetMacroForeignCbk
- lefrSetMacroSiteCbk
- lefrUnsetMacroForeignCbk
- lefrUnsetMacroSiteCbk

For more information on reader callback routines, see "LEF Reader Callback Routines" in the *LEF 5.8 C/C++ Programming Interface (Open Licensing Program)*.

LEF Reader Classes

The following LEF Reader classes were added in this release:

- lefiMacroSite
- lefiMacroForeign

For more information on reader classes, see "LEF Reader Classes" in the *LEF 5.8 C/C++ Programming Interface (Open Licensing Program)*.

2

Changed Features

This chapter describes the features that were changed in this release of the LEF application programming interface.

Layer Routines

The following syntax has been added to the listed layer routines.

lefiGeometries

```
int colorMask;
void addPath(int colorMask);
void addPathIter(int colorMask);
void addRect(int colorMask);
void addRectIter(int colorMask);
void addPolygon(int colorMask);
void addPolygonIter(int colorMask);
void addVia(int viaMasks);
void addViaIter(int viaMasks);
```

These are described below:

`colorMask` Defines the color mask number for the `GeomRect` structure.

`addPathIter(int colorMask)`

Adds the color mask number to the `lefiGeomPath` structure. The default value is 0.

`addPathIter(int colorMask)`

Adds the color mask number to the `lefiGeomPathIter` structure. The default value is 0.

`addRectIter(int colorMask)`

Adds the color mask number to the `lefiGeomRectIter` structure. The default value is 0.

`addPolygon(int colorMask)`

Adds the color mask number to the `lefiGeomPolygon` structure. The default value is 0.

`addPolygonIter(int colorMask)`

Adds the color mask number to the `lefiGeomRectIter` structure. The default value is 0.

`addVia(int viaMasks)`

Adds the via mask number to the `lefiGeomVia` structure. The default value is 0.

`addViaIter(int viaMasks)`

Adds the via mask number to the `lefiGeomViaIter` structure. The default value is 0.

For more information, see "lefiGeometries" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiGeomRect

```
int colorMask;
```

Defines the color mask number for the `GeomRect struct`. The default value is 0.

For more information, see "lefiGeomRect" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiGeomRectIter

```
int colorMask;
```

Defines the color mask number for the `GeomRectIter struct`. The default value is 0.

For more information, see "lefiGeomRect" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiGeomPath

```
int colorMask;
```

Defines the color mask number for the `GeomPath struct`. The default value is 0.

For more information, see "lefiGeomPath" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiGeomPathIter

```
int colorMask;
```

Defines the color mask number for the `GeomPathIter struct`. The default value is 0.

For more information, see "lefiGeomPathIter" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiGeomPolygon

```
int colorMask;
```

Defines the color mask number for the `GeomPolygon struct`. The default value is 0.

For more information, see "lefiGeomPolygon" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiGeomPolygonIter

```
int colorMask;
```

Defines the color mask number for the `GeomPolygonIter struct`. The default value is 0.

For more information, see "lefiGeomPolygonIter" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiGeomVia

```
int topMaskNum;
int cutMaskNum;
int bottomMaskNum;
```

Indicates the top, bottom, and cut mask numbers for the `GeomVia struct`. The default value is 0.

For more information, see "lefiGeomVia" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiGeomViaIter

```
int topMaskNum;
int cutMaskNum;
int bottomMaskNum;
```

Indicates the top, bottom, and cut mask numbers for the `GeomViaIter struct`. The default value is 0.

For more information, see "lefiGeomViaIter" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiLayer

```
void setMask(int num);
int hasMask() const;
Int mask() const;
```

These are described below:

`setMask(int num)`	Sets the color mask number on the layer.
`hasMask()`	Checks whether the layer has a color mask assigned to it or not.
`mask()`	Returns the color mask number of the layer.

For more information, see "lefiLayer" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiViaLayer

```
void addRect(int mask,
     double xl,
     double yl
     double xh
     double yn);
void addPoly(int mask,
     lefiGeometries* geom);
int rectColorMask(int index);
int polyColorMask(int index);
```

These are described below:

`addRect(int mask)`	Adds the color mask number to the rectangle inside the via.The default value is 0.
`addPoly(int mask)`	Adds the color mask number to the polygon. The default value is 0.
`rectColorMask(int index)`	Returns the color mask number on the rectangle.
`polyColorMask(int index)`	Returns the color mask number on the polygon.

For more information, see "lefiViaLayer" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.

lefiVia

```
void addRectToLayer(int mask,
     double xl,
     double yl,
     double xh,
     double yh);
void addPolyToLayer(int mask,
     lefiGeometries* geom);
int rectColorMask(int layerNum,
     int rectNum);
int polyColorMask(int layerNum,
     int rectNum);
```

These are described below:

`addRectToLayer(int mask)`

Adds the color mask of the via rectangle to the layer. The default value is 0.

`addPolyToLayer(int mask)`

Adds the color mask of the via polygon to the layer. The default value is 0.

`rectColorMask(int layerNum, int rectNum)`

Returns the color mask number of the indexed rectangle inside the via for that layer.

`polyColorMask(int layerNum, int rectNum)`

Returns the color mask number of the indexed polygon inside the via for that layer.

For more information, see "lefiVia" in the *LEF C/C++ Programming Interface (Open Licensing Program)*.