cadence

What's New in DEF 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® Design Exchange Format (DEF) 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 DEF API. New features are those that introduce new functionality into the DEF 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 DEF 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 LEF 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 DEF application programming interface.

General 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 DEF 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 `defrInit()`, 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 `defrClear()` or the next session `defrInitSession()` call.

  By default, the DEF parser works in the compatibility mode. To activate the session-based mode, you must use `defrInitSession()` instead of `defrInit()`.

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

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

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

Reader Setup and Control Routines

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

- defrInitSession
- defrClear
- defrGetAllowComponentNets

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

Writer Routines - Blockages

The following writer routies for blockages were added in this release:

- defwBlockagesLayer
- defwBlockagesLayerFills
- defwBlockagesLayerPushdown
- defwBlockagesLayerExceptpgnet
- defwBlockagesLayerComponent
- defwBlockagesLayerDesignRuleWidth
- defwBlockagesPlacement
- defwBlockagesPlacementComponent
- defwBlockagesPlacementPushdown
- defwBlockagesPlacementSoft
- defwBlockagesPlacementPartial
- defwBlockagesRect
- defwBlockagesPolygon
- defwBlockagesLayerMask

Most of these functions duplicate old style functions with similar names. The old-style functions will be obsoleted in the next parser release.

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

Reader Class Routines

The following reader class routines were added in this release:

- `defiComponentMaskShiftLayer`

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

DEF Reader Callback Routines

The following reader callback routines were added in this release:

- `defrComponentMaskShiftLayerCbkType`

For more information, see "DEF Reader Callback Routines" in the *DEF C/C++ Programming Interface (Open Licensing Program)*.

2

Changed Features

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

Reader Class Routines

The following syntax was added to the listed reader class routines.

defiBlockage

```
void setMask(int maskColor);
int hasMask() const;
int mask() const;
```

Description:

`setMask(int maskColor)`	Adds the color mask number to the blockage. The default value is 0.
`hasMask()`	Checks if the blockage is colored.
`mask()`	Returns the color mask number of the blockage.

For more information, see "defiBlockage" in the *DEF C/C++ Programming Interface Open Licensing Program)*.

defiComponent

```
void setMaskShift(int color);
int maskShiftSize();
int maskShift(int index) const;
```

Description:

`setMaskShift(int color)`
: Sets the layer mask shift information.

`maskShiftSize()`
: Returns the number of shifted layer masks.

`maskShift(int index)`
: Returns the layer masks by index. `maskShift(0)` will return the right-most digit.

For more information, see "defiComponent" in the *DEF C/C++ Programming Interface Open Licensing Program)*.

defiFill

```
void setMask(int colorMask);
int layerMask() const
int viaTopMask() const;
int viaCutMask() const;
int viaBottomMask() const;
```

Description:

`setMask(int colorMask)`
: Specifies the mask number information to the layer. The default value is 0.

`layerMask()`
: Returns the layer mask information.

`viaTopMask()`
: Returns the top mask number of the via.

`viaCutMask()`
: Returns cut mask number of the via.

`viaBottomMask()`
: Returns bottom mask number of the via.

For more information, see "defiFill" in the *DEF C/C++ Programming Interface Open Licensing Program)*.

defiNet

```
void addPolygon(const char* layerName,
     defiGeometries* geom,
     int *needCbk,
     int colorMask,
...);

void addRect(const char* layerName,
     defiGeometries* geom,
     int *needCbk,
     int colorMask,
...);

void addPts(const char* viaName,
     defiGeometries* geom,
     int *needCbk,
     int colorMask,
...);

int polyMask(int index) const;
int rectMask(int index) const;
int topMaskNum(int index) const;
int cutMaskNum(int index) const;
int bottomMask(int index) const;
```

Description:

```
addPolygon(const char* layerName,
defiGeometries* geom,
int *needCbk,
int colorMask
```

Specifies color mask information for a polygon. The default value is 0.

```
addRect(const char* layerName,
defiGeometries* geom,
int *needCbk,
int colorMask,
```

Specifies color mask information for a rectangle. The default value is 0.

```
addPts(const char* viaName,
defiGeometries* geom,
int *needCbk,
int colorMask,
```

Specifies color mask information for points. The default value is 0.

`polyMask(int index)` Returns the color mask information of a polygon by index.

`rectMask(int index)` Returns the color mask information of a rectangle by index.

`topMaskNum(int index)`

Returns the top mask number of the via.

`cutMaskNum(int index)`

Returns the cut mask number of the via.

`bottomMask(int index)`

Returns the bottom mask number of the via.

For more information, see "defiNet" in the *DEF C/C++ Programming Interface Open Licensing Program)*.

defiPath

```
void addMask(int colorMask);
void addViaMask(int colorMask);
int getMask();
int getViaTopMask();
int getViaCutMask();
int getViaBottomMask();
int getRectMask();
```

Description:

`addMask(int colorMask)`

Specifies the color mask information for a path.

`addViaMask(int colorMask)`

Specifies the color mask information for a via.

`getMask()` Returns the color mask information of a path.

`getViaTopMask()`	Returns the top mask number of a via.
`getViaCutMask()`	Returns the cut mask number of a via.
`getViaBottomMask()`	Returns the bottom mask number of a via.
`getRectMask()`	Returns the color mask information of a rectangle.

Additionally, a new enum was added to `defiPath`. The syntax is as follows:

```
enum defiPath_e {
     ...
     DEFIPATH_RECT,
     DEFIPATH_VIRTUALPOINT,
     DEFIPATH_MASK,
     DEFIPATH_VIAMASK
```

The returned value depends on the input data that is read in.

For more information, see "defiPath" in the *DEF C/C++ Programming Interface Open Licensing Program)*.

defiPinPort

```
     void addLayerMask(int mask);
     void addPolyMask(int mask);
     void addVia(const char*via,
          int viaX,
          int viaY,
          int mask);
     int layerMask(int index) const;
     int polygonMask(int index) const;
     int viaTopMask(int index) const;
     int viaCutMask(int index) const;
     int viaBottomMask(int index) const;};
```

Description:

`addLayerMask(int mask)`

Specifies the color mask for a layer on the pin port.

`addPolyMask(int mask)`

Specifies the color mask for a polygon on the pin port.

```
addVia(const char*via,
     int viaX,
     int viaY,
     int mask);
```

Specifies the color mask for a via on the pin port.

```
layerMask(int index)
```

Returns the layer mask number.

```
PolygonMask(int index)
```

Returns the polygon mask number.

```
viaTopMask(int index)
```

Returns top mask number of a via by index.

```
viaCutMask(int index)
```

Returns cut mask number of a via by index.

```
viaBottomMask(int index)
```

Returns bottom mask number of a via by index.

defiPin

```
void addLayerMask(int mask);
void addPolyMask(int mask);
void addVia(const char*via,
     int viaX,
     int viaY,
     int mask);
void addPortLayerMask(int mask);
void addPortPolyMask(int mask);
int layerMask(int index) const;
int polygonMask(int index) const;
int viaTopMask(int index) const;
int viaCutMask(int index) const;
int viaBottomMask(int index) const;
```

Description:

```
addLayerMask(int mask)
```

Specifies the color mask for a layer on the pin.

```
addPolyMask(int mask)
```

Specifies the color mask for a polygon on the pin.

```
addVia(const char*via,
int viaX,
int viaY,
int mask);
```

Specifies the color mask for a via on the pin.

```
addPortLayerMask(int mask)
```

Specifies the layer color mask number on the port.

```
addPortPolyMask(int mask)
```

Specifies the polygon color mask number on the port.

```
layerMask(int index)
```

Returns the layer mask number on a pin.

```
polygonMask(int index)
```

Returns polygon mask number on a pin.

```
viaTopMask(int index)
```

Returns the top mask number of a via by index.

```
viaCutMask(int index)
```

Returns cut mask number of a via by index.

```
viaBottomMask(int index)
```

Returns bottom mask number of a via by index.

For more information, see "defiPin" in the *DEF C/C++ Programming Interface Open Licensing Program)*.

defiTrack

```
void addMask(int colorMask,
     int sameMask);
int firstTrackMask() const;
int sameMask() const;
```

Description:

`addMask(int colorMask)`
`int sameMask);`

Specifies the mask number on the first routing track. If `sameMask` is true, all the routing tracks have the same mask number as the first mask.

`int firstTrackMask()`

Returns the color mask information of the first routing track.

`sameMask()` Indicates whether same mask is defined or not.

For more information, see "defiTrack" in the *DEF C/C++ Programming Interface Open Licensing Program)*.

defiVia

```
void addLayer(const char* layer,
     int xl, int yl,
     int xh, int yh,
     int colorMask);
void addPolygon(const char* layer,
     defiGeometries *geom,
     int colorMask);
int rectMask(int index) const;
int polyMask(int index) const;
```

Description:

```
addLayer(const char* layer,
   int xl, int yl,
   int xh, int yh,
   int colorMask);
```

Specifies color mask information of a rectangle. The default value is 0.

```
addPolygon(const char* layer,
   defiGeometries *geom,
   int colorMask)
```

	Specifies color mask information of a polygon. The default value is 0.
`rectMask(int index)`	Returns color mask for the defined rectangle.
`polyMask(int index)`	Returns color mask for the defined polygon.

For more information, see "defiVia" in the *DEF C/C++ Programming Interface Open Licensing Program)*.