Agilent Technologies

Advanced Design System 2011.01

Feburary 2011
Translating PCB Libraries to ADS

Acknowledgments

SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7-zip - 7-Zip Copyright: Copyright (C) 1999-2009 Igor Pavlov. Licenses for files are: 7z.dll: GNU LGPL + unRAR restriction, All other files: GNU LGPL. 7-zip License: This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful,but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA. unRAR copyright: The decompression engine for RAR archives was developed using source code of unRAR program.All copyrights to original unRAR code are owned by Alexander Roshal. unRAR License: The unRAR sources cannot be used to re-create the RAR compression algorithm, which is proprietary. Distribution of modified unRAR sources in separate form or as a part of other software is permitted, provided that it is clearly stated in the documentation and source comments that the code may not be used to develop a RAR (WinRAR) compatible archiver. 7-zip Availability: http://www.7-zip.org/

AMD Version 2.2 - AMD Notice: The AMD code was modified. Used by permission. AMD copyright: AMD Version 2.2, Copyright © 2007 by Timothy A. Davis, Patrick R. Amestoy, and Iain S. Duff. All Rights Reserved. AMD License: Your use or distribution of AMD or any modified version of AMD implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies.User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. AMD Availability: http://www.cise.ufl.edu/research/sparse/amd

UMFPACK 5.0.2 - UMFPACK Notice: The UMFPACK code was modified. Used by permission. UMFPACK Copyright: UMFPACK Copyright © 1995-2006 by Timothy A. Davis. All Rights Reserved. UMFPACK License: Your use or distribution of UMFPACK or any modified version of UMFPACK implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License

as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. UMFPACK Availability: http://www.cise.ufl.edu/research/sparse/umfpack UMFPACK (including versions 2.2.1 and earlier, in FORTRAN) is available at http://www.cise.ufl.edu/research/sparse . MA38 is available in the Harwell Subroutine Library. This version of UMFPACK includes a modified form of COLAMD Version 2.0, originally released on Jan. 31, 2000, also available at http://www.cise.ufl.edu/research/sparse . COLAMD V2.0 is also incorporated as a built-in function in MATLAB version 6.1, by The MathWorks, Inc. http://www.mathworks.com . COLAMD V1.0 appears as a column-preordering in SuperLU (SuperLU is available at http://www.netlib.org). UMFPACK v4.0 is a built-in routine in MATLAB 6.5. UMFPACK v4.3 is a built-in routine in MATLAB 7.1.

Qt Version 4.6.3 - Qt Notice: The Qt code was modified. Used by permission. Qt copyright: Qt Version 4.6.3, Copyright (c) 2010 by Nokia Corporation. All Rights Reserved. Qt License: Your use or distribution of Qt or any modified version of Qt implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the
terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies.User
documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. Qt Availability: http://www.qtsoftware.com/downloads Patches Applied to Qt can be found in the installation at: $HPEESOF_DIR/prod/licenses/thirdparty/qt/patches. You may also contact Brian Buchanan at Agilent Inc. at brian_buchanan@agilent.com for more information.

The HiSIM_HV source code, and all copyrights, trade secrets or other intellectual property rights in and to the source code, is owned by Hiroshima University and/or STARC.

Errata The ADS product may contain references to "HP" or "HPEESOF" such as in file names and directory names. The business entity formerly known as "HP EEsof" is now part of Agilent Technologies and is known as "Agilent EEsof". To avoid broken functionality and to maintain backward compatibility for our customers, we did not change all the names and labels that contain "HP" or "HPEESOF" references.

Warranty The material contained in this document is provided "as is", and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this documentation and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license. Portions of this product include the SystemC software licensed under Open Source terms, which are available for download at http://systemc.org/ . This software is redistributed by Agilent. The Contributors of the SystemC software provide this software "as is" and offer no warranty of any kind, express or implied, including without limitation warranties or conditions or title and non-infringement, and implied warranties or conditions merchantability and fitness for a particular purpose. Contributors shall not be liable for any damages of any kind including without limitation direct, indirect, special, incidental and consequential damages, such as lost profits. Any provisions that differ from this disclaimer are offered by Agilent only.

Restricted Rights Legend U.S. Government Restricted Rights. Software and technical data rights granted to the federal government include only those rights customarily provided to end user customers. Agilent provides this customary commercial license in Software and technical data pursuant to FAR 12.211 (Technical Data) and 12.212 (Computer Software) and, for the Department of Defense, DFARS 252.227-7015 (Technical Data - Commercial Items) and DFARS 227.7202-3 (Rights in Commercial Computer Software or Computer Software Documentation).

About Translating PCB Libraries

Many of today's design engineers prefer to use a combination of Agilent ADS and enterprise PCB EDA tools to take advantage of the strengths of both design environments. Because of this desire to use multiple tools, Agilent Technologies has developed the *Intermediate File Format* translators as a method for transferring designs between the Advanced Design System (ADS) and PCB design environments.

Intermediate File Format (IFF) is an ASCII file format that is both platform and application independent. The file has a simple, line-oriented command structure with a fairly rich set of constructs, thus simplifying design transfer. The IFF translators offered by Agilent Technologies provide a means for transferring IFF files between Advanced Design System and third-party electronic design automation (EDA) tools such as Boardstation from Mentor Graphics Corporation and Allegro from Cadence Design Systems.

Advanced Design System

Advanced Design System has been developed specifically to simulate the entire communications signal path. This unique solution integrates the widest variety of proven RF, DSP, and electromagnetic design tools into a single, flexible environment. Building on years of expertise developing new technologies for our EDA tools, Advanced Design System provides a broad range of high-performance capability. This makes it easy to explore design ideas, then model the electrical and physical design of the best candidates.

Boardstation

Boardstation from Mentor Graphics Corporation is an integrated environment for generating PC boards. Boardstation contains both a schematic tool (Design Architect), and a physical layout tool. The IFF interface to Boardstation will specifically only work with Library Management System (LMS) style libraries, and the special library that is provided with Mentor Graphics' RF Architect product. RF Architect is a special add-on to the Boardstation environment, that allows the standard ADS palette of components to be used within Mentor Graphics. Additionally, it allows the user to create parameterized layout instances, which is a requirement when generating transmission line components in a PC Board environment.

Allegro

Allegro from Cadence Design Systems is an enterprise level PCB design environment supporting schematic design capture and board physical layout. The Allegro IFF interface to ADS uses the same LMS-style intermediate files as Mentor Boardstation. Optional products support the import and creation of ADS-style schematics into Allegro through the IFF link.

The IFF Translator

The IFF translator provided by Agilent Technologies is an EDA framework integration software product that stores circuit component and connectivity information. This product enables you to exchange design information between ADS and other EDA frameworks that provide an IFF interface. Agilent's IFF interface enables you to generate IFF files from ADS schematic information as well as receive IFF files from other design environments that support IFF translation.

IFF Interface Major Benefits

The IFF interface enables you to translate schematic information between Advanced Design System and PCB design environments, resulting in the following benefits:

- Avoids re-entry of designs
- Helps eliminate the possibility of errors in design re-entry
- Time savings

IFF Interface Major Features

Key features of the IFF Interface enable you to perform the following:

- Import IFF files into ADS from PCB environment and vice versa
- Export IFF files from ADS to PCB environment and vice versa
- Preserve circuit component and connectivity information during transfer

Intended Audience

The audience intended for this documentation consists of CAD system administrators and CAD librarians who want to make their PCB environment libraries available for use in ADS. It is assumed that the users have some knowledge of both ADS and the PCB environment. It is also assumed that the users have some knowledge about component modeling and simulation.

Main Requirements

To enable the successful IFF translation of an RF Board design between ADS and PCB environments, you must ensure that the following requirements are met:

- The link between IFF and the PCB design environment is available and installed. The IFF translator for the PCB environment is supplied by the PCB EDA vendor.
- The component libraries used for creating the schematic in ADS and the PCB schematic editor have been implemented to support translation via IFF. This documentation will provide detailed information on library requirements. If equivalent libraries have not been set up for the PCB components in ADS, the PCB components will not be usable in simulations.

> **Note**
> Agilent Technologies supports the IFF Import and Export tools for ADS. The PCB EDA vendor is responsible for their product, which includes ADS equivalent part libraries, as well as IFF Import and IFF export . ADS and third-party IFF products are licensed separately. Contact your PCB EDA vendor for information on IFF translation tools for your target environment.

What's in this Documentation

The goal of this documentation is to help you get started, providing relevant examples that teach you how to set up and use the software, and showing you where you can get more information as you need it. This documentation contains the following:

- *Overview of Library Translation* (iffml) describes the overall process of how to move an LMS-style library into ADS.
- *Setting Up Your Environment* (iffml) describes the process to activate library translation tools in ADS.
- *Exporting Library Parts from PCB Environment* (iffml) describes the procedure for exporting LMS-style part information from your PCB library environment to IFF files.
- *Importing Library Parts from PCB Environment into ADS* (iffml) describes the use of the library import toolkit for initial library creation.

About ADS Design Kits

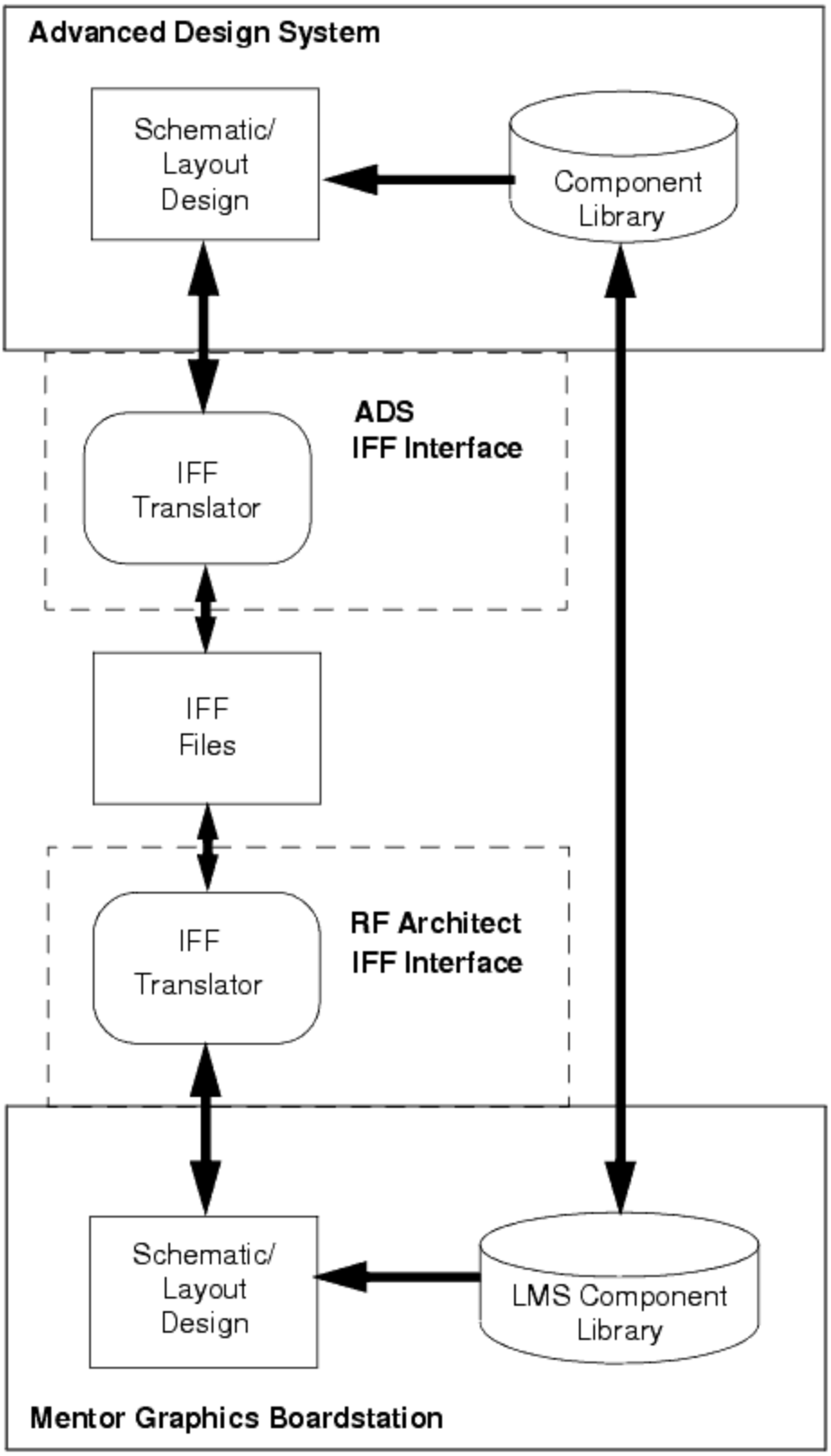
Once your PCB component library has been translated into Advanced Design System, the last step in the process is to modify your library so that it will function in a redistributable format that is file system independent. Agilent Technologies provides the tools for creating distributable ADS libraries referred to as *ADS Design Kits* . Even though design kit creation is a necessary step in the process, the creation, installation, and usage of ADS Design Kits is beyond the scope of this documentation.
For more information on converting your library into an ADS Design Kit, refer to your ADS Design Kit documentation or consult your Agilent Technologies sales representative.

Overview of Library Translation

Once the main requirements have been satisfied, your schematic can be transferred between Advanced Design System and your PCB environment. Importing and exporting can be initiated from either EDA design environment.

The IFF Schematic and Layout Transfer Use Model figure describes the general use model for translating a design using the IFF Interface as it applies to Advanced Design System and PCB packages, using Mentor Boardstation as a typical example. The link between the two EDA environments is established via the IFF Interface. The component libraries in both ADS and the PCB environment must be compatible to support an IFF translation.

Figure: IFF Schematic and Layout Transfer Use Model



Creating the ADS Component Library from the PCB Environment

The IFF Schematic and Layout Transfer Use Model diagram shows that, in order to do an IFF transfer of a layout and/or a schematic, you must have a component library in the PCB environment, and a component library that matches it in Advanced Design System (ADS).

The ADS library system is not identical to the PCB environment library system. Nor are the component definitions in ADS identical to the PCB environment component definitions.

What the IFF translator is doing when it translates a schematic or a layout is using a component definition that has been designated the same between ADS and the PCB environment.

These component definitions have many attributes in common. The common attributes include such things as the following:

- The size of the symbol
- The shape of the symbol
- The number of pins on the symbol
- The size of the layout footprint
- The layers that make up the layout footprint
- Properties of the component, such as the part number.

But, the PCB environment and ADS libraries do not share the same database, and they can have unique attributes that are specific to either ADS or the PCB environment (such as the simulation model).

The IFF translation process is assuming that the the PCB environment library was created first. This means that the ADS library must have been made from the the PCB environment library.

The ADS component library is an image of the PCB environment library. This leads to the following rules:

- New library components should always be made in the PCB environment
- All library changes should be made in the PCB environment
- The ADS representation will attempt to look as much like the source library as is feasible.

While it is possible to take ADS library components and transfer them to the PCB environment, this is not considered a good practice. Transferring libraries to the PCB environment will not be covered in this documentation. Library Translation assumes that all transfers are being made from the PCB environment to ADS.

A Brief Overview of a Typical PCB Library

Since one of the *rules* is that the ADS representation will attempt to look like the source library, it is important to designate what the important aspects of typical PCB library are that must be duplicated. The Mentor LMS library will be used as an example, since much

of the ADS side of the PCB/ADS link was developed based on Boardstation/LMS characteristics.

Note
This information is not official Mentor Graphics documentation. It is being presented based on observations made by using Mentor Graphic's tools. For more information on LMS libraries, consult Mentor Graphics documentation.

Library Management System (LMS) is Mentor Graphics method for creating a company-wide set of libraries that contain all of the information needed to develop, release, and maintain all of the parts that are used to manufacture a board design. A library being managed via LMS library will contain:

- Catalogs
- Parts
- Components
- Symbols
- Geometries

LMS is meant to manage libraries across multiple sites, and would typically be company wide. It would not be typical for a single site to use multiple sets of libraries that are managed via separate LMS setups. Parts from multiple vendors are managed by a single LMS setup.

Catalogs

A Catalog is a grouping of entries that share common attributes. Often, you will see a catalog referred to as a family of parts. Also, within Mentor Graphics, each catalog could be considered to be the listing of all of the elements within a single library. The catalog is actually a text file. Each line in the text file represents one catalog entry. Each entry is unique based on an index number that cannot be duplicated within all of the libraries that are being managed through LMS. For LMS managed libraries, this unique index is the part number. Entries within a catalog are normally referred to as Components within the context of Mentor Graphics.

Parts

A part is the complete set of information used to identify, describe, and distinguish any object that might be used within an electronic design. In LMS, a part is defined by an entry in a catalog which specifies and/or references all the data describing the various aspects of a part.

A part can have a number of attributes associated with it. All parts in an LMS library that you wish to transfer to ADS must have the following attributes:

- Part Number: The part number is the unique index that designates the catalog entry

of the part.

- Geometry: The geometry parameter specifies the footprint that a schematic instance is represented by in a layout.
- Reference Designator: The reference designator is a unique ID that is associated to a part placed within a schematic design. The reference designator is used to figure out which geometry a schematic instance belongs to in a layout. This value is undefined in a schematic design until a design is packaged.

Parts may also have electrical parameters (e.g. capacitance on a capacitor), but this is not required. It is very important to realize that, from an RF perspective, Mentor Graphics is primarily a manufacturing environment, not a simulation environment. Many Mentor Graphics parts will have no simulation parameters associated with them.

Parts can have as many Symbols or Geometries associated to them as the librarian wishes. Usually, a part will have a single geometry, and multiple symbols that will represent the part in a schematic with different rotations.

Parts can represent packages. A packaged part means that the part number has multiple distinct schematic instances that can be traced back to the single part number (this tracing is done via the Reference Designator). Packaged parts will be represented in Mentor by having several distinct symbols, with each symbol having pins that link to specific pins of the geometry file. The symbols do not need to be identical. Note also that, from a database standpoint, the symbols are stored separately from the part. Many parts can utilize the same schematic symbol.

A given part will usually inherit properties that have been set up in its catalog. Thus, a family of capacitors will have a capacitor catalog, that could specify what the appropriate symbols are for the entire family of capacitors. It is possible, though, to override the catalog entries for particular part numbers in the family.

Components

In the context of Mentor Graphics and ADS, a component is a user generated part. Rather than being a member of a catalog file, a component is something that is created in a library that is essentially not managed by LMS. Components would consist of one or more schematic sheets, layout, symbols for hierarchical representation, and, potentially, logical or behavioral simulation models. When a user transfers their schematic and layout designs between ADS and Mentor Graphics, they are transferring components.

Symbols

A symbol is a graphical representation of a part for use within logic based tools, such as schematic capture or VHDL code generation. A symbol will consist of symbol pins, symbol body graphics, symbol pin properties, and symbol body properties. The symbol pin properties and symbol body properties are independent of the catalog entry properties. A symbol can be used by more than one part. For example, if the parts C0123 and C0124

are contained within the catalog "capacitors", it is possible for both of those parts to be logically represented by the same symbol, "cap". In addition to the symbol properties, it is also possible to add in text. Text can be interpretive (e.g. display the value of PART_NO), or non-interpretive (e.g. put a label, "P", next to the positive symbol pin, which is named "pos").

Because Mentor Graphics schematics are used as documentation, multiple symbols will be used as opposed to simply rotating a symbol. This is done so that the location of text in the symbols can be controlled.

Symbols are linked to Geometries via a mapping file. As was noted, Parts may have more than one logical symbol to represent them within a schematic. This means that there is not necessarily a one-to-one match up between the symbol pins, and the leads of the geometry footprints. The mapping file contains the information that links a given symbol pin to a given footprint. The map file is specified in the catalog entry, and can be different for every part number that uses a given symbol (although the map file can be shared if multiple part numbers share the same symbols and footprints).

Geometries

A geometry is the physical description of an object used in a board design. A geometry consists of a type designation, a name, and the graphics which define the geometry's size and shape. Geometries are required to package logical symbols from a schematic to a physical part in a layout. A common term used to describe a geometry that represents a part is a footprint; geometry and footprint will be used interchangeably.

Parts may have multiple geometries that can be associated to them. For example, for a through-hole process, one geometry may be needed. For a surface mount process, a DIP package might be needed instead. This one part could then have multiple allowed geometries (although the software would disallow one of them during packaging, based on the process being used).

Geometries can also be shared amongst multiple parts. For example, part numbers C0123 and C0124 could both use the same through hole geometry, CAP_TH.

A mapping file is used to link the geometry to a symbol. The mapping file contains information that specifies how a geometry pin matches up to a symbol pin. The mapping file is organized to specify the name of the symbol, the name of the symbol pin, and the number of the geometry pin. The mapping file can be unique to a given part number, or shared amongst multiple part numbers, depending on whether the symbol name, the symbol pin names, and the geometry numbers match up correctly or not.

A Brief Overview of How an LMS-style Library is Represented in ADS

Based on the overview of LMS managed libraries in Mentor, a description of how each of the elements of a Mentor Graphics library are represented in ADS is in order. This section contains descriptions of how the Mentor library elements are represented, and why they were represented this way. Equivalent elements in other EDA vendor libraries are mapped onto the same ADS elements. The relevant elements are as follows:

Parts in ADS

ADS contains a construct for representing a part. Like in the PCB environment, a part can be thought of as the set of information necessary to identify, describe and distinguish an object in an electronic design.

In Mentor Graphics, the part description is handled by an entry in a catalog file that contains all of the necessary references for the part. ADS is organized slightly differently. In ADS, all parts are described by program calls to ADS' extension language functions. During part creation, these function calls will typically be collected into a single file, which will be loaded into memory sometime prior to usage of the part in a design. When the part description has stabilized, the files can be collected into a single file; the ADS library manager will then dynamically load the part descriptions as they are needed. The part description that is loaded in memory is called an Item Definition in ADS.

ADS does not have an "index" field to designate which entry to use. ADS has a name field. Because there is a single name space for the Item Definitions, each part must be given a unique name. The logical assumption is that the part number is simply used as the part name, so that a unique name is created. Unfortunately, this is incorrect.

The methodology of using IFF for doing the part transfers between ADS and Mentor Graphics pre-dates LMS being the defacto standard for library management in Mentor Graphics. Additionally, there are still numerous libraries that may need to have parts translated (genlib for example) where the parts are not defined as entries in a catalog file. As a final additional problem, ADS only supports one symbol per part, unlike Mentor Graphics.

To avoid problems, and fit in the "historical" mode, the ADS part name is derived by using the component name (the mgc_comps parameter in the catalog entry), the symbol name, and the part name. These three elements are concatenated together with underscores. For parts that are not in LMS catalogs, the part will simply be the component name, or the component name concatenated to the symbol name.

ADS has a single simulator (eesofsim) that is associated with it. The simulator itself

actually has two separate parts to it, a system level simulator (adsptolemy), and a circuit level simulator. At present, the expectation is that any devices that are imported into ADS from the PCB environment will be set up to work with the circuit level simulator.

When a part is imported into ADS, it does not have a definition for use with the simulator. It is up to the librarian to add whatever details are necessary for the part to simulate. Setting up a part is beyond the scope of this document. The PCB environment typically does not have any information that is readily usable for an RF simulation setup. It is possible to reuse the electrical properties that are defined in the MGC catalog entry (for example, the resistance on a resistor), but for RF, do NOT set up your devices as ideal components. At present, there is no way of automatically creating an accurate simulation model by looking at the geometry and device parameters.

By default, ADS expects that a single ADS schematic symbol will be associated to a single layout object. If the ADS package capability is enabled, as of ADS 1.5, the IFF importers and exporters for Mentor Graphics and ADS are unable to synchronize to use the ADS packaging information, and it will not be used. If you have multi-pack parts, the current recommendation is ADS for schematic editing only, and to use Mentor Graphics for doing all layout. If you do not have multi-pack components, all parts will have their Mentor Graphics geometry associated to them properly during the library import. See *Importing Library Parts from PCB Environment into ADS* (iffml) for more information on how this is accomplished.

The reference designator in ADS is essentially non-existent. ADS parts will all receive an instance prefix. The REF property can be used as the instance prefix in ADS. However, this will not operate identically to the Mentor Graphics usage of REF. In ADS, the instance name will be the reference designator. In a hierarchical design, each sub circuit still retains the same instance ID. Within Mentor Graphics, the reference designator is different for all instances, even if they are in the same subvariety. When ADS designs are transferred to Mentor Graphics, it is recommended that Mentor Graphics be allowed to re-assign the reference designators.

Catalogs in ADS

ADS has a tool called the Library Browser. This tool is responsible for dynamically loading part definitions at the request of the design environment. The Library Browser uses files that are made up of a set of item definition files. These files are fairly analogous to the LMS catalog file. When a part is requested, the Library Browser will search its files for the first definition that matches. The files that are used are based upon a search path that is set up so that only a desired set of components will be available. For more information about how to set these files up after importing from the PCB environment, refer to *Importing Library Parts from PCB environment into ADS* (iffml).

Components in ADS

In ADS, the difference between a part and a component is essentially that a part is a

system wide component that is not be user editable. All ADS parts contain the same database structure. During the library import process, the parts sent from Mentor Graphics are not different from the parts from an end schematic design.

Symbols in ADS

As with Mentor Graphics, a symbol is a graphical representation of a part for use within logic based tools, such as schematic capture or VHDL code generation. After importing a library from Mentor Graphics, the symbol that is generated for a part should look as close as possible to the original Mentor Graphics symbol in terms of pin placement and symbol body graphics. ADS does allow the placement of fixed text, but does not have any interpreted symbol text. All instance specific symbol text is applied to fixed locations of the symbol through the schematic editor.

ADS does not support the concept of a mapping file. Symbols are linked to Geometries in ADS based on the symbol pin number and the geometry pin number. This does lead to problems if a geometry is shared between parts that have different pin outs. As was noted, Parts in ADS can only have a single symbol associated with them. This leads to the parts being "split" in ADS. For example, if there is a part in Mentor Graphics called 11111, which is a part type "CAP", and has two symbols called "cap_h" and "cap_v", there will be two separate parts created in ADS, called CAP_cap_h_11111, and CAP_cap_v_11111. The definitions for these parts will be identical, with the exception of the symbol that is used to represent the part in the schematic. Each symbol will have its own geometry cross mapping built into the symbol itself.

Because the geometry to symbol cross mapping can be different for each part/symbol pair, it is not possible to guarantee a correct import by reusing the same symbol between different parts. ADS is capable of doing this, but it could lead to problems that are difficult to diagnose if the specialized cases that don't obey the rule that the same symbol should always map identically, independent of the part. This does mean that if one symbol is used in Mentor, it could be duplicated hundreds of times over in ADS.

Geometries in ADS

ADS allows for two ways of representing a layout for a part. Either a fixed layout can be used, or a parameterized layout can be used. A parameterized layout means that AEL (Application Extension Language) code is written to represent the layout. A fixed layout is simply a database element.

When geometries are imported from Mentor Graphics, they are imported into fixed layouts. In many cases, it would be appropriate to simply point the ADS part that uses the geometry to that fixed layout. However, Mentor Graphics does allow multiple geometries to be used for some parts. The goal of the specialized LMS library import (see *Importing Library Parts from PCB Environment into ADS* (iffml)) is to minimize the amount of manual work required by a librarian, as well as to minimize the amount of internal ADS library

knowledge needed by a Mentor librarian.

To cover the cases where multiple geometries can be specified based on the GEOM parameter, parts that are imported from Mentor Graphics are actually represented as parameterized layouts. A special AEL function has been written for ADS which is able to open a fixed layout database and draw the graphics in the database into the parameterized layout instance. Currently, this function only supports drawing non-hierarchical fixed layouts. This can be a problem where pad stacks are represented hierarchically in Mentor Graphics geometries.

Menus in ADS

As was noted in the section Catalogs in ADS, ADS has a tool called the Library Browser that is responsible for dynamically loading part definitions into memory for use with the ADS design environment. This tool also provides a graphical interface for selecting which parts are to be placed. Along with the concatenated Item Definition File (IDF), there will also be a record file, which contains a listing of the parts in the IDF file, as well as some descriptive information about each part.

Unfortunately, the record file is not quite the same as the Mentor Graphics LMS menu files. Both are text based, but, in ADS, there is no capability to have nested menus. Parts can be grouped together to form a single tree selection (a library). Additionally, groups of libraries can be grouped together (a library of libraries).

Customization Code in ADS

ADS has a C-like interpreted language called AEL (short for Application Extension Language). AEL allows the design tools to be modified. Menus can be added. Database traversal routines can be written that perform specialized checks.

It may be necessary to write certain functions in order to support the PCB environment library in ADS. This can include writing netlisting functions for specialized components, and writing code that will alter certain properties on instances in order to do IFF export. If this is required, you will currently need to contact Agilent EEsof Solution Services in order to get special training on the process involved in this. This topic is beyond the scope of this documentation. In most cases, it should not be necessary to write your own customization code.

Steps for Transferring an LMS-style Library to ADS

The following is a short check list for transferring an LMS-style library from the PCB environment to ADS. Each of these topics is covered in fuller detail within its own section. Once again, Mentor Boardstation and LMS is used as an example. The process for other

PCB environements is similar.

1. Make sure your environment is set up properly to do LMS exports from Mentor Graphics, and LMS imports into ADS. Details on this process are covered in *Setting Up Your Environment* (iffml).
2. Export your library parts from Mentor Graphics. The files will be exported to two separate IFF files, one for the geometries (LMSgeoms.iff), one for the symbols (LMSsymbols.iff). There will be an additional third file that contains the information about how the parts, symbols, and geometries are associated to each other (parts_mapping_file.txt). The procedure for making these files is covered in *Exporting Library Parts from PCB Environment* (iffml).
3. The next step is to actually import the IFF files created in step 2 into ADS. Because there are certain properties that are inherent to all LMS managed libraries, there are some extra automation steps that can be done in addition to the standard IFF import. A special LMS import process is available to handle this extra automation. In *Importing Library Parts from PCB Environment into ADS* (iffml), the user interface for the LMS import process is covered, as well as what the LMS import process is doing in addition to the standard IFF import.
4. Once the basic symbols and footprints have been imported, simulation models can be added to each component. There are several ways to do this, and significant automation is possible with some custom coding. This is beyond the scope of this document, but you can contact Agilent EEsof Solution Services for customized assistance.
5. Once the library has been imported and verified, it is necessary to set things up so that all ADS designers can see and use the library. This process is not covered in this documentation. For more information on converting your library into an ADS Design Kit, refer to your ADS Design Kit documentation or consult your Agilent Technologies sales representative.

Setting Up Your Environment

In order to use the Library Import Toolkit in Advanced Design System (ADS) and the PCB environment, there is a certain amount of setup required. This section is broken down into two main areas:

- Setting Up the LMS Toolkit Software for ADS
- Setting up the LMS Toolkit Software for the PCB Environment

After covering the information in this section, you should be ready to begin the process of transferring your LMS libraries from the PCB environment to ADS.

Setting Up the Library Import Toolkit in ADS

The Library Import toolkit is an add-on capability to the standard ADS product. Since most customers are not using ADS in conjunction with PCB library management, the software is disabled by default. However, the code is delivered on the standard ADS product CDs.

Note
The Library Import toolkit add-on product is not licensed, but must be enabled for use.

Enabling the Library Import Toolkit

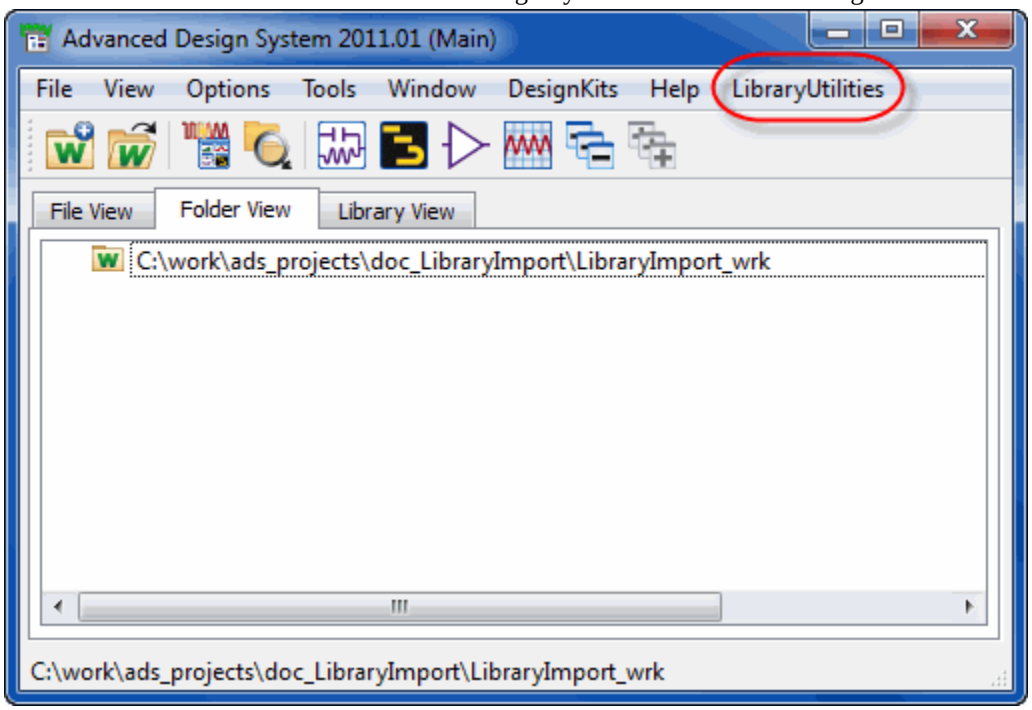
The LMS Toolkit is delivered as an ADS Addon. To enable the LMS add-on, perform the following steps:

1. From the main ADS window, choose **Tools** > **Manage ADS AEL Addons**.
2. Below the ADS Installation Addons section, select the appropriate PCB Library Import Tools addon. Select either the Import or User Tools option of the desired vendor.

   Note
   Import Tools option include User Tools.
3. Click **Close**.
   It is recommended to restart ADS to ensure a complete activation of the toolkit.

ADS Main Window with LMS add-on shows the menus that you should now see in your ADS main window.

Figure: ADS Main Window with LMS add-on



Setting up the LMS Toolkit Software for the PCB environment

Setup for each PCB environment is different, and is described in the vendor-supplied documentation. Please consult your PCB environment manuals or sales/support representative for exact details.

Exporting Library Parts from PCB Environment

The library export process is different for each vendor's PCB environment, and describing the details of the process is beyond the scope of this document. Each package, while it may use a different sequence of steps, produces the same standard output.

Required Output Files

LMSsymbols.iff - This file contains the symbols associated with the library parts.
LMSgeoms.iff - This file contains the geometries associated with the library parts.
parts_mapping_file.txt - This file contains a single entry for each part that was exported to IFF. Each entry contains information about which symbols and which geometries are associated to each LMS part.

Limitations for Library Parts in ADS

LMS parts that contain multiple symbols in the mapping file and have distinct COMP properties, are not supported in this release. During export of these LMS parts, LTK generates errors for parts, which cannot be correctly translated to ADS.

Importing Library Parts from PCB Environment into ADS

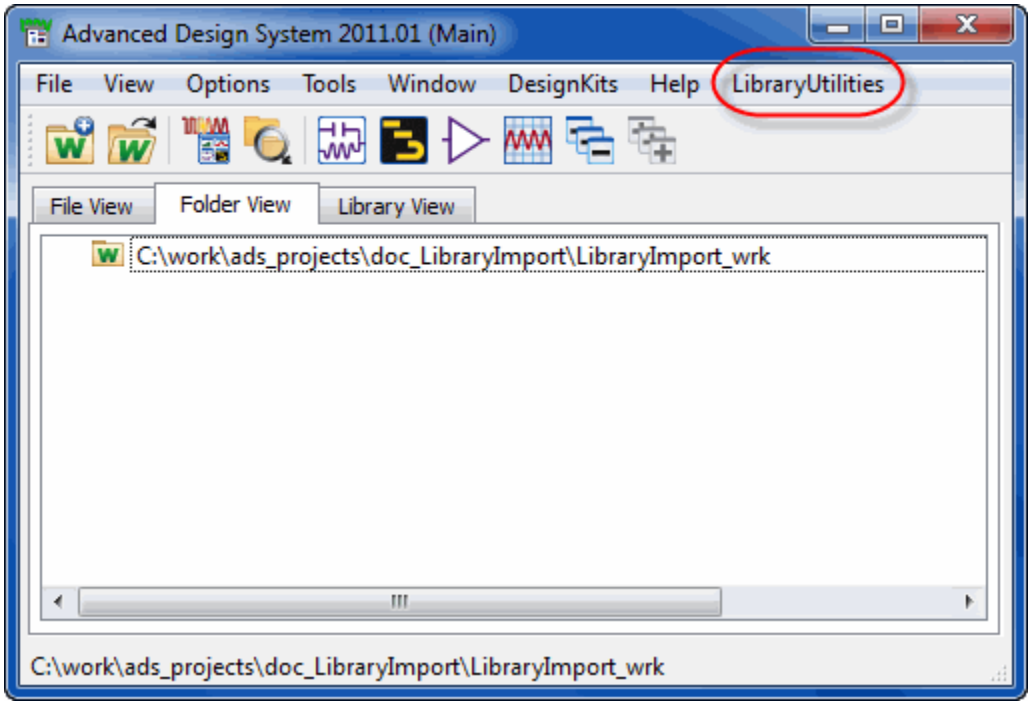
This section describes the procedure to use to import into ADS the IFF files that were generated by the PCB library administrator. It also describes the special processing that is done during the import process PCB parts work for ADS board layout.

Note
The Library Import toolkit, which is included with the ADS release, must be enabled before use. See *Setting Up the LMS Toolkit Software for ADS* (iffml) for toolkit setup instructions.

Procedure for Importing PCB Library IFF Files Into ADS

1. Start an ADS session. On the ADS main window, you should see the **LibraryUtilities** menu if you have enabled the Library Import toolkit.

   Figure: ADS Main Window at Startup

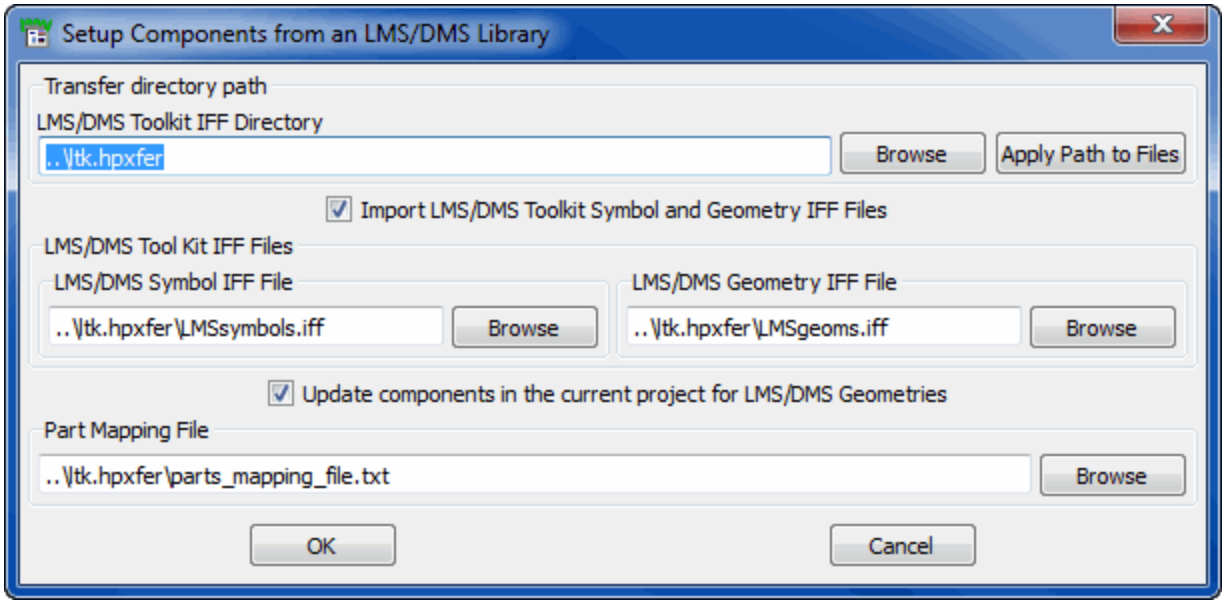


2. The PCB library parts are best imported into a clean, empty workspace. To simplify the import and library customization process, it is helpful to import each category of parts (resistors, capacitors, etc) into a separate workspace. They can be merged later into a single library if desired. You can specify a different name for the import workspace.

   When you create the new workspace, set the default layout units and resolution to match that of the PCB environment. This will make testing a bit easier.

3. The next step is to set up the import. From the ADS Main window, choose **LibraryUtilities > Import LMS/DMS Library Parts** to open **Setup Components from an LMS/DMS Library** for Mentor library import. Specify the component fields (described in detail below) and click **OK** to begin the import. You can choose amongst other PCB vendor specific import dialogs.

Figure: Setup Components from an LMS/DMS Library



Component Fields

The following section describes the different fields of **Setup Components from an LMS/DMS Library** dialog box.

LMS/DMS Toolkit IFF Directory

This option enables you to specify the directory where the the PCB environment IFF export placed the library IFF files. As was noted in *Exporting Library Parts from PCB environment* (iffml), the library export toolkit in the PCB environment will create a directory (by default named *ltk. hpxfer*) which contains the symbol and geometry IFF files, as well as a part mapping file. Leave all exported files in a single directory. By default, the export directory is assumed to be one directory level up from the current working directory (note that, when ADS opens a workspace, it changes the current working directory to be the workspace's directory). If the value of this field is changed and the Apply Path to Files button is pressed, the LMS/DMS Symbol IFF File field, the LMS/DMS Geometry IFF File field, and the Part Mapping File field will be updated to use the new path. This update is done via text replacement, so if a path was specified for any of these other fields that was different from the initial PCB export directory, the user specified path will be respected.

Import LMS/DMS Toolkit Symbol and Geometry IFF Files

This checkbox allows you to decide if you wish to import the symbol and geometry IFF files. If a huge amount of components is being imported, it may be necessary to split the import process and the part mapping process to avoid memory problems. Normally, this option should be set to true.

LMS/DMS Symbol IFF File

This is the IFF file that contains the symbol data. At present, the IFF file name will default to being LMSsymbols.iff.

LMS/DMS Geometry File

This is the IFF file that contains the geometry data for the library parts. The default name of LMSgeoms.iff should not need to be changed.

Update Components in the Current Project for LMS/DMS Geometries

This checkbox allows you to decide if the special library import post processing will be done. The post processing will link the library parts to their geometries, and modify the symbol pin numbers so that ADS design synchronization will work with those geometries. If you wish to handle all of the component setups manually, you may wish to omit this step by deselecting the checkbox. Also, if a huge number of components (i.e. more than 2000 distinct parts) are being imported all at once, it may be desirable to do the post processing as a separate step to avoid memory problems during the import.

Part Mapping File

The part mapping file contains information for the library import post processor that is used to link the imported LMS parts to their geometries. The LMS librarian export will generate a file called parts_mapping_file.txt that contains this information.

Once the component fields have been set up, click **OK**. If the Import LMS/DMS Toolkit Symbol and Geometry IFF Files field is selected, the geometry file and symbol file will be imported. The geometry file will be imported first, followed by the symbol file. It is very important that the library import toolkit be used instead of the standard **File > Import > IFF**. There are special options that are set for the IFF importer when a PCB library file is imported. For the PCB library import, there will not be any feedback when the IFF imports are completed. A log file (ltkimport.log) should be consulted to view error messages. In ADS, this is now a non-graphical process, so the only indication that something is happening will be the stop watch icon, that will prevent you from doing anything within ADS.

If the Update Components in the Current Project for LMS/DMS Geometries field is also set, the post processing will be done. It is not necessary to do the IFF imports and the post processing at the same time, but this is normally the desirable way to do things. At this time, the item definition file for the part will also be updated to contain the information needed for layout and design synchronization to work properly.

What You Should Have After the Import is Completed

After the IFF files are imported, and the part mapping is complete, you should check the local library in your import workspace. Within the library, you should have the following:

- One cell for each distinct geometry that is used by the LMS components.
- One or more cells for each LMS part that was imported. The name of each cell will be a concatenation of the mgc_comps value, the name of the symbol, and the name of the part number. Each part cell includes an itemdef.ael file containing the item definition, which corresponds to the PCB library catalog definition. The item definition file will contain information relevant to simulating the part in ADS, as well as information about how to physically represent the part.

What is the LMS Toolkit Post Processor Doing?

In addition to conveniently importing both the layout and symbol IFF files simultaneously, the LMS import also has an option to do post processing based on a file, parts_mapping_file.txt. Sample parts_mapping_file.txt shows a sample parts_mapping_file.txt file. The file is divided up into four separate columns. The first column is the part number. Note that, for LMS to be used with ADS, it is recommended that the MGC_INDEX property and the PART_NO property should be identical. Next is the component field, which comes from the value of mgc_comps property. Next is a list of available symbols for the part number. Finally is a list of available geometries for a part number.

Figure: Sample parts_mapping_file.txt

```
# LMS Parts Component/Symbol/Geometries Mapping File for Conversion to EEsof
# Generated: 2001/04/23 at 15:09:20

################################################################################
# MGC_INDEX      Component      Symbol      Geometries
################################################################################

CK05BX100K_KEMET  capacitor  ["ANSI_0", "ANSI_270"] ["CK05"]

CK05BX100M_KEMET  capacitor  ["ANSI_0", "ANSI_270"] ["CK05"]

CK05BX104K_KEMET  capacitor  ["ANSI_0", "ANSI_270"] ["CK05"]

CK05BX104M_KEMET  capacitor  ["ANSI_0", "ANSI_270"] ["CK05"]

CK05BX150K_KEMET  capacitor  ["ANSI_0", "ANSI_270"] ["CK05"]

CK05BX150M_KEMET  capacitor  ["ANSI_0", "ANSI_270"] ["CK05"]

CWR11J-156_KEMET  pol_capacitor  ["ANSI_0", "ANSI_270"] ["CWR11_D"]
```

Because the name of the geometry (e.g. "CK05") is different from the part number (e.g. "CK05BX100K_KEMET"), they will have separate design files created for them in ADS. This means that ADS needs a way of figuring out which geometry goes with a given ADS part number. Rather than forcing you to manually set up the geometry associations, an automated process was set up that can take care of it for you. This is the LMS post processing step.

Figure: Item definition file before post processing

```
set_simulator_type(-1);
create_item("capacitor_ANSI_0_CK05BX100K_KEMET","capacitor_ANSI_0_CK05BX100K_KEMET","X",16,-1,NUL
L,"Component Parameters",NULL,"%43?global %;%d:%t %# %44?0%:%31?%C%:_net%c%;%;%e
%b%r%8?%29?%:%30?%p %:%k%?[%1i]%;=%p
%;%;%;%e%e","capacitor_ANSI_0_CK05BX100K_KEMET","%t%b%r%38?%:\n%39?all_parm%A%:%30?%s%:%k%?[
%1i]%;=%s%;%;%;%e%e%;","capacitor_ANSI_0_CK05BX100K_KEMET",3,NULL,0,
create_parm("PART_NO","",577,"StdFileFormSet",-1,prm("StdForm","CK05BX100K_KEMET")));
set_design_type(1);
library_group("BPL_RLS_LIB", "BPL_RLS_LIB", 1, "capacitor_ANSI_0_CK05BX100K_KEMET");
library_group("*", "*", 1, "capacitor_ANSI_0_CK05BX100K_KEMET");
```

Item Definition File Modifications

Item definition file before post processing shows the AEL file created for the component capacitor_ANSI_0_CK05BX100K_KEMET without doing the post processing. Item Definition File after post processing shows the same file after the post processing has been completed.

Note
For more information on the create_item command, consult the AEL programmer's guide. This section will only be discussing the parts of the create_item that are relevant to LMS components.

The first obvious difference should be that the file has been "pretty printed". The create_item arguments have been split, so that only one argument for the command is on each line. There is a reason for this - you may need to edit the file by hand, or may wish to write your own scripts that will edit the files. The reformatting makes this much simpler. The reformatting has no effect on how the component itself works.
Additionally, the netlist format and instance format strings have been replaced by their internal variable values, ComponentNetlistFmt and ComponentAnnotFmt. These values are only replaced if the formatting string does match. After an IFF import, the netlist format string and the annotation format string will always match the default variable values, unless special properties have been added to the IFF file (e.g. through the Library Translator). This change has no effect on how the component works.
Most of the other fields for the create_item call remain unchanged. Here are the fields that are changed, and what has changed about them:

- The third field represents the instance name prefix. By default, it will be "X". In Mentor Graphics, the instance prefixes are gotten by using the Ref property. Typically, the Ref property will be a single character, followed by a question mark (e.g. "C?"). The reference designator is used by the layout, and is a unique ID for an instance at any level of the hierarchy. Technically, this is not an identical match to the ADS instance name, because the instance name in ADS is the same for instances that are in sub hierarchies (ADS keeps track of their difference by having a hierarchy path, which will be unique, such as X1.X2.C1). The post processor will check the symbol properties of the component, and automatically look for the Ref property. It will replace the instance name prefix with the value of the Ref property without the question mark. If the Ref property is not found, the value remains unchanged.
- Field 13 of the create_item call specifies the physical layout type that is used to represent the part. By default, it will be 3, which corresponds to having no physical layout representation. This is where the parts_mapping_file.txt file comes in (see Sample parts_mapping_file.txt). For the part shown in Item Definition File after post processing (CK05BX100K_KEMET), the parts_mapping_file.txt file shows that it has a single geometry available for it, "CK05". That means that the initial value of field 13 (3, meaning no artwork) can't be accurate. The post processor will automatically change the value to **macro_artwork** . This means that the part will now use an AEL function that will parametrically define the artwork used.
- Field 14 of the create_item call is a string field. In this field, the name of a fixed artwork design is specified if the value of field 13 is **fixed_artwork** . Or, if the value of Field 13 is **macro_artwork** , the name of the AEL artwork macro function is specified. As was already noted, if a physical geometry is available for a part, field 13 will always be changed to **macro_artwork** automatically. Thus, field 14 must be changed to be the name of the AEL artwork macro that is to be used.
  The automatically generated artwork macro shows the artwork macro that is to be used for the example part. Note that this function was created automatically by the post processor. The name is constructed by taking the ADS design name (capacitor_ANSI_0_CK05BX100K_KEMET), and prefixing it with MACRO_. The assumption is that this will give the artwork function a unique name. You might be thinking at this point that it would have made more sense to make the part a fixed

artwork type, and specify "CK05". For *this* part, that would work. However, the post processor is dealing with a very large number of parts, that can allow multiple physical design footprints to be available for a single part. There were two things at work here - first, to try to have the resulting AEL files that are imported to look as much alike as possible. And, secondly, to have a single methodology that would work for all Mentor Graphics LMS components. If **fixed_artwork** is used, only one footprint can ever be available for the device, so it is not general enough to work for all cases. By using a macro, multiple fixed artworks can be utilized.

The artwork function that is automatically created will pass in all parameters for the part to the function. One of those parameters must be GEOM. The value of GEOM is assumed to be the name of the fixed artwork design to be used. A single function call, LMSArtworkFromGeom(GEOM), will open the design, and convert the database calls into the necessary AEL calls that will allow the fixed geometry to be used within an AEL macro. As the value of GEOM is changed on an instance by instance basis, the artwork of the instance is also changed.

Figure: Item Definition File after post processing

```
/* Start LMS Customization */

defun MACRO_capacitor_ANSI_0_CK05BX100K_KEMET (PART_NO, GEOM)
{
    LMSArtworkFromGeom(GEOM);
}

/* End LMS Customization */

/* Start LMS Customization */

/* Setup the GEOM form for the capacitor_ANSI_0_CK05BX100K_KEMET component */

create_constant_form("CK05", "CK05", 0, "CK05", "CK05");
create_form_set("capacitor_ANSI_0_CK05BX100K_KEMET_GEOM_FORM", "CK05");

/* End LMS Customization */

set_simulator_type(-1);

create_item("capacitor_ANSI_0_CK05BX100K_KEMET",
            "capacitor_ANSI_0_CK05BX100K_KEMET",
            "C",
            16,
            -1,
            NULL,
            "Component Parameters",
            NULL,
            ComponentNetlistFmt,
            "capacitor_ANSI_0_CK05BX100K_KEMET",
            ComponentAnnotFmt,
            "capacitor_ANSI_0_CK05BX100K_KEMET",
            macro_artwork,
            "MACRO_capacitor_ANSI_0_CK05BX100K_KEMET",
            0
,create_parm("PART_NO","",577,"StringAndReferenceFormSet",-2,prm("StringAndReference","'CK05BX100K_KEMET'"))
,create_parm("GEOM", "Geometry Footprint", PARM_NOT_NETLISTED | PARM_NO_DISPLAY,
"capacitor_ANSI_0_CK05BX100K_KEMET_GEOM_FORM", -1, prm("CK05"))
);

set_design_type(1);
library_group("BPL_RLS_LIB", "BPL_RLS_LIB", 1, "capacitor_ANSI_0_CK05BX100K_KEMET");
library_group("*", "*", 1, "capacitor_ANSI_0_CK05BX100K_KEMET");
```

So why is one artwork macro required for each part? This has a disappointingly simple answer. The artwork macro routines do not allow you to specify which parameters you wish to pass to the artwork macro. That means that, whether you need or want them, *all* of the part parameters are passed to the artwork macro. As an additional problem, if you do not explicitly set your parameters up via the Library Translator, there is no guarantee of what the order of the parameters will be once the AEL item definition file is created, or which parameters will even exist. Because you may want a particular parameter editing order, it is not acceptable to move GEOM to the front of the parameter list. So, the only

alternative is to create a custom function for each part, that has its parameter order matched up to the parameter order that is specified in the create_item function call. If you choose to add new parameters to your item definition, you need to make sure that the artwork macro is updated so it will match your new item definition.

In addition to changing fields 3, 13, and 14 of the create_item call, the post processor will also modify two of the parameters that will typically be needed for an LMS part, PART_NO and GEOM. Additionally, if either parameter does not exist, it will be created automatically based on the design name information and the parts_mapping_file.txt entry.

- PART_NO: Three things will be changed about the PART_NO parameter. First, the formset will be changed from "StdFileFormSet" to "StringAndReferenceFormSet". This is necessary because the artwork macro must have the field passed to it as a string value, meaning it must have double quotes for the value. If it does not, the artwork macro interpreter tries to interpret the part number parameter as a variable name, which causes the layout tool to get messed up. The next field of the create_parm function call is the unit type.

For StringAndReferenceFormSet, ideally this should be set to a value of -2 (which is the enumerator for a string unit). This again tells the system that values for the parameter should always be double quoted when netlisting or calling artwork macros. Finally, the default value (specified by the prm function call), has its form changed from "StdForm" to "StringAndReference", which is the preferred form to use with StringAndReferenceFormSet. The default value will also have the double quotes added to it's value.

Figure: The automatically generated artwork macro

```
defun MACRO_capacitor_ANSI_0_CK05BX100K_KEMET (PART_NO, GEOM)
{
    LMSArtworkFromGeom(GEOM);
}
```

- GEOM: This field is changed more radically than the PART_NO field. By default, the GEOM field will be imported using the StdFileFormSet, and StdForm for its form. As it happens, this would mean that the designer could type in any value for GEOM. In LMS, there are a fixed set of choices available for a part, and those choices are sent to ADS in the parts_mapping_file.txt file (Sample parts_mapping_file.txt). What is needed is to change the GEOM field from a text editing field to a dropdown list field, where the designer can only choose values from a preset number of choices.

Geometry form set created for an LMS part shows the magic code that is needed to make the dropdown list in ADS. The create_constant_form call will create a single list entry. In the example part, one geometry is available ("CK05"), so a form is created that will have as its value the name of that geometry. As many forms as necessary can be created. The create_form_set call then allows you to group the list entries into a list that can be used in

a parameter. The geometry form set for any given part is always the name of the design, suffixed with _GEOM_FORM, which will generate unique form set names for all LMS parts.

Figure: Geometry form set created for an LMS part

```
create_constant_form("CK05", "CK05", 0, "CK05", "CK05");

create_form_set("capacitor_ANSI_0_CK05BX100K_KEMET_GEOM_FORM",
"CK05");
```

With the list form set available, the GEOM parameter can now be modified to use that list. Field 3 of the create_parm call specifies the name of the form set to use. By default, it was "StdFileFormSet". The post processor will change that value to the name of the geometry form set that was created, in the example this means it is changed to "capacitor_ANSI_0_CK05BX100K_KEMET_GEOM_FORM". In addition to this, the default value must also be changed. The prm function can have multiple arguments to it. When "StdForm" is specified, it means that a text field will be available, and the second argument will be the value to set the text edit field to by default. For the geometry form, the value will be a constant. This means that only a single argument is necessary for the prm command: the name of the form to use. The default value will always be set by the post processor to be the first member of the form set list, which, in the example, is "CK05".

What About the Comments?

All of the special code that is added to the item definition file is bracketed by the comments "/* Start LMS customization */" and "/* End LMS Customization */". This is partially to allow you to know what has been added in. It is also to allow the post processor to know what has been added in by the post processor. As it happens, you can run the post processor at *any* time, not just after you've done an import. If you modify your item definitions to add new parameters, and want to make sure the artwork macro is up to date, you can simply re-run the post processor, and the LMS add on code will be updated. Keep in mind that the create_item fields will be modified if you do change any of the fields or parameters that the post processor modifies on its own. If you want to add in your own AEL formsets or other code, the post processor will not touch it, unless you stick it inside of the LMS customization comments.

Symbol File Modifications

The item definition file modifications are half of the post processing process. Additionally, the symbols need to be mapped so they will match up to the physical layout footprints for the part. In Mentor Graphics, a mapping file is specified that tells to_layout how to match up the symbols to their appropriate layout footprint. This is all well and good until you get to ADS, which does not support the concept of a mapping file. ADS does all of its layout to

symbol matching by looking at the pin numbers of the layout and the pin numbers of the symbol. Pin 1 on the symbol will always be matched to Pin 1 of the layout footprint, Pin 2 to Pin 2, and so on.

This one to one matching is simple, and if you look at the capacitor example you would not see any problems with it. All that needs to be done is to make sure that what is thought to be Pin 1 on the ADS symbol really is Pin 1 on the footprint. This is part of what the post processor is doing.

In the IFF layout file, each footprint will have pins. These pins get a particular number, based on information in the layout. The symbols in Mentor Graphics, though, could care less what their pin numbers are. The match-up is all done in to_layout by consulting the layout mapping file. Capacitor mapping file shows the mapping file that is used by the sample capacitor that has been used for this example. Notice that, in the mapping file, "POS" corresponds to the *name* of the symbol pin, and "1" corresponds to the layout pin number for its footprint. This information is passed on to ADS in the layout IFF file by consulting the mapping file, and adding a property to the layout pin record that specifies what the symbol pin name should be for the layout pin.

Once the symbol and layout footprint are imported into ADS, the post processor will read the information in the layout, and alter what the symbol pin numbers. This is *required* , because when the symbols are imported into ADS, there is no way of guaranteeing what the symbol pin number is, and, as was noted, the ADS layout generator will always map from schematic to layout based on the pin numbers.

Figure: Capacitor mapping file

```
# MAPPING FILE: "capacitor.map"
#
SYMBOL A 0
 PIN "POS" "1" 0
 PIN "NEG" "2" 0
```

The post processor will only read the first component available in the components geometry form set list. This sets one limitation on LMS components in ADS - the pin mappings for all the footprints must be identical.
While the post processor runs, you will see it open up the symbol for the part it is working on. What is happening is that each symbol pin for the symbol is being selected. Its name is looked up. The footprint is consulted, to find which layout pin is supposed to map to that particular pin name. The pin number is then changed so it matches the layout pin. Based on the mapping file shown in Capacitor mapping file , the symbol pin named "POS" will have its pin number changed to 1, and the symbol pin named "NEG" will have its pin number changed to 2.

Many LMS parts have geometries that contain power pins or ground pins. Typically, these are not represented on the symbol itself. This is not a problem for the post processor or ADS. The symbol pins do not need to start sequentially from 1, and do not need to remain in numeric sequence without gaps. It is okay to have a symbol that has symbol pin numbers 2, 4, 5, where pins 1, 3, and 6 could be two ground pins and one power supply pin. It should be noted that this could represent a problem for the netlister. Consult Adding Simulation Setups to an IFF File , or Setting Up Components for Simulation in ADS , for more information about creating custom netlisting that ignores the symbol pin order.

Limitations of Setting up LMS Parts for Layout

Additionally, the post processing step makes several assumptions about the mapping between symbols and the layout. First, if multiple geometries are available for a part, ADS will force all of the geometries to use the same mapping (i.e. if "POS" is pin 1 for the first geometry, it must be pin 1 for all geometries used). Additionally, only one mapping can be available for any given geometry. As was noted, the geometry pin will receive a property that tells it which symbol pin it will map to. In Mentor, each part can have it's own mapping file, which would allow multiple parts to map to the same geometry, and those parts could have different symbol pin names. In ADS, if multiple parts use the same geometry, they all need to use the symbol pin name (i.e. if CK05 is used for a capacitor and a resistor, to have the mappings set up properly, both need to have "POS" and "NEG" used for the symbol pin names). If this is not true, the post processor can only set up one of the symbols properly - the other symbol would need to be set up manually.