It's the components and how they are connected that creates your novel, electronic product. The components are the basic building block - so what exactly is a component, and where are they stored?

In the real world, components are tangible objects and are therefore relatively easy to identify. But in the virtual world of design, components have to be abstracted into a range of models that can be used across the various design domains. During the design capture and implementation processes, a component is represented as a symbol on the schematic, as a footprint on the PCB, as a SPICE definition for simulation, as a signal integrity description for analysis, and as a three-dimensional model for 3D component and PCB visualization - one component, with a different representation for each design domain.

Each design domain requires a specialized model. For schematic capture the model is referred to as a symbol. Because the design environment is built around the idea that design starts with the schematic, the schematic symbol is used to tie the various models together. Parameters then add the additional detail needed to identify the real-world physical component, such as the wattage, tolerance, purchasing details, datasheet references, and so on.
The symbol model links to the other models, to fully describe that component in each domain. The 3D model is actually placed within the footprint, more on this later.

**Models and Model Libraries**

Each type of model and component needs to be stored in a model file or library. Some of these use a standardized file format, like SPICE, while others use a proprietary format, like the SchLib file used to store schematic symbols or components.

Some model kinds, like SPICE, are stored one-model-per-file, these are referred to as model files. Others, like Altium PCB footprints, are stored in a file that accepts one or many footprints, these are referred to as model libraries.

**Summary of the Model and Library Kinds**

Supporting such a broad range of the design tasks performed through the electronic development process requires numerous model kinds and and library formats.

Below is a summary of each model and library kind.

<table>
<thead>
<tr>
<th>Design Domain</th>
<th>Model or Library Kind</th>
<th>Model File / Library</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schematic capture</td>
<td>Symbol</td>
<td>*.SchLib</td>
<td>Altium schematic symbol - can be just a symbol, or other models linked and parameters added to define a complete component.</td>
</tr>
<tr>
<td>Design Domain</td>
<td>Model or Library Kind</td>
<td>Model File / Library</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------</td>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PCB design</td>
<td>Footprint</td>
<td>*.PcbLib</td>
<td>Footprint or pattern that defines the components mounting and connection points on the PCB - can store both the fabricated elements (pads, silk, mask, etc), as well as an optional 3D representation of the fitted component.</td>
</tr>
<tr>
<td>3D PCB design</td>
<td>3D Body</td>
<td>*.PcbLib</td>
<td>Altium design object - place one or more 3D bodies on the footprint to build a 3D model of the component in the PcbLib.</td>
</tr>
<tr>
<td></td>
<td>STEP</td>
<td>*.STEP, *.STP</td>
<td>Standard STEP format 3D model - can be imported into a PCB 3D Body object and placed on the footprint in the PcbLib.</td>
</tr>
<tr>
<td></td>
<td>Parasolid 3D©</td>
<td>*.X_T, *.X_B</td>
<td>Parasolid format 3D model (ASCII or binary) - can be imported into a PCB 3D Body object and placed on the footprint in the PcbLib.</td>
</tr>
<tr>
<td></td>
<td>SOLIDWORKS 3D©</td>
<td>*.SldPrt</td>
<td>SOLIDWORKS format 3D model - can be imported into a PCB 3D Body object and placed on the footprint in the PcbLib.</td>
</tr>
<tr>
<td>Circuit simulation</td>
<td>SPICE3f5 / XSpice / PSpice model</td>
<td>*.mdl</td>
<td>SPICE 3f5 model file - describes a component's behavior, using the SPICE syntax.</td>
</tr>
<tr>
<td></td>
<td>SimCode model</td>
<td>*.ckt</td>
<td>SPICE sub-circuit file - a SPICE-format netlist detailing a circuit that models the behavior of the component.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*.txt + *.mdl</td>
<td>ASCII Digital SimCode model file/library and the intermediate model linkage (MDL) file required to map the digital model to the symbol - used for modeling digital components in a mixed-signal circuit simulation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*.scb + *.mdl</td>
<td>Compiled Digital SimCode model file/library and the intermediate model linkage (MDL) file required to map the digital model to the symbol - used for modeling digital components in a mixed-signal circuit simulation.</td>
</tr>
<tr>
<td>Signal Integrity analysis</td>
<td>IBIS model</td>
<td>*.ibis</td>
<td>Input/output Buffer Information Specification file - a behavioral model that describes the electrical characteristics of an input or output pin.</td>
</tr>
<tr>
<td>Component creation</td>
<td>Altium Integrated Library</td>
<td>*.IntLib</td>
<td>Compiled library - stores all model kinds within it. An IntLib is the compiled output from a LibPkg project.</td>
</tr>
<tr>
<td>Component creation</td>
<td>Altium database linked library</td>
<td>*.DbLink</td>
<td>Database link interface file - uses parametric data stored in SchLib components, to query an external ODBC data-source for a suitable record (eg, a part number, or; the resistance, wattage and package). The data from fields mapped in the DbLink is retrieved and added to the component during placement.</td>
</tr>
<tr>
<td>Component creation</td>
<td>Altium database library</td>
<td>*.DbLib</td>
<td>Database library file - presents an external ODBC data-source as an Altium component library, allowing components to be placed directly from the database. The data from fields mapped in the DbLib is retrieved and added to the component during placement. Requires the Altium models (symbol, footprint, etc) to be specified in each database record.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*.SVNDbLib</td>
<td>DbLib with version control - interfaces to an SVN repository for version control of the schematic and PCB models.</td>
</tr>
<tr>
<td>Component creation</td>
<td>Vault component</td>
<td>*.CmpLib</td>
<td>Component definition file for an Altium Vault component - links Vault-based models and parametric data together, when a release is performed the component item is created in the Vault.</td>
</tr>
</tbody>
</table>

**The Component**

A component can range from something simple, like a two pin resistor, right through to a large FPGA with hundreds and hundreds of pins. To give flexibility over how the components can be organized on the schematic sheets, the software includes a number of useful display-related features.
Is it a Symbol or a Component?

Once you have linked in other domain models and add parameters, the symbol becomes a component. Many designers continue to refer to it as a symbol when they see it on the schematic - because that's what you see - but attached to that symbol are the other domain models and parametric details that define it as a component. This article will use the term **symbol** to describe the graphical objects and pins that are placed in the schematic library editor, in all other situations it will be referred to as the **component**.

The symbol is simply the graphical objects and the pins (left image), once the models are linked and the parameters added, it becomes a component (right image).

The symbol is created in the Schematic library editor. As soon as you add link in models and add parameters, it becomes a component. The component detail can be added in the schematic library editor, or if you use a DbLink or DbLib type library, the detail can be extracted from a database and added to the symbol during placement on the schematic (more on these later). Alternatively, the detail can be added after the symbol has been placed on the schematic sheet.

**The Symbol**

The symbol is created in the schematic library editor. It is created by placing graphical objects to build the required shape, and then the pins are added. It is the pins that bring electrical intelligence to the component, they are what the software identifies as the electrical connection points, and what are mapped to the pads on the PCB footprint.
Symbols can range from the small and simple, like this MOSFET, through to high pin-count components like the FPGA, created over multiple parts.

Notes about symbol creation:

- Objects placed in the library editor are stacked in the order they are placed, use the Edit » Move commands to change their display order.
- Only one end of a pin is electrical, this end is referred to as the hotspot - make sure the hotspot (the end you hold the pin by) is away from the component body. Refer to the Pin object for more information.
- In the schematic library editor, the component's properties dialog is opened by double-clicking the name in the SCH Library panel, or clicking the Edit button below the list of component names.
- In the images on this page the designator and comment are displayed on the library editor sheet, this option is enabled / disabled for the current library in the Schematic Library Options dialog (Tools » Document Options).
- The fonts can be configured individually for each object as they are placed, or you can set your preferred fonts for new objects by editing each of the appropriate objects in the Schematic - Default Primitives page of the Preferences dialog.
- Multiple pins can be imported from a spreadsheet into the current component, via Smart Grid commands in the right-click menu in the SCHLIB List panel.
- Install the Schematic Symbol Generation Wizard extension to quickly build high-pin count components, the Wizard also supports importing pin details from a spreadsheet, via the right-click menu.

Refer to the article, Creating the Schematic Symbol, for details of the symbol creation process.

Refer to the article, Creating the PCB Footprint, for more information on how to create a footprint.

Before you start creating component symbols, it is worth spending some time configuring your preferred defaults. The defaults for all schematic design objects (both library and sheet) are configured in the Schematic - Default Primitives page of the Preferences dialog.
During the installation of a new version of the software you will be prompted to load your default preferences from the previous version. You can also save them to a file or the cloud (via the dropdown at the top of the Preferences dialog), handy if you need to transfer them to another computer.

Aliases - Multiple Names for the Same Component

Sometimes components are logically equivalent, but have different performance specifications. An example would be a logic gate that is available in a variety of logic families, for example a 74ACT32 and a 74HC32. In this case the symbol is drawn once, and then another name, or alias, is defined for each equivalent component required.

Component aliases are added via the SCH Library panel during component creation. Component aliases can be thought of as one component, with multiple names. Each alias presents as a unique component when that library is added in the Libraries panel. Note that only one set of parameters can be defined for all aliases. Because of this restriction the alias feature is less popular now, as many designers require each library component to map to a real-world component, which can not be done for each alias.

Display Modes - Multiple Presentations of the Same Component

Another design requirement that is sometimes needed is to be able have different display representations for the same component. For example, some of your clients might prefer to have their resistors drawn as a rectangle, while others prefer a wavy line.

Each of these representations is referred to as a Mode. You can define additional symbolic representations for a component by adding a new Mode, either from the schematic library editor Tools menu, or via the Mode toolbar.
A resistor created with two display modes. The library editor includes a Mode toolbar, which can be used to add/remove and step through the modes.

The required mode is chosen when the component is placed from the library onto the schematic sheet, using the **Mode** selector in the component's properties dialog. The default placement mode is the mode that was displayed in the library editor when the library was last saved.

Each mode must include the same set of pins, if they do not a warning will be generated when the project is compiled. This is required as you can only define one set of **pin-to-pad mappings** for each footprint. Pins can be hidden in a mode if required, and do not need to be in the same location in each mode.

**Breaking the Component into Multiple Parts**

In some instances it is more appropriate to divide the component into a number of symbols, each of which is referred to as a **Part**. Examples include: each resistor in a resistor network; the coil and contact sets in a relay; each I/O bank in a large FPGA; or each pin in a connector, for example if you prefer to place connector pins throughout the sheet, rather than routing the wiring to a single connector symbol.

These components are referred to as **multi-part components**. Each part is drawn individually in the schematic library editor, and pins are added accordingly. The image below shows the same resistor network drawn as a single part, then as 4 separate parts.

**Notes on working with multi-part components:**

- In the library editor, use the **Tools » New Part** command to add another part to the current
component.

- Use the *SCH Library* panel to move between parts in a multi-part component, as shown below.

![SCH Library panel](image)

- The designator of a multi-part component includes a suffix to identify each part. The suffix can be alpha or numeric, set your preferred mode for the **Alpha Numeric Suffix** in the *Schematic - General* page of the *Preferences* dialog. Note that this option is a software installation environment setting, it does not get stored in the library or the schematic file, so does not travel with the design files.

- As well as the displayed parts, there is also a part zero. This is a non-graphical part that is used for pins that need to be present on all parts, such as power pins. A pin is assigned to part zero in the *Pin Properties dialog*. If required, these pins can be pre-connected to a specific net, and hidden from display. When the design is compiled they are automatically connected to the specified net.

- Multi-part components are considered homogeneous, that is, all parts are equivalent during design annotation, and can potentially be swapped during the annotation process. To lock a specific part in a placed component, for example the coil of a relay, enable the **Locked** checkbox in the component's properties dialog (shown below).

![Properties dialog](image)

- During schematic annotation, the parts in a multi-part component are packed together to complete a component, in accordance with the *Matching Options* configured in the *Annotate dialog* (as shown below). The enabled checkboxes on the left determine which component properties must match for parts to be eligible to be packed together. If you need to control the packing of specific parts so they are together in the same physical component, for example a pair of op-amps in a filter design, and do not want to assign and lock them manually, add an addition parameter to that component and enter a value that defines which parts are to be packed together. The image below shows the *Annotation* dialog matching options. The enabled parameters are used for multipart component matching, the ResPack parameter has been added to control the packaging of resistors into packages - when the parts have the same parameter value they can be packaged together into the same physical component. Note the Strictly option, if this is enabled then the part must include this parameter to be packed. Be careful of this option if you have different types of multipart components that you are controlling the packaging of - all of them must include that parameter if Strictly is enabled.
PCB part swapping can only be performed on a component whose parts are defined as a multi-part component. Learn more in the Pin and Part Swapping article.

**The Component Type**

In a design environment, you may also need to create design entities that are not necessarily components that will be mounted on the finished PCB. For example, there might be an external module that connects to the board, that you would like to draw as a component and include on the schematic for design clarity, but you do not want this to be included in the BOM for this board. Or there might be mechanical hardware, such as a heat sink with mounting screw, that is needed in the BOM, but you do not want to include on the schematic.

These situations are managed by setting the component's Type. For the example just described, the component type could be set to **Graphical**. Another special class of component would be a test point - this component is required on both the schematic and the PCB, it should be checked during design synchronization, but is not required in the BOM. In this case the component Type is set to **Standard (No BOM)**.
For a non-standard type of component, set the Type accordingly.

As well as being used to determine if a component should be included in the BOM, the **Type** field is also used to determine how that component is managed during component synchronization. All of the **Standard**, **Net Tie** and **Jumper** Types are fully synchronized - that is the component is passed from the schematic to the PCB, and the net connectivity is checked. For the **Mechanical** and **Graphical** Types, the component is not passed from schematic to PCB. But if a component with one of these types has been placed manually in the PCB and the matching Type option has been chosen, then component-level synchronization is performed, but no net-level connectivity checks are performed.

Refer to the *Component Properties dialog* for information on the various Type options.

**Using Parameters to Add Detail to the Component**

Much of the detail that describes a component is defined as parameters, they allow the designer to define additional textual information about the component. This can include electrical specifications (i.e., wattage or tolerance), purchasing or stock details, designer notes, references to component datasheets, basically any purpose the designer chooses.

Parameters can be defined in the schematic library editor during component creation, they can be added automatically during placement if the component is placed from a database-type library (more on this later), or they can be added manually once the component has been placed on the schematic.

**Adding Parameters to a Component**

For an individual component, parameters are added in the component's properties dialog. As mentioned, this can be done in the library, or once the component has been placed on the schematic sheet.
Parameters add detail to the component; this one includes 2 component links.

User-defined parameters can be displayed on the schematic alongside the component if required, by enabling the appropriate Visible checkbox. Edit the parameter to enable the display of the parameter name as well.

Component parameters can also be included in a generated PDF, by enabling the **Include Component Parameters** option in the PDF generation settings. PDFs can be generated directly from the schematic via the Smart PDF wizard, or from an Output Job file. Click on a component in the PDF to display the parameters, as shown below.
Using Parameters to Link to Reference Information

In the previous two images, there are two pairs of ComponentLink parameters. These are special purpose parameters that are used to define links to external data. These links can be accessed via the right-click menu on the schematic component, or in the Libraries panel, as shown below.

These links can also be included in a PDF generated from the schematic, either via the Smart PDF feature, or via a PDF generated from an Output Job file. The image above shows how the list of component parameters presents in the PDF, any that are URLs can be clicked on to browse from the PDF to that location.

Right-click on a component on the sheet (left image) or in the library (right image) to access the component links.
As well as the ComponentLinks parameter pairs, there is also the HelpURL parameter, when this is present you can press F1 over a component on the schematic to open the specified link (URL, PDF, TXT or DOC).

Refer to the Parameter object to learn more about linking to reference information.

Mapping a Parameter into the Component's Comment Field

When the design is transferred from the schematic editor to the PCB editor, the only textual information that is transferred for a component is the Designator and the Comment. To support passing other component parameters to the PCB, you can map any of the component's parameters into the component's Comment field, using a technique known as string indirection. When a string is using the indirection feature, it is referred to as a special string.

This is done by entering the parameter name in the form =ParameterName. For example, to map the value of the component parameter called Value into the component's Comment field, enter the string =Value, as shown below.

To display the actual data on the schematic sheet instead of the special string, enable the Convert Special Strings option in the Schematic - Graphical Editing page of the Preferences dialog.

Use the special strings feature to map any parameter value to the component's Comment.

Special strings allows the mapping of any parameter to any string. The string can be a component string, a free string placed on the schematic sheet, or a string placed in a schematic template. The parameter can be a component parameter, a document parameter or a project parameter. Refer to the string object for more information on special strings.
Adding Parameters to Multiple Components

Parameters are a key element of each component, and it is common for many of the parameters to be used across multiple components. As well as adding them individually to each component, you can also use the **Parameter Manager** command to add them to multiple components. As well as adding / removing parameters, the values of parameters can also be edited across multiple components.

The Parameter Table Editor can be used to edit all of the parameters across all of the components.

**Notes about editing multiple parameters:**

- Parameters can be edited across the components in a library, or across components used in a schematic design, using the same process. Select **Tools » Parameter Manager** in either the schematic or schematic library editors to start the process.
- Because parameters can be added to a variety of different objects, the first step is to select which object kinds the parameters are to be edited in, in the **Parameter Editor Options** dialog.
- Parameter editing is performed in the **Parameter Table Editor dialog**. The dialog can be accessed in different ways, so the dialog caption can change. The way you work in the dialog is the same though: use the standard Windows selection keystrokes to select the cells of interest, then right click and choose an edit action.
- Changes are not performed immediately, they are done via Engineering Change Orders (ECOs).
- Each cell that is being changed is marked by a small colored icon, refer to the **Parameter Table Editor dialog** page for a description of each icon.

To learn more about working with parameters, refer to the article, **Adding Design Detail with Parameters**.

**Linking from the Component to External Supplier Data**

*Main article: Linking to Supplier Data*

From the electronic product designer's perspective, one of the most important aspects of component creation is linking from that component, out to the real-world component that it represents. This
linkage can range from simply entering the component's Part Number in as a component parameter, through to linking to the purchased component in your company database via a DbLink or DbLib (more below). Another approach is to link directly from the design component to the component's supplier, via the Linking to Supplier Data feature.

The key to linking to supplier data is a pair of component parameters, namely Supplier n and Supplier Part Number n. Using these, the software connects to that supplier's web-services, where it can access detailed information about the part: including manufacturer; manufacturer part number; price; voltage rating; and so on. The software has direct access to web-hosted information for a number of major component suppliers, configured in the Data Management - Suppliers page of the Preferences dialog.

Your interface into the supplier linking feature is the Supplier Search panel, click the button down the bottom right to display the panel.

The Supplier Search panel gives you direct access to up-to-date component data.

To work with the panel:

- Click the button to open the Data Management - Supplier page of the Preferences
dialog, where you enable the required suppliers. Some of the suppliers include account log in
details, these are used to access account-specific options, such as customer-specific price
breaks. Note that the preferences also include keywords settings, and Parameter Import
Options, which are used to map from the supplier's naming, to your preferred parameter
names. Parameters can be excluded if required, and can also have a numeric suffix added
(handly if you plan on adding multiple supplier links to the same component).

- In the Supplier Search panel, click the ... button to configure the Filter options.
- Enter your search string into the Keywords field at the top of the panel and press Enter on the
  keyboard, or click Search.
- Search results are listed in the panel, click on a result to display detailed parametric information
  about that component in the lower section of the panel.
- Drag from the panel onto the schematic, or right click on a part in the results list and select
  either:
  - Add Supplier Link to Part - add just the Supplier n and Supplier Part Number n
    parameters to a component.
  - Add Supplier Link and Parameters to Part - add all available parameters, including
    the Supplier n and Supplier Part Number n parameters, to a component.
- If you do not want all parameters, select the required parameters using the standard Windows
  selection techniques, then right click on one of the selected parameters and run the Add
  Parameters to Part command.
Parameters can be selectively added, if required.

Supplier links can be added to an existing component, or a new component can be created, by configuring the workspace as described below, and then right-clicking on the component in the Supplier Search panel and selecting the appropriate command.

To work between the panel and a:

- **Schematic library** - add parameters to the current component, or add a new component to this schematic library. Alternatively, drag and drop from the panel onto the symbol, or the components name in the SCH Library panel to add parameters to that component.
- **DbLib / SVNDbLib** - enable the display of the Table Browser, then add parameters to the selected component record, or add a new component record to the database.
- **Component on a schematic sheet** - choose one of the Supplier Search panel's right-click commands, when you move the cursor over the schematic sheet it will change to a crosshair, click on the required component on the sheet. Alternatively, drag and drop from the component list in the panel, onto the component on the schematic sheet.
Adding Models to the Component

At the schematic stage, the design is a collection of components that have been connected logically. To test or implement the design, for example circuit simulation, PCB layout, signal integrity analysis, and so on, it needs to be transferred to another domain. To achieve this, there must be a suitable model of each component for the target domain.

Models are linked to the schematic component by adding them on the Models region of the component's properties dialog.

Models are added in the component's properties dialog, each model type opens a different model editor. As part of the process of linking the model, information about the component must be mapped from the schematic to the target model.

In the Schematic library editor, models can be added in via the component's properties dialog, via the SCH Library panel, and the model region displayed across the bottom of the editing workspace.

The 3D model is not linked directly to the component symbol. Instead, the 3D model is placed in the PCB footprint. Why? because the 3D model must be rotated, aligned and positioned correctly, relative to the footprint, so it makes sense to do this in the footprint editor. To learn more about working with 3D models and placing them in a footprint, refer to the article Creating the PCB Footprint.
Mapping the Model to the Symbol

The domain-specific information resides in model files that have a predefined format, such as IBIS, MDL and CKT files. There is other information needed for the system to access that domain-specific information, such as the pin-mapping and net listing between the schematic symbol and the domain specific model. This information is defined in a domain-specific model editor which opens when a model is added or edited - you can see examples in the image above. As well as referencing the model file, it will also include any pin mapping or netlisting information required for that model kind.

For information about mapping a model to a symbol, press **F1** when the required model dialog is open.

These are a number of predefined analog device models that are built-in to SPICE. For these component types there is no separate model file required, all of the information needed to model them is configured in the *SIM Model* editor.

Where the Models are Found

At the top of each model dialog there is a **Name** field, this must contain the name of the model file. You can type the name in, or **Browse** to find it. At the bottom of the dialog there will be a string that shows where that model has been found.
When the named model is found, the model appears and information is displayed about where it has been found.

The software’s ability to find the model is influenced by the setting that define where it can look for models. This is the setting just below the model name - the label of this area of the dialog will change depending on the model-kind - for the footprint shown in the previous image it is the **PCB Library** setting. The options range from Any, meaning search all available libraries for this model, through to Integrated Library or Vault, which require that the model can only be used from the specified Integrated Library or Vault.

Although they vary slightly from one model type to another, the model editor dialogs generally include these options:

<table>
<thead>
<tr>
<th>Library Option</th>
<th>Behavior</th>
<th>Dialog Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>Searches all available libraries for a matching model.</td>
<td><img src="image" alt="PCB Library with Any option" /></td>
</tr>
<tr>
<td><strong>Library name</strong></td>
<td>Only searches available libraries of this name for a</td>
<td><img src="image" alt="PCB Library with Library name option" /></td>
</tr>
<tr>
<td></td>
<td>matching model.</td>
<td></td>
</tr>
<tr>
<td><strong>Library path</strong></td>
<td>Only searches a library of this name, in this location, for</td>
<td><img src="image" alt="PCB Library with Library path option" /></td>
</tr>
<tr>
<td></td>
<td>a matching model.</td>
<td></td>
</tr>
<tr>
<td><strong>Integrated library</strong></td>
<td>Only searches for the model in the integrated library that</td>
<td><img src="image" alt="PCB Library with Integrated library option" /></td>
</tr>
<tr>
<td></td>
<td>the component was placed from. The integrated library must</td>
<td></td>
</tr>
<tr>
<td></td>
<td>be available.</td>
<td></td>
</tr>
<tr>
<td><strong>Vault</strong></td>
<td>Only searches for the model in the Altium Vault that this</td>
<td><img src="image" alt="PCB Library with Vault option" /></td>
</tr>
<tr>
<td></td>
<td>component was placed from. The software must be connected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>to that Vault.</td>
<td></td>
</tr>
</tbody>
</table>
Available Libraries

Vault-based components are placed directly from the Vault, so the concept of searching for components and locating their relevant models is simply a case of connecting to a Vault and searching or browsing its contents from within the Vaults panel. For a Vault component, the model location setting is locked to *Use <ModelKind>* from component library *<VaultName>*.

The most common approach for traditional library-based component models is that they are sourced from any of the libraries that are currently available. This is the function of the *Any* setting detailed above, search for the named model in all Available Libraries.

The *Libraries* panel shows the available libraries in its dropdown library selector list, as shown in the adjacent image.

Note that the dropdown has a filter, accessed by the button, use this to display/hide footprint and legacy PCB3D libraries.

Managing Library Search Locations & Available Libraries

The list of available models and libraries is managed in the *Available Libraries* dialog. Open the dialog
by clicking the Libraries button in the Libraries panel, or by selecting the Design » Add/Remove Library command.

The dialog has 3 tabs, all of the libraries and model locations defined in these tabs are collated to make up your list of available libraries. When an action is performed that requires searching for a model, such as transferring the design from schematic to PCB layout, the libraries are search in the order of the tabs, then within each tab, in the order that the libraries/models are listed. As soon as the correct model is located, the searching process ceases.

**Available Project Libraries**

Libraries that are part of the current project are automatically detected and listed under the Project tab of the Available Libraries dialog, when that project is the active project in the software. The advantage of project libraries is that whenever the project is opened, the model/libraries will automatically become available. The disadvantage is, if the models/libraries are not stored in the same project folder structure as the design files, they can easily be forgotten if the project files are moved.

![Available Libraries dialog](image)

The Project tab provides a list of available libraries in the currently loaded project.

Any library can be a project library, they do not need to be stored in the project's folder. Right-click on the project name in the Projects panel and select the Add Existing to Project command to include libraries as part of the project.

**Installed Libraries**

Libraries and models that have been made available in your installation of the software, are referred to as installed libraries. These are listed in the Installed tab of the Available Libraries dialog. These models/libraries are available to any design currently open in your installation of the software. Note that this list of libraries is saved with the environment, not with any of the open projects.

Installed Libraries can be listed using an absolute path, or a path relative to the Library Path Relative To: setting. The advantage of using a relative path is that this lets you create a common sub-environment across multiple PCs and easily move the design files between them. Additionally,
installed libraries can be temporarily deactivated by clearing the **Activated** checkbox, rather than needing to be removed.

**Search Path**

The Search Path tab presents a list of libraries that have been located according to the path settings defined in the **Search Path** tab of **Options for Project** dialog, which can be accessed through the **Paths** button in the **Search Path** tab in the **Available Libraries** dialog. Each search path defines a folder, and can include sub-folders if the **Recursive** option is enabled (available during path definition). All model and library files found down the search path will be available. Search paths are saved with the project.

Retrieving models using search paths can be slow if there are a large number of files in the search path folder(s), for this reason it is not recommended to use this approach for PCB libraries, as they can be large files containing many footprints. The feature was developed to provide a way of referencing available simulation and signal integrity models.

Use the Search Path feature to locate simulation and signal integrity models.
While the Altium Designer environment offers flexibility and control over specific model/library locations, it does require the correct file extension to be used for each model type. For example, a footprint will only be found if it is in a file with a .Lib or .PcbLib extension. Similarly, a SPICE .SUBCKT will only be found if it is in a .ckt file, and a SPICE .MODEL will only be found if it is in a .mdl file. Whenever a model search does not yield a match, an error will appear in the Messages panel.

**Moving a Design from One Location to Another**

When a component is placed from a library into a design, it is cached in the design file so that the document can still be opened at any location without requiring its source libraries to be present or loaded. This is helpful when moving a design from one location to another, since it is not necessary to move the libraries as well. Note that the original source library name and the model names are also stored within the placed component's properties.

You can generate a schematic library of all components placed in the current schematic project, or a PCB library of all footprints placed in the current PCB, from the respective editor's Design menu.

**The Libraries**

Now that you've got your head around the models and the components, let's explore the storage options.

Models and components are store in libraries, the Altium design software supports a number of different types of libraries. How do you decide which library-type is best for your company?

**Schematic and PCB Models and Model Libraries**

In the early days of electronic design automation there were two design domains, schematic capture and PCB layout. The outputs from schematic capture was a set of printed schematic sheets, and the output from the PCB layout was the PCB fabrication and assembly files.

To support this, the designer needed to be able to create and store the schematic symbols, ready to be placed on the schematic sheet. Details such as the component's value, voltage, wattage, footprint and so on, were often added after the symbol was placed from the symbol library onto the schematic sheet.

As for the footprints, the focus of the editing capabilities was implementing the design as a set of layers that contain the shapes needed for the phototools - this output-driven focus dictated what shapes were supported, right down to the font used for the designator and other silkscreen strings.

These simple symbols and footprints were created and stored in libraries, with both types initially having the file extension .Lib. Once Windows relaxed the 3 character limit on file extensions, these were changed to .SchLib for schematic symbols, and .PcbLib for the footprints.
Schematic and PCB libraries are storage containers for models.

**IntLib - The Integrated Library**

*Main article: Working with Integrated Libraries*

Over time, there was a growing desire to be able to separate the component creation process from the product design process. Instead of adding the component detail after the symbol had been placed on the schematic sheet, design teams wanted to fully define the component in the library. Other design domains, such as circuit simulation, were also being developing, models for these domains needed to be supported too. As part of creating ready-to-use components you also need to be able to verify the symbol-to-model mapping, and then deliver these components in a single, ready-to-use library file.

To satisfy these requirements, the integrated library (.IntLib) was developed. The integrated library carries all model kinds in a single file. The source for an integrated library is a library package (.LibPkg), a design project that brings together the source symbols, footprints and simulation models. When the library package project is compiled, the symbol-to-model mapping is verified, and the Integrated Library is created. Using an IntLib means that the library becomes something you simply place ready-to-use components from - delivering a good solution for medium to large organizations that want to separate component creation from product design.
When the LibPkg is compiled the model linkages are verified and the integrated library created - the IntLib includes a copy of each model needed for every component.

**DbLink - Linking the Components to a Company Database**

*Main article:* [Working with Database Libraries](#)

The integrated library delivered a good alternative to simple model libraries, solving the challenge of managing the component data separately from the product design process. What it did not solve was the ever-increasing desire to create a bridge between the electronic design domain and the broader company component data systems. To solve this challenge, there needs to be a way of linking from the component, into the company database.

This is not a trivial problem to solve, as there is no single approach used to define, store and manage large volumes of component data. The databases that companies use are varied, and are often custom-built to suit the needs of that organization. To provide a connection from the electronic design environment into the database, the ODBC standard was chosen.

Open Database Connectivity (ODBC) is an open standard application programming interface (API) for accessing a database. By using ODBC statements in a program, you can access files in a number of different databases, including Access, dBase, DB2, Excel, and Text.

To create the link, a new interface file was developed, known as the Database Link file (.DbLink). Strictly speaking this is not a library, instead it is a project Settings file that provides an interface between the parametric data in the design components (stored in standard model libraries), and the records in the database. Database records can be queried either by using a single key lookup, such as a part number, or a user-defined Where clause. By using a Where clause, the designer does not need to search the database, find the part number, and include it in their schematic component. Instead, they define a where clause that queries the database based on component data, such as the Value, Voltage, Current, Footprint, and so on.
The DbLink file defines the:

- Target database and connection methodology
- Lookup method (single key or where clause)
- Mapping between the required database cell and an EDA design parameter

Using this approach, the designer can have simple design libraries, particularly for discrete components. For example, their capacitor library might have a single ceramic capacitor component. The designer places the capacitor on the schematic, enters the capacitance, voltage and tolerance, and selects the required footprint from a list of pre-defined footprints. They then query the database. The Where clause defined in the DbLink file is used to create an ODBC query, the database is queried, and returns the requested data from the found record, such as the part number, relevant company BoM data, and so on, which is added to the placed component.

The simple, generic component that was placed, is now a fully described, BoM-ready company component.
The DbLink performs the database query according to the lookup settings (a Where clause in this example), when a match occurs that record’s mapped values are returned and loaded into the parameter editor.

The parameter table editor appears, where the parameter changes can be reviewed (and changed), when accepted they are applied to the components by ECOs - the generic component is now a fully detailed company component.

The strength of this approach is that it gives a view into the company's component database, without requiring EDA-type data to be entered into that database. The ability to use a Where clause also means that the designer is free to place a loosely specified (generic) component, which does not get locked down until they perform the query process. The database can be queried at any time during the design process, including during BoM generation.

**DbLib - Placing a Component from the Company Database**

*Main article: Working with Database Libraries*

The Database Link document delivered an excellent solution for connecting the company component data directly into the EDA design components. However, for companies that want to create a one-to-one link between the company component record and the design component (using a single key lookup, typically the component part number), using a DbLink requires the EDA librarian or designer to manually trawl through the database, extract the part number, and add it into each design component.

These companies want a data-centric solution, rather than the EDA-software-centric solution delivered by the DbLink file.

To deliver this, the Database Library (.DbLib) was developed. The DbLib allows the component record in the database to be the component. Rather than operating as a linkage file between the EDA library and the company database, the DbLib installs as a design library, sourcing its data from the company database, displaying it directly in the Libraries panel.

The DbLib is installed in the same way as a schematic library or an integrated library. Acting as a window between the design environment and the company database, it displays data sourced directly from the company database, inside the design environment. Using a DbLib, the designer is able to
browse or search component records, and place a component directly from the database into their design. When the **Place** button is clicked, the symbol referenced in that record is located in the SchLib, and the mapped data loaded into the component's parameters - in essence the EDA component is assembled as it is being placed on the schematic sheet.

This approach requires that references to the EDA models are stored in the database record (symbol, footprint, simulation model, and so on). The models themselves continue to reside in standard Altium libraries, such as SchLibs and PcbLibs.

The DbLib maps the database data into the Libraries panel, and is also used to map the chosen record data into the component parameters, during placement.

The DbLib acts as a window into the company database, the view that it provides is framed by what fields are mapped into the component.
SVNDbLib - The Version Controlled Database Library

Main articles: Working with Database Libraries, Version Control and Design Repositories

Another common requirement of larger companies is to be able to version control the intellectual property that the company creates. A Version Control System (VCS) is an excellent way of managing changing electronic data - as well as holding a secure copy of the last-committed version of your data, it also allows you to roll-back to any previous version.

Working with an SVNDbLib is essentially the same as working with a DbLib - you install the SVNDbLib as a library in the Libraries panel - the only difference is that the models are under version control. That means the models must be stored in a repository, and a working copy checked out. It is the working copy that is displayed in the Libraries panel and used when the Place button is clicked.

The use of the version controlled models is transparent, you simply place the required component from the Libraries panel. To ensure that the latest revision of a model is placed, whenever the Place button is clicked the repository is queried and if necessary, the working copy of the model is automatically updated. The working copy can also be refreshed at any time, via the Libraries panel right-click context menu.

The management of the version controlled models is also straight-forward, you can edit a model via the context menu in Libraries panel. Right-click on a component in the list to edit the symbol, right click on a footprint in the model list to edit that footprint. Once editing is complete the model file is saved, you then right-click on the model's name in the Projects panel and use the Version Control sub-menu to commit the changes to the repository. Your changes will immediately be visible in the Libraries panel.

A key requirement for using an SVNDbLib is that all of the model must be stored in their own file. This is necessary to allow an individual model to be updated and a new version of that updated model committed to the VCS repository. To simplify this process, the software includes a Library Splitter Wizard, which can quickly break an existing library into a set of one-model-per-library files.
Version-controlled models can be edited directly from the panel (left image), saved, then Committed back into the repository, ready for use (right image).

The SVN in the SVNDbLib file extension stands for Subversion, a popular and free version control system. Altium Designer includes Subversion and an SVN client, so it can create new repositories and also perform all of the typical VCS-type actions, such as Commit (check-in), Update (check-out), Revert, and so on. Many designers like to install an external SVN client as well, the free TortoiseSVN client is popular, as it has excellent functionality and is easy to use.

To learn more about working with version control from within the software, refer to the Version Control and Design Repositories article.

The Altium Vault

Main article: Working with Vault Components

The internet has bought a massive growth in global connectivity. With it, has come the ability for a company to connect all of its sites together, across the globe. For this to deliver benefits to an electronic product development company, they need to be able to store access and manage their design data in a globally accessible way.

Enter the Altium Vault. The Vault has been developed to deliver a comprehensive, globally-
accessible, electronic design data storage system. It can handle full design projects; components; as well as shared resources, such as templates and design reuse elements.

For each of these data types, it includes a broad range of data management features, including: revision management; lifecycle status management; supply chain data; and where-used capabilities. The supply chain data feature is comprehensive,

Underlying this, it has organization-level operational features to establish and enforce your electronic design workflow - ensuring every member of the team is working from the same approved templates, while connecting and storing all design data in the Vault, throughout the design's lifecycle. Backups and recovery of Vault data are easily maintained from a single, organized location. Plus, user-access management and desktop software installations and updates, can all be administered from a centralized network administrator interface.

How you use the Vault is up to you, for example your company might choose to simply use it for components, but not for projects or other design data.

The models used in your components start life by being created in the traditional way, on a designer's PC. When the models are ready, their library is linked to the Vault and the models released into it.

A Vault component is defined in a component library (*.CmpLib). This is a spreadsheet-like document, where you bring together the required Vault models, and add the component parameters.

An individual component can be created directly in the Vault, by creating a new component Item and editing it (right-click menu). This process opens the CmpLib in Altium Designer, once the component is complete and released into Vault, the CmpLib is can be closed without being saved. Using this approach the CmpLib is not a source file, it exists only to provide a method of editing Vault-based component data. At any time that same component can be edited again and released back into the Vault, creating revision 2 of that component.

This direct, or fileless editing of components, is an excellent approach for an individual component, allowing the Vault to manage the ongoing revisioning process of that model or component.
Vault components are defined in a CmpLib, here an entire series of resistors is being configured, ready for release into a Vault.

If you are creating a series of components, for example a resistor series, you can also create and store the CmpLib locally. The entire series can be defined in a single CmpLib, then using the spreadsheet-like editing capabilities you can quickly build the entire series. When ready, the CmpLib is linked to the Vault, and the components you choose are released into the Vault.

Vault components are accessed through the Vaults panel, as shown below. They can also be accessed from a web browser, ideal when you need to give people outside of the design team access to your component details.

The Vaults panel presents its content in a similar fashion to the Windows File Explorer - a tree of folders is displayed down the left of the panel, with the Items in the currently selected folder shown in upper section on the right of the panel. When you select an Item, its details are shown in the lower section on the right of the panel, as shown in the image below.
Vault components are accessed via the Vaults panel, use the Search feature to locate a component of interest, set the display to Preview mode to examine its detail.

Use the Vault panel's **Search** feature to quickly locate an Item of interest.

If you'd like to explore Vaults, and how they can be used for components, why not connect to the Altium Content Vault. It contains tens of thousands of ready-to-use components. Go to the [Data Management - Vaults](https://www.altium.com/documentation/display/ADES/((Understanding+Models,+Components+and+Libraries))_AD) page of the Preferences dialog to connect to the Vault.

---

**Source URL:**
https://www.altium.com/documentation/display/ADES/((Understanding+Models,+Components+and+Libraries))_AD