As emphasized at the outset of the report, it is not our purpose here to review the Analytica modeling system or its language. EXHALT is a computer program; to understand it fully, much less to change it, one must understand both modeling and programming. That said, working with an Analytica model is much easier than programming in C or some other general-purpose language. To illustrate, suppose we wish to understand how the various situational multipliers are applied. If we open the EXHALT program by double clicking on its file, we obtain Figure E.1. If we then double click on the “EXHALT Model Interface Overview User Notes Modifications EXHALT Credits”

Figure E.1—EXHALT’s Top-Level Window
Model” module, we obtain Figure E.2. When we began building EXHALT, we did so by constructing Figure E.2—without specifying details of the individual models.

If we now double click on the Blue Effectiveness module, we obtain Figure E.3. Again, drawing this figure was the way we designed the model. The figure is not after-the-fact documentation, but rather the starting point.

If we double click on the Kills per Shooter-Day module, we obtain Figure E.4. We are now at the lowest level in the model.

The window indicated in Figure E.4 shows two variables; the one on the left is the kills per shooter day (yes, this variable has the same name as the module within which it sits); the one on the right is a variant of the first expressed as a multiple of what a “standard shooter” (an F-15E) can kill in a day. Let us look at the former. If we double click on Kills Per Shooter-Day, we obtain the window shown in Figure E.5. This defines everything there is to know about the variable.

At the top of the window we see the computer’s name for the variable. Below that we see the equivalent English-language title we have given it. It is this title that appears in the various diagrams. Reading downward, we see “description.” This is an example of self-
Figure E.3—The Blue Effectiveness Model
KILLS PER SHOOTER-DAY

Blue Effectiveness (input)

Kills per Standard Shooter-Day

Standard Shooter Type

**Figure E.4—The Kills Per Shooter-Day Module**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Killoshooterday</th>
<th>Units:</th>
<th>AFVs/shooter-day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>KILLS PER SHOOTER-DAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Kills per shooter-day achievable by each aircraft type.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expression:</td>
<td>Effectiveness_mult<em>Kps</em>Sr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value:</td>
<td>Calc</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure E.5—The Definition Window for Kills Per Shooter-Day**
documentation: Whenever we introduce a new node, Analytica encourages us to fill in this description and thus provide an English-language explanation of what the node “means.” Analytica, of course, cannot force us to do a good job.

Continuing, we find the “definition.” This is the algorithm itself. It is equivalent to an equation saying:

\[
\text{Killspershooterday} = \text{Effectiveness\_mult} \times \text{Kps} \times \text{Sr}. \tag{E.1}
\]

This, then, is the level of the model’s “meat,” where the algorithms are expressed. To be sure, we chose one of the easiest-to-understand nodes in the entire model, but it is sufficient to explain the visual-modeling environment.

Below the definition field are a list of inputs and a list of outputs (a misnomer, since it is actually a list of the variables that use Killspershooterday as an input). These are generated and updated automatically whenever we build the model visually by drawing arrows between nodes. That is, when we first built the model, we drew a diagram. Once the diagram existed, it implied that, when we got around to writing the equation defining Killspershooterday, its definition window (Figure E.5) would already show the inputs to that variable and the variables that use Killspershooterday as inputs. This makes it easier to write the algorithm in the first place because the correctly spelled computer names are available inside the window on menus and one merely has to add plus signs, multiplication signs, or whatever. In this case, the variable is just a product of Effectiveness\_mult, Kps, and Sr.

We know that Kps and Sr are primitive inputs because they are indicated by rectangles. But what is Effectiveness\_mult? If we double click on it, we obtain Figure E.6. Here we see that the overall multiplier of effectiveness, relative to just Kps\*Sr, is a product of a number of individual complicating factors, as discussed in the text. We see corrections for the dash tactic (with the correction depending on which employment strategy Blue is using), a factor indicating the level of risk the pilots are taking, and so on.

For the purposes of this report, let it suffice to note that we could trivially change this equation if we wanted to do so. We would
merely edit it. Or we could go back to the diagrams and indicate what additional variables should affect Effectiveness_mult, then return to this window to edit. That would have the advantage of making the additional variables be present on menus when we edit the equation.

Suppose, now, that we want instead to simplify. Suppose we want to “turn the multipliers off.” All we have to do is replace the definition with “1.” Further, we know that the only effect this will have in the entire model is to change the calculation of Killspershooterday, because that is the only variable that depends on Effectiveness_mult (see the list of outputs at the bottom). We could have seen this also by looking at the diagrams and observing that the only arrow out of Effectiveness_mult goes to Killspershooterday. In any case, as soon as we edit the definition, changing it to 1, the change takes effect.
There is no need to recompile. Instead, as in a spreadsheet, changes are interactive. Another useful feature is that, if one changes the names of nodes—either to sharpen up meaning or, during the design phase, as one thinks more clearly—editing a name anywhere causes the changes to ripple through the entire program automatically.

This should suffice to indicate some of the convenience of the visual-modeling environment, but let us again emphasize that most of the node definitions are much more complex and definitely look like “computer code.” No one should expect to understand the inner workings of EXHALT fully, much less change it, without having a reasonable understanding of programming (e.g., at the level of BASIC). Moreover, much of the power of Analytica involves its use of mathematical arrays. While the arrays greatly simplify the equations and overall cognitive complexity for those who understand arrays, such arrays can be subtle and difficult for those who do not. Our advice to users is to go through Analytica’s excellent tutorial, which uses live models, before attempting to understand fully or to change EXHALT. Even two days of effort can be very useful here.