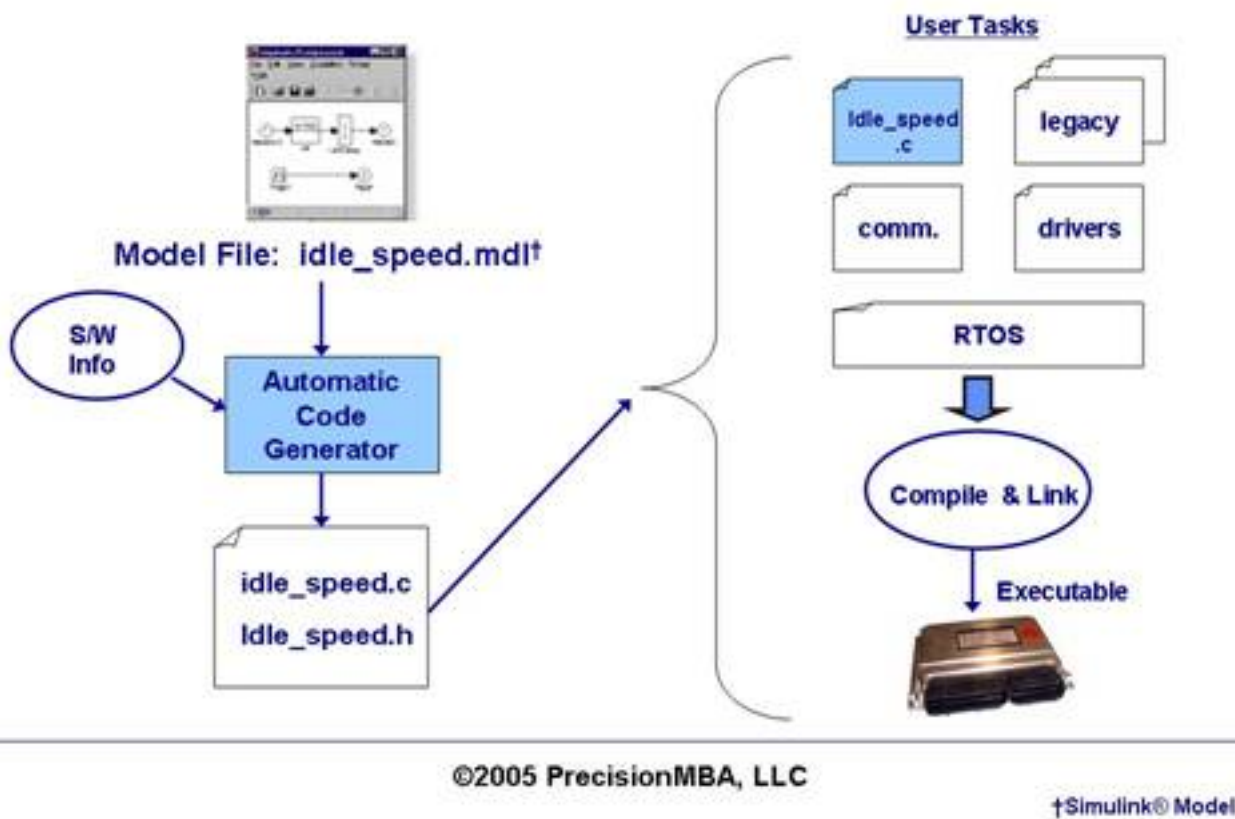


# Automatic Code Generation, a.k.a. "Auto-code"

Automatic code generators are personal computer (PC) applications that read math model files and generate compile-able code that replicates the behavior of the model. For the purposes of our discussion, we will limit the scope to automatic code generation tools whose output is intended to be embedded on production ECUs or FADECs. Let me start off by saying that auto-code is probably not what you think it is. A common misconception is that auto-code tools generate executables for downloading to an ECU. This is not the case. Auto-code tools are C-code (or FORTRAN, or ADA) file generators. They will, for instance, generate code corresponding to a model of an idle speed strategy, but the integration of the resulting C-code into the ECU is a task left to the user. See Figure 1.

## Automatic Code Generation - Defined

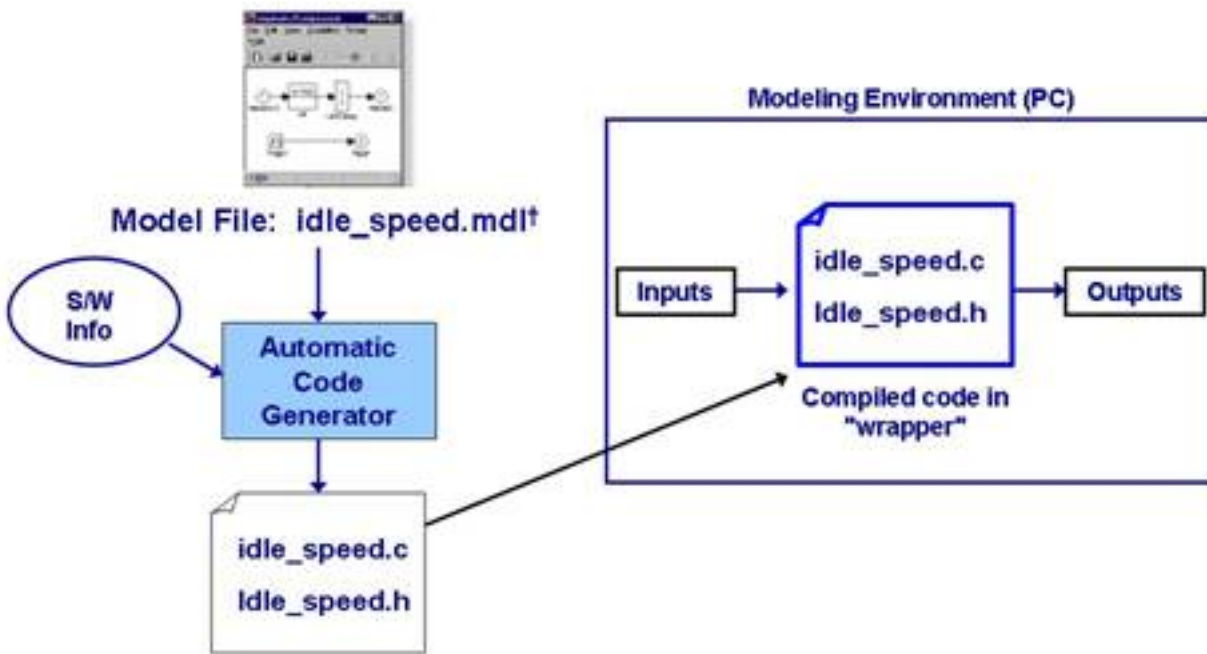


The typical process for using an auto-code tool as follows:

1. The new strategy, such as an idle speed control algorithm, is modeled and tested in a math modeling application. The strategy can then be tested on a real plant, such as a gasoline engine using [Rapid Controls Prototyping \(RCP\)](#).

2. When satisfied with the performance of the algorithm, the user can then invoke the auto-code application.
3. As show in Figure 1, the user must add "S/W Info" to the model, via the auto-code application. The reason is that the model is implemented using double-precision floating point variable types. These types are not computationally or memory efficient if the target ECU is fixed-point, or less than 32-bit.
4. The code generator is not artificially intelligent (yet!). It needs guidance from the user to determine how to generate code. It is possible to set default data types and generate code, but this will not result in optimal code. At a minimum the user must specify the data types of the input variables. The data typing is typically variable width (# of bits), sign, and precision. The width can be 8, 16, or 32 bit. The sign is "signed" or "unsigned." The precision is number of bits past the decimal for fixed point data types. Note that C-code language typing (i. e. double, Char, Int, Uint, etc.) is ambiguous and depends on the compiler. The auto-code generator needs the types explicitly defined.
5. The great benefit of having an auto-code generator built into the simulation environment is that you can simulate the model using expected inputs and then log the output response of all the blocks in the model. This will give you upper and lower bounds on the range of the variables. Knowing the range of the variables, the code generator can "auto-scale" the fixed point data types thus relieving the user of this tedious task.
6. In addition to auto-scaling of fixed point variables, another benefit of developing code in a simulation environment is that the generated code can be compiled and run against the model. This is Software-In-the-Loop simulation (SIL). See Figure 2. SIL lets you determine how close your fixed-point implementation of the model matches the "ideal" double-precision floating point implementation.

## Software-In-the-Loop (SIL) Simulation



©2005 PrecisionMBA, LLC

†Simulink® Model

- Once the code has been verified using SIL the user will then need to integrate the code module into their ECU base code. For example the code module may be a function call or "in-lined" in an existing code file. The ECU code then needs to be rebuilt by cross-compiling all the code modules and linking them together to form the final downloadable executable.

The efficiency of the best auto-code tools on the market today rival that of an experienced human programmer. Auto-code tools were launched into the market in the mid-late 1990s and have steadily matured. In multiple independent studies, they have found to be efficient and reliable. The European OEMs have been quick to adopt this technology, with the Japanese catching up fast. The US OEMs have been slow to use auto-code.

So why aren't they used more often? The US transportation electronics market has been slow to adopt this technology for several reasons. The first of which is that it involves a significant process change over hand-code processes. Key among these is configuration control and inspection. The code is no longer the sole configured item in the process. With an auto-code process, the model needs to be stored as well as its associated "S/W Info" file. Add to this the version control of the modeling program,

and the version of the auto-code tool and you can see it is clearly more involved than storing a date-stamped C-code file.

Existing OEM software development processes are still based on peer code-reviews. An auto-code process assumes that the code does not need to be inspected. In fact, a change to the model may result in changes in several places in the code. Significant changes to the model results in a completely different code file so a "before/after" code review process is no longer relevant.

Automatic code generation is a promising new paradigm on software development. The technology is mature and the performance now, in my opinion, beyond dispute. The challenge now is to change software development processes to adopt this new technology. It is as big a process and mindset change as the move from assembler code to C-code. We now no longer examine assembler code. The auto-code paradigm abstracts the C-code in the same way. We are now relating to a model rather than a code file.

The last reason for slow adoption is that software engineers feel threatened by this technology. These fears are unfounded as there is plenty of specialized implementation information that needs to be crafted with the model. Software engineers would be well served to think of this technology as a new tool to make them more productive in their task. Tools like these do not make the market for skilled engineers shrink, but rather expand the market by enabling more innovative and challenging projects.

If you are considering automatic code generation and need advice, please [contact](#) us.

[Home](#)

©2005 PrecisionMBA, LLC