

Model Style Guidelines for Flight Code Generation

Tom Erkkinen*

The MathWorks Inc., Novi, MI, 48375

This paper describes model style guidelines and best practices for the automatic generation of flight code that is optimized for efficiency, traceability, and clarity. Although the focus is on flight code generation, developers using Model-Based Design with automatic code generation for other applications such as real-time simulation, rapid prototyping, and hardware-in-the-loop testing will benefit from the guidelines provided.

I. Introduction

Embedded flight control systems are increasingly being developed using Model-Based Design with automatic code generation. Using this approach, systems and software engineers model and simulate algorithms using block diagrams, state machines, and data dictionaries. Code is then automatically generated from these models and deployed onto embedded microprocessors in aerospace vehicles. The model architecture, block selection, data objects, and code generation options significantly impact the automatically generated code.

Before the software development process begins, it is useful to have model style guidelines. Model style guidelines assist developers by making the transition from behavioral specifications to flight code straightforward. As with coding standards, organizations usually establish their own model guidelines and approaches to model preparation for flight code generation.

Many certification standards require the use of design and code standards during the development process. For example, RTCA/DO-178B¹ used by the FAA for flight software certification contains the following in Section 5.2.2.1a: *Low-level requirements and software architecture developed during the software design process should conform to the Software Design Standards and be traceable, verifiable, and consistent.*

Organizations should review existing information when creating their own in-house company standards. This paper presents model style guidelines and reference information from aerospace sources as well as sources from other industries. For example, one popular and widely used guidelines document is the MAAB style guidelines². It is used by most major automotive companies worldwide and assists OEMs and suppliers in exchanging models for software specification.

Model guidelines are critical for large and complex projects. Simulation times and model updates can take hours instead of minutes without a model architecture in place that properly partitions the model components. For large projects, it also is important to plan and account for additional development issues such as how to best manage multi-user and multi-location modeling environments. An application is likely to be large-scale if, for example, the model is too big for one person to know all the details, it has more than 10,000 blocks, or it links to over 10 custom libraries. Many of the guidelines described in this paper are targeted toward large-scale models.

II. Guidelines

Before the software development process begins, it is useful to have model style guidelines in place, as described in this section. Model style guidelines provide quality assurance teams with a known standard for assessing the quality of the flight software. As with coding standards, aerospace organizations usually establish their own model guidelines and best practices in preparing models for flight code generation.

* Embedded Applications Manager, 39555 Orchard Hill Place Suite 280, and AIAA Member.

When establishing in-house standards, it is useful to review published guidelines and standards for models and code from organizations including:

- MathWorks Automotive Advisory Board (MAAB)
- Motor Industry Software Reliability Association (MISRA)

MAAB Style Guidelines

The MathWorks Automotive Advisory Board (MAAB) was originally established in 1998 to coordinate product feature requests to The MathWorks from customers including Ford, Daimler Benz (now DaimlerChrysler), and Toyota. The meetings have now expanded to involve most of the major OEMs and suppliers worldwide. An important output of the MAAB has been a set of modeling style guidelines that are publicly available. The MAAB guidelines are written for Simulink®³ and Stateflow®⁴.

The motivation for creating the MAAB guidelines is as follows:

The MAAB guidelines are an important basis for project success and teamwork – not only in-house, but also when cooperating with our partners and subcontractors. Respecting the guidelines is one key prerequisite to achieving:

- *System integration without problems*
- *Clean interfaces*
- *Uniform appearance of models, code and documentation*
- *Reusable models*
- *Readable models*

Each company adopting the MAAB guidelines will often have exceptions, additions, or restrictions for their in-house rules. An example guideline is shown below in Figure 1.

ID Title	db_0042: Ports in Simulink models
Priority	strongly recommended
Scope	MAAB
Automation	possible
Prerequisites	
Description	<p>In a Simulink model, the ports comply with the following rules:</p> <ul style="list-style-type: none"> • Inports should be placed on the left side, but they can be moved in to prevent signal crossings. • Outports should be placed on the right side, but they can be moved in to prevent signal crossings. <p>The name of an inport or outport is not hidden. (“Format/Hide Name” is not allowed.)</p>
Benefit	<p>Respecting the guideline ensures ...</p> <ul style="list-style-type: none"> • Clear interfaces. • Readable models. • Uniform appearance of models. • May eliminate signal crossings
Penalty	<p>Breaking the guideline ...</p> <ul style="list-style-type: none"> • May cause unreadable models.

Figure 1. Example MAAB Style Guideline.

Clearly this MAAB rule, whose benefits include readability and clarity are important to safety-critical software development efforts. Having a clear, concise, and unambiguous specification is one of the main requirements for any Software Design Specification used in the aerospace industry.

Note that the guidelines have not been updated in several years and do not reflect current product capabilities, but efforts are underway to update MAAB guidelines. Although developed by the automotive industry, MAAB guidelines are also used by many aerospace organizations and some aerospace organizations are participating in the MAAB guidelines update effort.

MISRA-C Guidelines

The Motor Industry Software Reliability Association (MISRA) first published the *Guidelines for the Use of the C Language in Vehicle Based Software*⁵ in April 1998. Version 2 of MISRA-C (known as MISRA-C:2004) was released in October 2004. MISRA-C, as it is commonly known, provides guidance for C programming of safety-related embedded systems. As with MAAB guidelines, MISRA-C was created in large part by the automotive industry but it is also used in other industries, including aerospace and defense, due to the pervasiveness of C as the language of choice for developing embedded systems.

MISRA-C does not tailor its guidelines if automatic flight code generation is used. There is debate in the industry regarding this point, especially as companies focus development effort, including inspections, at the model level instead of the code. This process evolution is familiar to those who were involved in the migration from Assembly to C. However, most would agree that MISRA rules focusing on C compiler portability and interpretation are important regardless of whether the code is developed automatically or by hand, as the following example rule illustrates:

MISRA Rule 11 (required): Identifiers (internal and external) shall not rely on a significance of more than 31 characters. Furthermore, the compiler/linker shall be checked to ensure that 31-character significance and case sensitivity are supported for external identifiers.

This rule is important in order to avoid name clashes with identifiers that have long names. This rule also makes code more portable, since most compilers and linkers support at least 31 characters of significance.

For flight code generation, the identifiers are often specified in the model as signal or parameter names. Even though MISRA-C addresses code style guidelines, it makes sense to include relevant rules in a company's model style guidelines, since the model is the input to the code generator. The MathWorks has a MISRA-C information package for the Real-Time Workshop® Embedded Coder⁶. Contact the author for details on how to obtain this.

III. Embedded Software Development

Now that company style guidelines as described in the previous section are in place, the flight software development process can begin. At some point early in the development process, a model moves from advanced research into formal software development. This transformation is where the model becomes a specification, and development activities such as inspections and version control are imposed. Initially, the specification model focuses on establishing and assessing system behavior within an ideal environment, such as double-precision floating point desktop simulations. Later, the behavioral specification is detailed and constrained so that the generated code performs adequately in the embedded environment, such as in a single-precision or fixed-point microcontroller.

Flight software models and executable specifications consist of diagrams and data. The information which follows provides guidelines for both diagrams and data as outlined below.

- Diagrams
 - Solvers
 - Architecture
 - Blocks
- Data
 - Data Dictionary
 - Configuration Sets

A. Diagrams

Model style guidelines for diagrams are needed to ensure that the diagrams are easy to understand, easy to review and check using automation scripts, and support all aspects of Model-Based Design including simulation, code generation, rapid prototyping, and hardware-in-loop testing. With good guidelines in place, design models that have been fully exercised and validated should not need to be rewritten in order to generate code. Some fundamental guidelines that developers need to pay attention to when creating diagrams are discussed below.

Solvers

A system model is a hybrid composition of sensors, actuators, controller (or application), and plant (or environment). Often behavioral models build and simulate the system using continuous-time and variable-step solvers. The reason for this is that the sensors, actuators, and plant exist in a non-digital world where input/output relationships operate in a continuous or analog manner as function of many parameters (for example, atmospheric temperature). These types of relationships are often best modeled using continuous-time blocks and variable-step solvers. However, the embedded controller is the exception. It executes in discrete-time using fixed-step increments.

Converting a controller model to discrete, fixed-step early in the development process saves time and possible rework later. Simulink Control Design includes a model discretizer which automates continuous to discrete model conversion. Developers need to specify the sample rate and transformation method (often Tustin is used with or without pre-warp) to use the discretizer.

Note that the Real-Time Workshop Embedded Coder supports continuous-time blocks. This may be fine for simulation acceleration and rapid prototyping, but for flight code generation it is highly recommended that discrete blocks be used. Also note that code can be generated with variable-step solvers using the Rapid Simulation Target (RSIM). RSIM is useful for simulation acceleration and Monte Carlo batch run studies, but not embedded code.

Architecture

Model guidelines are particularly useful for large or complex projects. Guidelines for creating components and interfaces to components are important for managing large models. For enterprise-level projects, it is necessary to account for additional development issues such as how to best manage multi-user and multi-location modeling environments. Some criteria have been established and published for determining if a project is enterprise wide, as well as describing strategies for managing large scale models⁷.

An application is likely to be enterprise-wide if:

- The model is too big for one person to know all the details.
- It has over ten thousand blocks.
- It links to over ten custom libraries.
- Many team members share a top-level model with multiple components.
- More than five engineers collaborate on one design.
- More than fifteen engineers share the same model.
- Many teams in an organization share reusable components.
- There are more than twenty active configuration controlled models.
- There are multiple active releases of the large design.
- A dedicated tools team supports Model-Based Design products.

A tool that helps you establish the size of the models is the Simulink Diagnostics command in MATLAB®, *sldiagnostics*. Here is an example MATLAB function call and the results:

```
>>[txt,dat]=sldiagnostics('fuelsys','CountBlocks')  
  
fuelsys Total blocks : 258  
          Clock      : 1  
          Constant   : 31  
DataTypeConversion : 1  
          ....
```

The 'Libs' option is also useful since it lists the number of libraries used.

A number of new capabilities are available in Simulink to assist with managing large-scale models including the use of Model blocks, which let one create parent models containing blocks with references to other models. Fundamentally, this model referencing approach allows developers to employ a scaleable component-based architecture within Model-Based Design. In fact, some large corporations are using model referencing techniques to create models approaching a half million blocks.

Model blocks also facilitate a number of other enterprise modeling issues:

- Improves configuration management, since each model referenced by a Model block is a completely separate file and configuration item from the model referencing it.
- Improves loading, updating, and simulation speed, since models are referenced, not copied, and loaded into memory only when necessary.
- Supports incremental code generation by generating code only for the Model block that has changed, which reduces code generation times and minimizes the impact of changes.

This last item is particularly useful to minimize regression testing during maintenance or delta updates of already validated and certified flight code. Figure 2 shows a model dependency graph for a model built using model blocks.

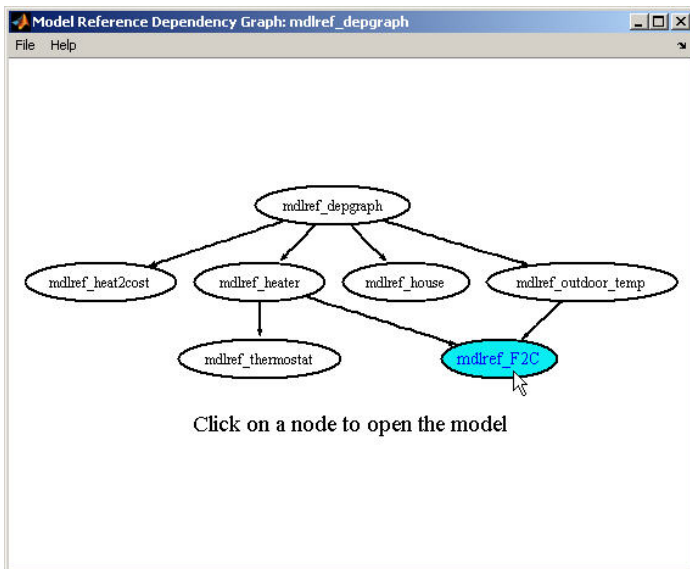


Figure 2. Example model architecture dependency graph.

Bus objects are another new item that helps with enterprise-wide models. Developers can use bus objects to specify the interfaces control description (ICD) between components. An example interface might contain compiled lists of data elements of various data types such as real, integer, and Boolean. Bus objects that are made “non-virtual” are translated into C structures in the generated code.

Blocks

One of the first items new users to automatic code generation want to understand is which blocks are appropriate for embedded applications. The answer is subjective as is often the case with guidelines. Blocks in Simulink, however, that are primarily designed for plant modeling or rapid prototyping environments are likely to not be suitable for flight code deployment. In particular, high-integrity code generation developers typically disallow blocks if the code they generate:

- depends on continuous time
- references non-finite values (Inf, -Inf, and NaN)
- contains instrumentation code only suitable for rapid prototyping

Based on these and other factors, Simulink documents whether a block in the main Simulink library is suitable for production code (i.e., embedded code used in production systems). The Simulink Block Data Type Support table,

shown in Figure 3, shows the data types supported by each block; whether or not code can be generated from the block; and if the block is not recommended for production code.

In addition, some blocks in the table list caveats and notes that should be taken into account when they are used, including:

- This block cannot be used inside a triggered subsystem hierarchy.
- These blocks do not reference absolute time when configured for sample-based operation. In time-based operation they depend on absolute time.
- These blocks generate code that relies on memcpy or memset (string.h) under certain conditions.
- M-file S-functions are not supported for the real-time code generation format. S-functions that make calls into MATLAB are not supported for production code.
- Consider using the Embedded MATLAB block instead.

Sublibrary	Block	Double	Single	Boolean	Base Integer	Fixed-Point	Code Generation Support	
Math Operations	Abs	X	X		X	X	X	
	Algebraic Constraint	X						
	Assignment	X	X	X	X	X	X (N2)	
	Complex to Magnitude-Angle	X	X				X	
	Complex to Real-Imag	X	X	X	X	X	X	
	Dot Product	X	X	X	X	X	X	
	Gain	X	X		X	X	X	
	Magnitude-Angle to Complex	X	X				X	
	Math Function (Exponential)	X	X				X	
	Math Function (log)	X	X				X	
	Math Function (10^u)	X	X				X	
	Math Function (log10)	X	X				X	
	Math Function (magnitude^2)	X	X		X	X	X	
	Math Function							

Figure 3. Simulink Block Data Type Table.

B. Data

Data consists of all the non graphical information used within a model. Model style guidelines for data are needed to ensure that the data is easy to understand, consistent, and properly applied, especially when many diagrams and models act upon a given data item throughout the entire application.

Guidelines discussed herein for data includes the following items:

- data dictionary
- configuration settings

Data dictionary

After the behavioral specification model is established, the development task becomes more focused on the implementation. Detailed data typing, including the use of single-precision or fixed-point data types, is a key activity during this time. The Model Explorer in Simulink lets engineers develop, view, and manage a data dictionary that includes Simulink and Stateflow data. The data attributes can be specified in using various classes of data objects. See Figure 4.

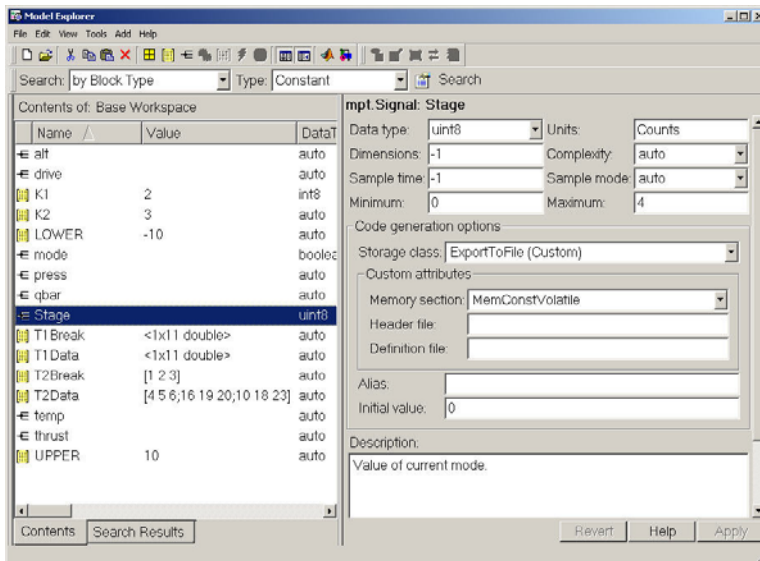


Figure 4. Data Dictionary with MPT Data Objects.

The most basic class is Simulink data objects but another class of objects, MPT data objects, are better suited for flight code generation since they offer a rich variety of data class attributes. For flight code generation, it is recommended that the data object attributes be completely established to limit the possibilities of misunderstanding and reduce the likelihood of inappropriate cast operations.

Simulink has several settings that check for data conversion and cast operations. Developers of high-integrity code should examine these settings and set them accordingly.

Simulink diagnostic categories involving data conversions include:

- Data integrity
- Data conversions
- Connectivity
- Model referencing

Data objects are stored in the MATLAB workspace and can be loaded or saved in files external to MATLAB. One of the more common workflows used by embedded system developers is to have multiple data dictionaries for a given model. This allows developers to target multiple processors using multiple data dictionaries without change the model. This is called a data-dictionary driven model and much literature has been published to this effect⁸.

Configuration settings

Embedded code can be generated once the diagrams have been built and all the data dependencies are resolved. At this point, it is important to review and understand the various code generation options to get optimized code for a particular system’s needs. Optimizations could focus on improving code efficiency, or other items such as traceability. Some settings may cause tradeoffs between efficiency and traceability or readability. For flight code applications, traceability and readability tend to win-out over code efficiency optimizations. For example, if *block reduction* is enabled, it will reduce code size but may make tracing each block to a diagram more difficult.

Simulink’s code generation options are now unified within the Model Explorer’s configuration pane and stored as a configuration set. A model can have multiple configuration sets, but only one configuration set can be active at any given time. Configuration sets can be stored with the model or saved as standalone.

A popular modeling style is to have multiple configuration sets associated with a given model. The appropriate configuration set is activated based on where the development program is in the Model-Based Design workflow.

For example, one may have a single model with the following configuration set options:

- Rapid_Prototyping_Config
- Flight_Code_PowerPC_Config
- Software_In_Loop_Config
- Hardware_In_Loop_Config

In this case, there is one model, using one code generator, with multiple configuration sets for multiple purposes.

One quick way to get a first cut at an optimized configuration is to use a Configuration Wizard block provided in the Real-Time Workshop Embedded Coder block library. To use, simply add a Configuration Wizard block to a model and double click it. An M-file script executes and configures parameters to preset values in the model's active configuration set without manual intervention. There is also a custom configuration block that can be associated with an example M-file script and be adapted to the flight program's requirements.

Software organizations should establish a particular configuration set and use it consistently throughout the life-cycle of the production program. This is similar to how code compiler settings are established and baselined.

IV. Conclusion

This paper introduced model guideline concepts for Model-Based Design with automated flight code generation. Some specific tips and techniques were presented based on experiences of many software organizations in multiple industries including aerospace and defense. Several published guidelines and commercial tools were referenced, providing additional insight and capabilities. Automatic code generation is a fast-paced technology with lots of new capabilities and user experiences. Novice and expert users should plan to stay abreast of the latest features and update style model guidelines as these new technologies are introduced.

V. References

- ¹"Software considerations in airborne systems and equipment certification" RTCA/DO-178B, RTCA Inc., Dec. 1992.
- ²"Controller Style Guidelines for Production Intent Using MATLAB®, Simulink® and Stateflow®," MathWorks Automotive Advisory Board (MAAB), dated April 2001, www.mathworks.com/industries/auto/maab.html.
- ³Simulink, Software Package, Ver. 6.2.1, The MathWorks, Inc., Natick, MA.
- ⁴Stateflow, Software Package, Ver. 6.2.1, The MathWorks, Inc., Natick, MA.
- ⁵"Guidelines for the Use of the C Language in Vehicle Based Software", Motor Industry Software Reliability Association, ISBN 0 9524156 9 0, April 1998.
- ⁶Real-Time Workshop Embedded Coder, Software Package, Ver. 4.2.1, The MathWorks, Inc., Natick, MA
- ⁷"*Strategy for Successful Enterprise-Wide Modeling and Simulation Using COTS*", by Rob Aberg and Stacey Gage, American Institute of Aeronautic and Astronautics GN&C Conference, dated August 2004.
- ⁸"Multi-Target Modeling for Embedded Software Development for Automotive Applications", by Grantley Hodge, et al, Visteon Corp, SAE Technical Paper No. 2004-01-0269, March 2004.