

One of TRNSYS's major strengths is the ease with which users may write new components to expand upon the capabilities of the program. Three features provide the foundation of this expandability. First is TRNSYS's open architecture. With one exception, all standard components are provided along with their source code to act as a reference and to act as the basis for adding new components. Second, all components whether standard components or user written components are formulated in the same manner and follow the same steps and progression throughout the code; writing new components becomes a matter of using a template and adding the appropriate functions, utility calls and equations to write a model of your own. Thirdly, and perhaps most importantly, the TRNSYS kernel does not impose any hierarchy whatsoever on the components that are used in a simulation. Nor does it make any assumptions regarding the order in which components should be solved to simulate the system. Many other simulation packages do impose such a hierarchy. To take the example of simulating a building, they often solve the building loads using one part of their kernel, then proceed to solve the system with another part of their kernel and finally solve the plant aspect of the system last. Many do not revise a previous step based on calculations in a subsequent step. Thus the system could well not meet the load at a given time step. However, the building, at the next time step, has no idea that the load was not met. TRNSYS, by contrast, continues to iterate through all components (whether by successive substitution or Powell's method) until convergence is reached. While this can make controls in a TRNSYS simulation somewhat more cumbersome, it has a major advantage from the point of view of someone writing a new component; that is that there is no need for the component writer to modify the TRNSYS kernel in anyway in order to accommodate and include their new component. There is no need, for example to decide where in the management hierarchy the new component that you are writing fits best. Components in TRNSYS are automatically called by the kernel as soon as they are found in a given simulation. By assigning a number to each component, the TRNSYS kernel anticipates and automatically calls all component numbers that it finds in a simulation input file. lf a particular Type number is found in the input file but is not found in the compiled code by the TRNSYS kernel, a dummy place holding component is called in its stead and the simulation ends with an error.

TRNSYS的核心优势之一在于用户能够轻松编写新组件以扩展程序功能。其可扩展性建立在三大特性之上：其一，开放式架构设计。除个别特例外，所有标准组件均附带源代码，既作为参考模板，也为新增组件提供开发基础。其二，统一的组件构建范式。无论标准组件或用户自定义组件，均采用相同建模逻辑并遵循一致的代码执行流程。开发者只需基于模板嵌入特定函数、调用工具库并编写方程，即可构建专属模型。其三（亦是最关键的特性），仿真内核摒弃组件层级限制。 TRNSYS内核既不强加组件间的层级关系，也不预设系统仿真中的求解顺序——这与诸多同类软件形成鲜明对比。以建筑仿真为例：传统软件通常先由内核某模块计算建筑负荷，再由其他模块依次求解系统及设备参数，且后续计算不修正前序结果。这可能导致特定时间步长的负荷需求未被满足，而下一时间步长的建筑模型却无法感知此偏差。反观 TRNSYS，其通过连续替代法或鲍威尔法对所有组件进行迭代计算直至收敛。虽然这会使控制逻辑略显复杂，却为新组件开发者带来关键优势：无需以任何形式修改内核即可集成新组件。开发者不必纠结组件在管理层级的归属位置——只要组件出现在仿真中，内核即自动调用。内核通过编号识别组件，自动调用输入文件内所有组件编号。若输入文件中的特定类型编号未在编译代码中实现，内核将调用占位组件替代并终止仿真报错。此设计确保了系统模型的动态耦合性，使各时间步长的计算结果能实时互馈，从而显著提升仿真精度。

The following sections of this manual will step the new TRNSYS Type programmer through the process of filling in a Type template in order to create a new model for use in TRNSYS simulations.

本手册后续章节将逐步引导TRNSYS组件开发者完成模板填充流程，从而创建可在TRNSYS仿真中使用的新模型。



At the most fundamental level, a component (referred to as a Type) in TRNSYS is merely a black box. The TRNSYS kernel feeds inputs to the black box and in turn, the black box produces outputs. The kernel takes care of solving the system of black boxes. To delve a little deeper, however, TRNSYS makes a distinction between inputs that change with time and inputs that do not change with time. Examples of inputs that might change with time are temperature, flow rate, or voltage. Examples of inputs that do not change with time are area or rated capacity. In the early days of TRNSYS, this distinction was critical because computing time was very costly and inputs that do not change with time can be set once at the beginning of a simulation thereby saving a good bit of calculation time. With modern computing power, this distinction is less critical but TRNSYS retains the designation of time dependent inputs and time independent inputs. Time dependent inputs are referred to as INPUTS while time independent inputs are referred to as PARAMETERS.

在TRNSYS的底层逻辑中，组件（称为Type）本质上是一个黑箱系统。TRNSYS内核向该黑箱输入数据，黑箱则生成相应输出，内核负责协调整个黑箱系统的求解过程。深入而言，TRNSYS对输入数据进行了关键区分：时变输入：随时间变化的物理量（如温度、流量、电压）；非时变输入：保持恒定的属性值（如面积、额定容量）。这种区分源于早期版本的历史背景：当计算资源极其宝贵时，将非时变输入设为仿真初始常量可显著提升计算效率。尽管现代算力已弱化此需求，TRNSYS仍保留以下术语规范：INPUTS：特指具有时间依赖性的输入变量（首字母大写）；PARAMETERS：表征时间无关的输入参数（全大写）。

At each iteration and at each time step, a component turns the current values of the INPUTS and PARAMETERS into OUTPUTS. No distinction is made among OUTPUTS; all OUTPUTS are assumed to be time dependent and are recomputed by a component whenever appropriate.

在每次迭代的每个时间步长中，组件都会将当前的INPUTS（时变输入）和PARAMETERS（非时变参数）转化为OUTPUTS（输出变量）。需特别说明的是：输出变量不作类型区分：所有OUTPUTS均默认为时变变量；动态计算机制：组件会在满足条件时自动触发输出值的重计算。

TRNSYS has one more input / output distinction; that is DERIVATIVES（微分变量）. Components that solve differential equations numerically will often have DERIVATIVES as well as INPUTS, OUTPUTS and PARAMETERS. From the simulation user's point of view, the DERIVATIVES for a given Type specify initial values, such as the initial temperatures of various nodes in a thermal storage tank or the initial zone temperatures in a multi zone building. At any given point in time, the DERIVATIVES of a Type hold the results of the solved differential equation. If these DERIVATIVES are set in a component as OUTPUTS the simulation user can see, plot and output these values.

TRNSYS 在输入/输出体系中还存在另一关键区分：DERIVATIVES（微分变量）。采用数值方法求解微分方程的组件，除常规的INPUTS、OUTPUTS和PARAMETERS外，还需包含DERIVATIVES。从用户视角看（simulation user's point of view）：初值设定功能：DERIVATIVES 为指定类型定义初始状态（如储热罐节点初始温度、多区域建筑初始室温）；动态承载机制：在任意时刻点，微分变量承载着已求解微分方程的实时结果；可视化输出：当DERIVATIVES被设为OUTPUTS时，用户可实时查看、绘图及导出这些值。

With TRNSYS v. 14.2 and lower, the TRNSYS kernel and Types were all written in FORTRAN and were compiled together into a run-time executable that offered nothing in terms of a graphical interface, online plotting or even a progress bar. As the Microsoft Windows operating system gained popularity and functionality, the TRNSYS structure shifted to that of an exe/dll. In this scenario, a comparatively small executable was written and provided by the TRNSYS developers to handle all of the Windows operating system requirements. The kernel and Types were still written in FORTRAN but were now compiled into a construct called a "dynamic link library" (or DLL for short). The exe and dll spoke back and forth, made use of Windows functionality such as graphical online plotting, yet did not confuse the Type programmer with any of the Windows trivialities. With TRNSYS versions between 14.2 and 16.0, one complexity was that a user wishing to include a non-standard Type in their project had to recompile the DLL to include both the standard code and the non-standard Type (and, incidentally needed a Fortran compiler to do so).With the release of TRNSYS y.16.0, a major effort was made to allow for a multiple dll structure. That is to say that the main exe/dl structure remained but the standard dll could also call out to other dlls and load Types and routines found therein.

在TRNSYS 14.2及更早版本中，内核与组件均采用FORTRAN编写，并共同编译为运行时可执行文件——该架构既不提供图形界面，也无在线绘图功能，甚至缺少进度条显示。随着微软Windows操作系统的普及与功能演进，TRNSYS转向exe/dll组合架构。架构演进关键节点：轻量化可执行文件 (exe)、由官方开发的小型可执行程序处理Windows系统交互需求。动态链接库 (DLL)：内核及组件仍用FORTRAN编写，但编译为DLL模块。协同工作机制：exe与dll通过双向通信调用Windows图形功能（如在线绘图），组件开发者无需介入底层Windows交互逻辑。版本过渡期痛点 (14.2至16.0)：用户集成非标准组件时，需重新编译DLL以合并标准/非标准代码，强制依赖FORTRAN编译器（显著提升技术门槛）。革命性升级 (TRNSYS 16.0+)：实现多重DLL架构，保留主exe/dll框架的同时，支持标准DLL调用扩展DLL，动态加载扩展库中的组件与例程（routines）。

With TRNSYS v. 16.x and lower, the kernel passed certain pieces of information to each Type and recuperated information set by the Types by means a list of arguments in the Type calling statement. Not all of the information required by Types was available by means of the argument list, however. Types got that other information by a hodgepodge of methods such as subroutines, COMMON blocks, access functions, USE statements, and data modules. With the release of TRNSYS 17, the argument list was removed and new access functions were added; Types were expected to use access functions to get all of the data that they need from the kernel.

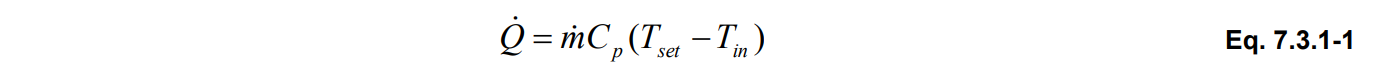
在TRNSYS 16.x及更早版本中，内核通过组件调用语句的参数列表向各组件传递特定信息并回收组件设置的数据。然而，该参数列表无法提供组件所需的全部信息，开发者不得不采用多种拼凑式方法获取额外数据，例如：子程序调用 (subroutines)、公共数据块 (COMMON blocks)、访问函数 (access functions)、USE声明 (USE statements)、数据模块 (data modules)。TRNSYS 17 的架构革新：取消参数列表（argument list）；新增访问函数体系；组件（Types）必须通过访问函数从内核获取全部所需数据。

The remainder of this section of the Programmer's Guide will use the example of a simple liquid heating device to illustrate the process of writing a new component for TRNSYS.

本编程指南的后续章节将以简易液体加热设备为例，逐步演示如何为TRNSYS编写新组件的完整流程。

Consider a simple heater that raises the variable inlet temperature of a flowing liquid to a user defined temperature. Writing an energy balance on the fluid of inlet temperature, , at a mass flow rate :

考虑一个简易加热器，其功能是将流动液体的可变入口温度提升至用户设定的目标温度。对该液体建立能量平衡方程：已知入口温度为，质量流量为，则有：



The first decision that must be made by the Type programmer is: what are going to be the PARAMETERS, INPUTS and OUTPUTS of the model. Two possibilities become apparent. lf the goal of the component is for the end user to be able to specify an inlet temperature and a desired outlet temperature, and in return, find out how much energy was required to bring the liquid from its inlet condition to its outlet condition, then should obviously be an output, while , , and should be INPUTS. , could be designated as either a PARAMETER or as an INPUT depending upon whether or not the liquid specific heat can be assumed to be constant or whether it varies significantly with liquid temperature. lf, on the other hand, the goal of the component is for the end user to provide inlet conditions, a control signal and a heater output that is full ON whenever the control signal is set to ON, then perhaps the output of the model would be , and the inputs would be , , and .

组件开发者的首要决策在于明确定义模型的参数（PARAMETERS）、输入（INPUTS）与输出（OUTPUTS）。此处存在两种典型设计逻辑：（1）当组件目标为允许终端用户设定入口温度与期望出口温度，并据此求解将液体从入口状态加热至目标状态所需的能量时，加热功率应明确作为输出量，而质量流量、设定温度及入口温度应作为输入量；至于比热容，需根据其是否随液体温度显著变化进行辩证处理——若可视为恒定值则定义为参数（PARAMETER），若存在明显时变特性则需归为输入量（INPUT）。（2）反之，当组件目标转变为由用户提供入口条件、启停控制信号及加热器满负荷功率（控制信号为"ON"时恒定输出），此时模型的输出量应为实时出口温度，输入量则对应质量流量、入口温度与实际加热功率。此种设计逻辑的差异本质上取决于组件在系统仿真中的功能定位，开发者需根据能量流与信号流的传递方向动态配置变量属性。

For the purposes of this example, the lNPUTS will be , , and , the OUTPUTS of interest will be , the instantaneous heating rate and the outlet temperature, . The PARAMETERS characterizing the heater will be , , and , allowing the heater to be capacity limited in the amount of energy thatit can deliver to the liguid stream. Note that will also be an OUTPUT so that this information isavailable to other components. We begin at the top of the new component subroutine.

在本示例模型中，输入变量（INPUTS）设定为入口温度、运行设定值及质量流量；核心输出量（OUTPUTS）则包含瞬时加热功率与出口温度。该加热器的特征参数（PARAMETERS）由设备固有参数、流体比热容及最大加热容量构成——其中的引入使模型具备功率上限约束功能，可限制向流体传递的最大能量。值得注意的是，质量流量将同时作为输出量，以确保该关键数据能被系统中其他组件调用。组件开发工作将从新建子程序的顶层结构展开。



Every Type begins with the same line:

所有组件的开发均以相同的代码行作为标准化起点：



Where “n" in TYPEn is a number between 1 and 9999. To avoid problems with more than one component being assigned the same number, the following conventions have been adopted:

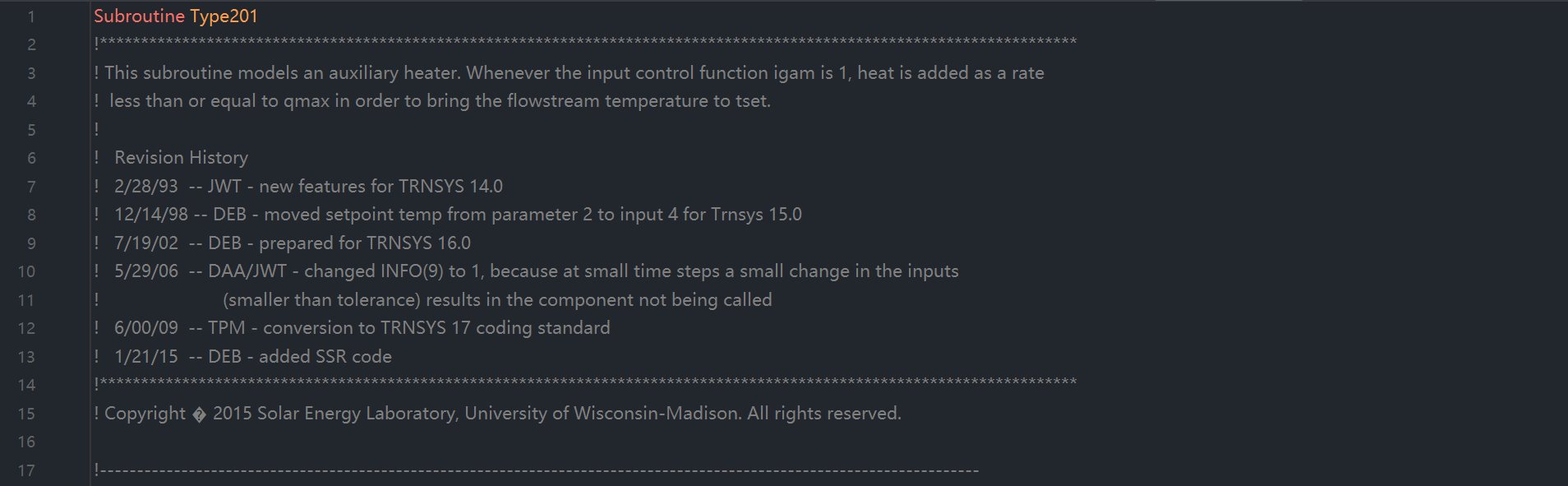
在TYPEn中，编号n作为组件唯一标识符（取值范围1至9999），为避免多组件编号冲突，采用以下管理规程：（n的编号规则详见Document-07-ProgrammerGuide文档Page7-18，此处不再赘述）。

For this example, Type number 201 will arbitrarily be chosen. Thus the first line of the subroutine should be:

在本示例中，选择201号组件。因此，该子程序的首行代码应遵循如下规范格式：



程序截图：





With the release of TRNSYS v. 16.0, a great deal of effort was put into allowing Types to exist either in the standard TRNDll.dll along with the other standard Types or to exist in an External DLL. The TRNSYS16 (and later) kernel examines the contents of a particular user library directory at the beginning of each simulation where all external DLLs must be located if they are to be used. It then loads all files with the extension \*dll, and examines them to find out if there are any exported Type subroutines contained therein. lf it finds exported Type routines it loads them into memory for the duration of the simulation. The main advantage to end users is that they no longer have to recompile the TRNDll.dl file when they receive a new component that they wish to use in their simulation. Instead, they merely need to drop a DLL containing that Type into the appropriate directory. As a Type programmer, there are certain steps that you must follow in order to compile your Type and link it as an external DLL. For more information on linking your component into a DLL, please refer to sections 7.2 (if you are using the built-in Type Studio) or section 7.6 (if you are using the Intel Visual Fortran compiler) of this manual. From a code point of view, however, you need only add one line to your component, right at the top of the file after the SUBROUTINE declaration and before any USE statements. The syntax of the line is:

TRNSYS 16.0版本的架构升级实现了组件的双模式部署机制：开发者可选择将组件集成于标准TRNDll.dll库，或将其编译为独立的外部DLL模块。升级版内核（16.0及以上）在每次仿真初始化阶段，将自动检索特定用户库目录（该目录必须集中存放所有外部DLL），加载所有扩展名为\*.dll的文件并检测其中是否包含可导出的Type子程序；若识别到有效组件，内核将在整个仿真周期内将其动态载入内存。此项革新的核心优势在于显著简化终端用户的操作流程——当获取新组件时，用户无需重新编译整个TRNDll.dll文件，仅需将包含目标组件的DLL文件置入指定目录即可实现即时集成。对于组件开发者而言，若需将组件编译为外部DLL，必须遵循特定的技术流程：使用内置Type Studio开发工具请参阅本手册第7.2节，采用Intel Visual Fortran编译器则参考第7.6节。在代码实现层面，开发者仅需在组件源文件顶部添加单行声明指令（具体位置位于SUBROUTINE声明之后且所有USE语句之前），其标准语法格式为：



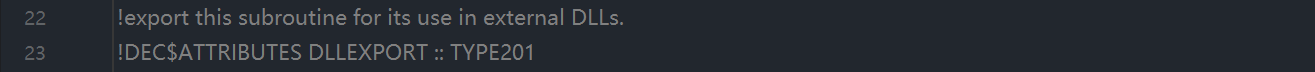
Where n is the Type number. For our example (Type201) we would have the following line added:



It is important that the syntax begin in the leftmost column. In other words, do not put any tabs or spaces to the left of the first exclamation point. The exclamation mark is recognized as a comment by the FORTRAN code. It is, however, noticed by the FORTRAN compiler as an indication that this subroutine should be exported for access by other DLLs. This kind of syntax is called a "compiler directive." It indicates something to the compiler about how to compile the code without affecting the executable FORTRAN itself.

该语法指令必须严格左对齐书写——即感叹号必须置于首列起始位置，其左侧不得存在任何空格或制表符。虽然FORTRAN代码将感叹号识别为注释标识符，但编译器会特别解析此指令，将其视为子程序导出声明，从而使该子程序可被外部DLL调用。此类语法结构被称为"编译器指令(compiler directive)"，其本质是向编译器传递元级操作指示：在不改变可执行FORTRAN代码逻辑的前提下，指导编译过程实现特定功能。

程序截图：





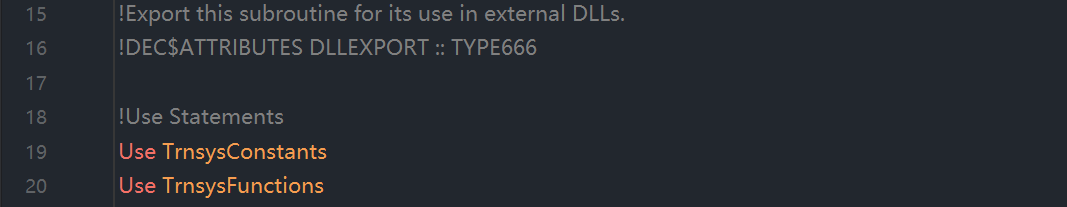
Whereas previous versions of TRNSYS relied heavily on Fortran COMMON blocks, TRNSYS 16 (and later version) Types should access global constants through various Data Modules that are included with the source code. A Data Module is really nothing more than a file that contains a series of variable declarations. They are a convenient way of declaring the variables that are accessible by various subroutines (Types, and kernel routines) within a program all in one location. If a variable contained within a data module is needed by a given component, it is accessed by adding a “USE" statement directly after the DLL Export instruction (see section 7.3.3 above). The syntax of a USE statement is as follows:

相较于旧版TRNSYS对Fortran公共数据块（COMMON blocks）的重度依赖，TRNSYS 16及后续版本的组件应通过源代码附带的数据模块（Data Modules）访问全局共享常量。数据模块本质上是一种集中管理变量声明的文件，其通过统一声明可供程序内各类子程序（包括组件与内核例程）访问的变量，实现全局数据的高效管理。当特定组件需调用数据模块中的变量时，只需在DLL导出指令后立即添加"USE"声明语句（详见第7.3.3节），其标准语法格式如下：



（类似python的 import XXX）

程序截图：



The above has the effect of declaring ALL of the variables or functions contained within the module in your Type. Alternatively, if you only want to use a particular variable or function that is contained within the module but do not want access to all of the variables therein, you can use the syntax:

在组件开发中，直接使用USE ModuleName语句将自动声明该模块内所有变量和函数，使其在当前组件中全局可见。若开发者只需调用模块中的特定变量或函数，同时避免引入冗余标识符污染命名空间，则应采用以下精准导入语法：



（类似python的 from XXX import XXX）

This will only declare the variable(s) listed after the "ONLY" keyword. The advantage of listing only those variables to which we want access is that we can reuse the names of global variables locally as long as we do not request access to them through a USE statement. A complete list of Data Modules accessible from a given Type or subroutine of a Type are listed in sections 7.4.1, 7.4.2. The most common Data Modules that you will need to access from your Type are: TrnsysFunctions, TrnsysConstants, and TrnsysData. TrnsysFunctions contains a series of functions that return commonly useful data such as the simulation start time or time step, or which perform common functions such as reserving a new logical unit number in a global list for use by an external file accessed by your Type. The TrnsysConstants module contains values that do not change throughout a simulation but which are again, of common interest, The TrnsysData module carries a series of data structure sizing constants such as the number of allowable UNITS in a given simulation or the number of allowable EQUATIONS in a given simulation.

这只会声明紧跟在“ONLY”关键字之后列出的变量。仅列出我们想要访问的那些变量的优势在于，只要我们不通过 USE 语句请求访问全局变量，就可以在局部作用域中重用全局变量的名称。可从给定类型（Type）或其子程序访问的完整数据模块列表详见第 7.4.1 节和第 7.4.2 节。您最常需要从类型中访问的数据模块包括：TrnsysFunctions、TrnsysConstants 和 TrnsysData。其中，TrnsysFunctions 模块包含一系列函数，这些函数返回常用数据（例如模拟开始时间或时间步长），或执行常用功能（例如在全局列表中为新逻辑单元号分配资源，以供您的类型访问的外部文件使用）。而 TrnsysConstants 模块则包含在整个模拟过程中保持不变但对许多应用同样重要的常量值。另外，TrnsysData 模块提供了一系列数据结构尺寸常量，例如特定模拟中允许的单元（UNITS）数量上限或允许的方程（EQUATIONS）数量上限。

For the purposes of our example, the liquid heater will only need some of the TRNSYS's Access

Functions. Thus, in the USE statement section of the code, we need only add the following line:



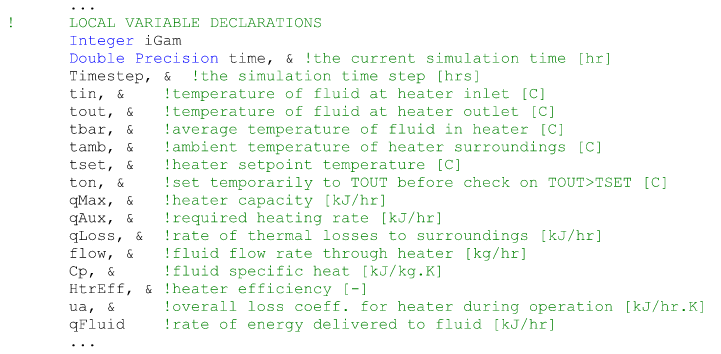
As previously mentioned, it is permissible to list the names of those functions that we wish to access from the TrnsysFunctions module. However, since there is little chance that we will be writing Functions of our own in this component, the chances that the name of a local function and the name of a global function will conflict, is minimal.

正如前文所述，允许仅列出需要从TrnsysFunctions模块访问的特定函数名称。不过，由于在此组件中自定义函数的可能性极低，因此局部函数与全局函数名称出现冲突的情况基本不存在。



Having defined the subroutine, exported the component, and given the component access to global TRNSYS variables, we next need to declare all of the local variables that we intend to use in the Type. From a practical point of view, there is no way to anticipate and simply enter all of the variables that will be needed during the course of developing a new component. The following list is almost always created as the Type is being written and as local variables are needed.

在完成子程序定义、导出组件并赋予组件访问全局TRNSYS变量的权限之后，接下来需要声明该类型（Type）中要使用的所有局部变量。从实际开发角度看，我们无法提前预知新组件开发过程中所需的全部变量，因此以下变量列表几乎总是随着类型代码的编写过程逐步完善——开发者通常边开发边添加所需的局部变量。



For those less familiar with FORTRAN syntax, blue text indicates a keyword that is recognized as a declaration or a function by the compiler. Green text indicates a comment (comment lines in FORTRAN begin with an exclamation mark). The “&" simply indicates a line continuation.

对于不太熟悉FORTRAN语法的开发者而言，蓝色文本表示编译器可识别的声明关键字或函数名称，绿色文本代表注释（FORTRAN中的注释行以感叹号!开头），而"&"符号则专用于表示续行符（即当代码行过长需换行书写时使用的连接标识）。

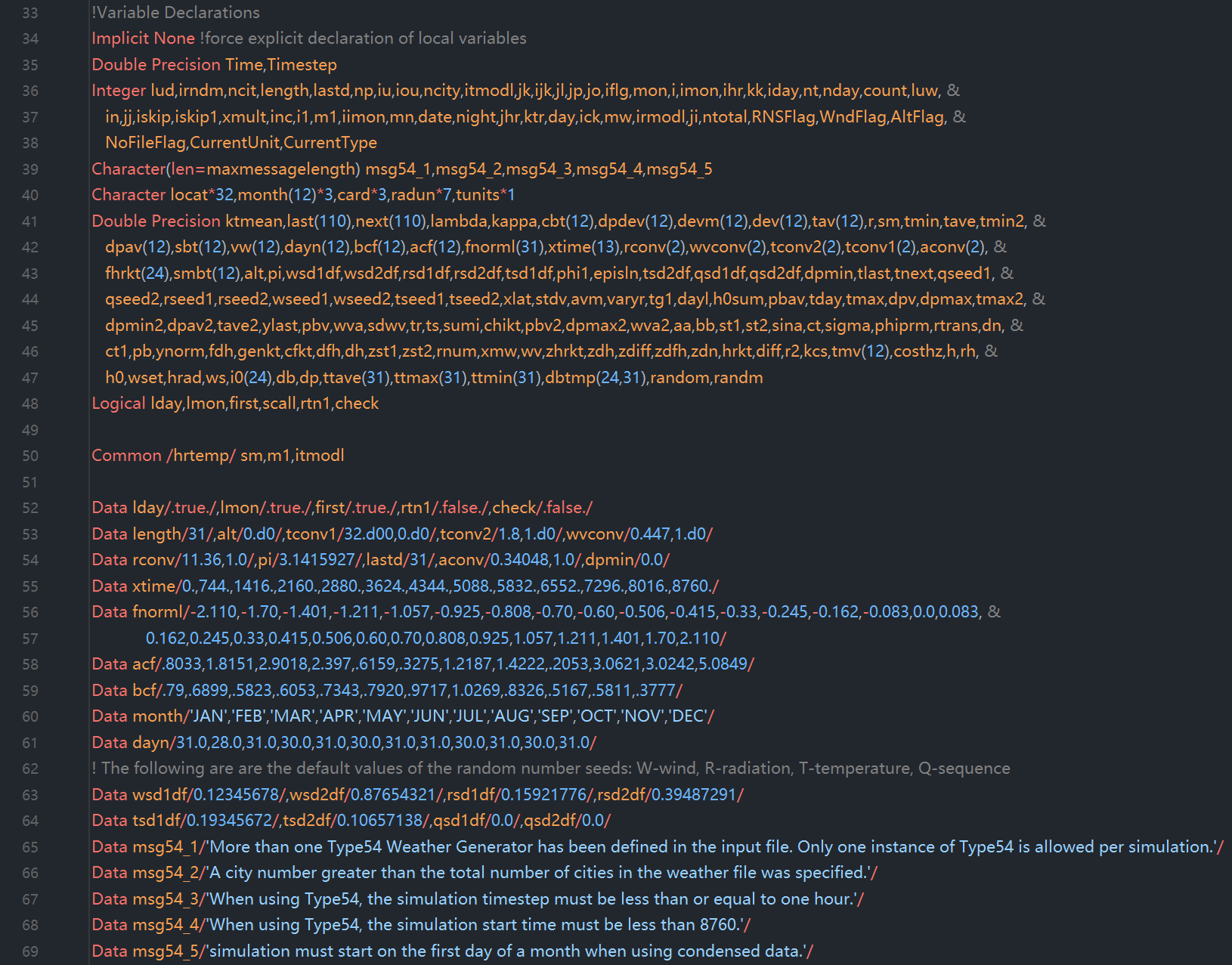


It is advantageous in some Types to set some constants that will be used throughout the Type outside of the executable portion of the code using a DATA statement. The advantage of doing so is that by defining a constant in a DATA statement, that definition is loaded at the simulation start and never is there a time when simulation time has to be spent setting or resetting those values. Often times Type programmers will set a variable called Pl to the value 3.1415926 in a DATA statement, or they will set a variable for the conversion factor between degrees and radians in a DATA statement. Any variable whose value is set in a DATA statement must be declared among the Local Variable declarations discussed in section 7.3.5 above. While no DATA statements are necessary for our Type201 water heater, a DATA statement setting the value of pi and the conversion factor between degrees and radians would look like the following:

在某些类型（Type）开发中，通过DATA语句在代码可执行部分之外设置常量具有显著优势。这种做法的主要价值在于：DATA语句定义的常量会在模拟启动时一次性加载，从而避免在模拟运行期间重复分配或重置这些值所产生的时间开销。类型开发者通常会采用DATA语句将圆周率变量（如Pl）定义为3.1415926，或者设置角度与弧度的转换系数。需要特别注意的是，所有通过DATA语句赋值的变量都必须在7.3.5节所述的局部变量声明区进行声明。虽然我们的Type201热水器模型无需DATA语句，但若需定义圆周率及角度弧度转换系数，其标准语法结构如下所示：



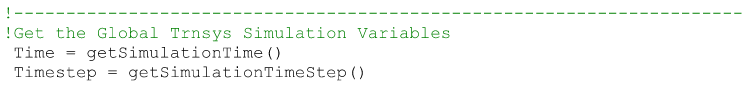
程序截图：



**（可执行代码部分开始）**

We have now arrived at the executable section of the Type; from here on, each line of the code will be executed sequentially, starting at the top. It is therefore necessary to put those lines that should be executed every single time step and every single iteration right at the beginning of the executable section. In the case of almost all Types, the only thing that needs to be done this regularly is to retrieve the values of some global variables that are used in controlling what the Type does at what point in the simulation. Users familiar with coding standard prior to TRNSYS 17 will recall that a good bit of information was passed to the component through the argument list. In the TRNSYS 17 (and later) coding standard, some of this information must now be obtained using Access Functions. Variables such as the simulation time, the time step, the simulation start time, and the simulation stop time, the current Unit number, the current Type number must be retrieved on an “as needed" basis. Almost all Types, this one not included, need to know the simulation time and the simulation time step. These two variables are often used to control Type functionality. Again, the local variable names for these two variables have already been declared in the block of local variables that are required by every Type. The local variable name for the simulation time step is “Timestep" and the local variable name for the simulation time is simply “Time" The code that should be added to Type201 is as follows:

至此我们已进入类型（Type）的可执行代码段；从本行开始，代码将自上而下顺序执行。因此必须将每个时间步长和每次迭代都需执行的代码置于可执行段起始位置。对于绝大多数类型而言，唯一需要如此频繁执行的操作是获取控制组件运行时序的关键全局变量值。熟悉TRNSYS 17之前编码规范的用户应当记得，以往大量参数需通过参数列表传递至组件；而在TRNSYS 17（及后续版本）的编码标准中，部分参数必须通过访问函数（Access Functions）按需获取。诸如仿真时间、时间步长、仿真起始时间、仿真终止时间、当前单元编号及当前类型编号等关键变量，必须根据实际需求通过访问函数动态获取。需特别说明的是（本组件除外），几乎所有类型都需要获取仿真时间和时间步长这两个变量，它们常被用于控制Type的行为逻辑。这两个变量的局部命名已在各类型的标准局部变量声明区预先定义：时间步长变量命名为"Timestep"，仿真时间变量则直接命名为"Time"。以下是Type201需添加的代码示例：



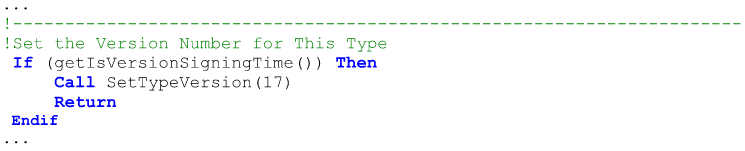
（special calls，特殊调用）

Version signing is one of a series of special calls that is made to a Type when the Type is not intended to perform any manipulations of inputs or outputs. Once, at the very beginning of a simulation, a Type needs to log itself as having been written using a given TRNSYS coding standard. Throughout TRNSYS's history, new requirements have periodically been added to the recommended practices in coding components. These requirements reflect new enhancements that have been added or reflect changes is recommended coding practices that come from the developers of Fortran compilers. Prior to the release of TRNSYS version 16, it was left to the Type programmer to simply make the required modifications before being able to use an existing Type with the new version of TRNSYS. With the release Of TRNSYS16.0, an effort was made to allow for backwards compatibility in Types. That is that the TRNSYS 16 kernel was designed in such a way that it was able to deal with Types that were written to conform with the TRNSYS 16 coding requirements, and to deal with Types that had been written for TRNSYS 15(which have different coding requirements). The same was true of the TRNSYS 17 release; TRNSYS 16 and TRNSYS 15 components (and their specific needs) can be handled by the TRNSYS 17 kernel provided that the Types log themselves properly during the "version signing" step. In order for the kernel to know how to treat a Type, however, each Type must log itself as having been written with TRNSYS 17(or later), TRNSYS 16, or the TRNSYS 15 (and earlier) conventions. For additional information about the differences in conventions, please refer to section 7.5 of this manual.

版本签名（Version signing）是TRNSYS在无需执行输入输出操作时对类型（Type）发起的一种特殊调用。**每次模拟启动之初，类型必须声明其遵循的TRNSYS编码规范标准。**纵观TRNSYS发展历程，组件编码的推荐实践规范会定期更新：这些更新既包含新增功能增强，也反映Fortran编译器开发者提出的编码实践变更。在TRNSYS 16发布前，类型开发者需手动修改现有类型才能适配新版软件；而自TRNSYS 16.0起，系统实现了向后兼容机制——其内核设计可同时处理符合TRNSYS 16编码规范的类型，以及为TRNSYS 15开发（编码规范不同）的类型。TRNSYS 17版本延续此特性：只要类型在"版本签名"阶段正确声明自身规范，TRNSYS 17内核即可兼容处理TRNSYS 16及15版本的组件（及其特定需求）。需特别强调的是，为使内核准确识别处理逻辑，每个类型必须声明其开发遵循的是TRNSYS 17（或更高版本）、TRNSYS 16、还是TRNSYS 15（及更早版本）的规范。相关规范差异详见本手册第7.5节。

For the purposes of the liquid heater example, we are writing a TRNSYS 17 (and later) style component so we must add lines of code that “version sign" our Type as complying with the TRNSYS 17 coding requirements. Note that the coding requirements did not change between v17 and v18 so the following pertains to both TRNSYS version. The code lines that must be added are:

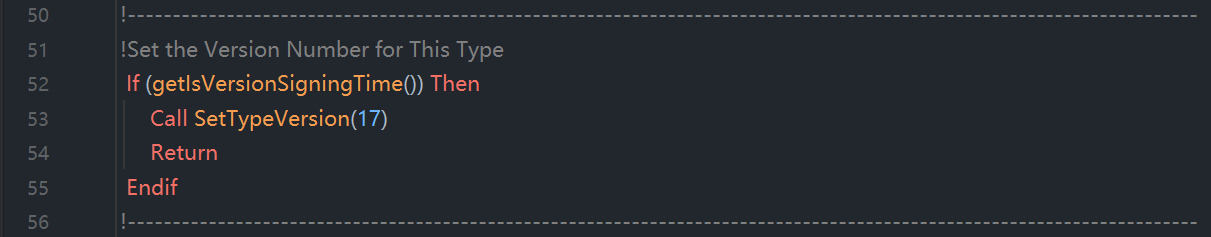
就本液体加热器示例而言，我们正在编写遵循TRNSYS 17+规范的组件，因此必须添加版本签名代码声明本类型符合TRNSYS 17编码规范。请注意v17与v18版本的编码规范完全一致，故下列代码同时适用于这两个TRNSYS版本。需添加的核心代码如下：



If a Type fails to register the coding standard for which it was written, the TRNSYS kernel will not allow the simulation to proceed and will generate an error, writing it to the simulation list and log files.

若类型（Type）未声明其遵循的编码规范，TRNSYS内核将强制终止模拟进程并生成错误报告，该错误信息将同步写入模拟列表文件及日志文件。

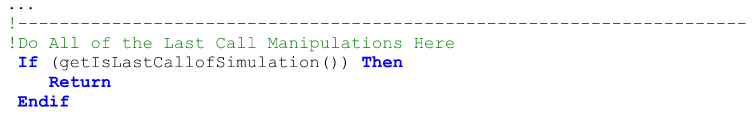
程序截图：



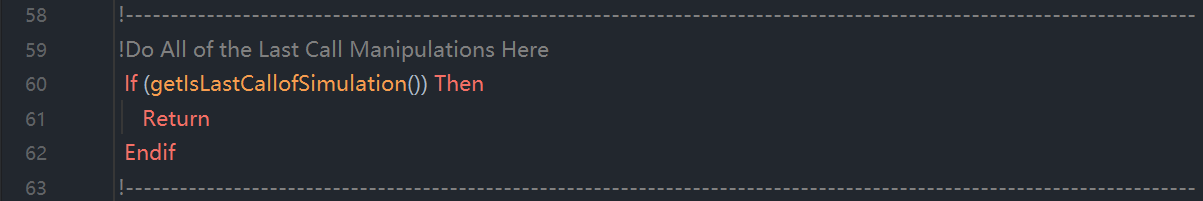
（special calls，特殊调用）

The second special case call to Types occurs at the very END of each simulation. Whether the simulation ends correctly or ends in error, each Type is recalled by the TRNSYS kernel before the simulation shuts down. This call allows Types to perform any "last call" manipulations that may be necessary. These may include closing external data files that were opened by the Type during the course of a simulation, or calculating summary information that may have been collected during a simulation. The Type that we are writing does not require any such last call manipulations, so it is advisable to simply return control directly to the TRNSYS kernel. The code that is added to Type201 is as follows:

第二种特殊调用发生在**每次模拟结束时**。无论模拟正常终止还是异常中断，TRNSYS内核都会在关闭前召回所有类型（Type）。此调用允许类型执行必要的"末次操作"，例如关闭模拟期间开启的外部数据文件，或计算模拟过程中收集的汇总信息。鉴于当前开发的热水器类型无需此类终末操作，建议直接返回控制权至TRNSYS内核。以下是Type201需添加的代码实现：



程序截图：

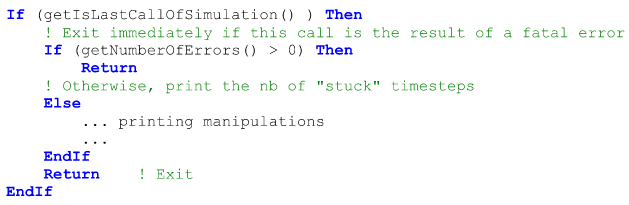


You may want your component to actually do something at the end of the simulation. It could, for instance print a message to the list file or close logical units that were used during the simulation. Standard component Type 22 (Iterative Feedback controller) performs end of the simulation manipulations that you might use as an example.

若需组件在模拟结束时执行特定操作，例如向列表文件输出终止信息或关闭模拟期间使用的逻辑单元号，可参考标准组件Type 22（迭代反馈控制器）的终末操作实现方案。

The Access Function "getIsLastCallOfSimulation()" also returns a "true" result if the simulation terminates with a fatal error. In that case, the component that generates the error returns control to the TRNSYS kernel, which calls all components one last time in order to perform their "end of simulation" operations You can check if the very last call occurs because of an error or as part of the normal simulation process by calling getNumberOfErrors(). Some end of simulation operations are unnecessary or might crash TRNSYS if the simulation ends with a fatal error. .g. Type 22 handles the "last call" as follows:

当模拟因致命错误终止时，访问函数getIsLastCallOfSimulation()同样会返回"true"值。此时，触发错误的组件将控制权交还TRNSYS内核，内核将对所有组件发起最终调用以执行其"模拟终止"处理流程。开发者可通过调用getNumberOfErrors()函数，判断最终调用是由异常错误触发还是正常模拟流程的组成部分。需特别注意的是：若模拟因致命错误终止，部分终止操作可能无需执行或导致TRNSYS异常终止，例如标准组件Type 22按如下方式处理“末次调用”：



Notes:

● If the simulation ends without errors, the "last call" manipulations call happens after the user has allowed the simulation to terminate by clicking on the "yes" or "continue" button at the end of the simulation.

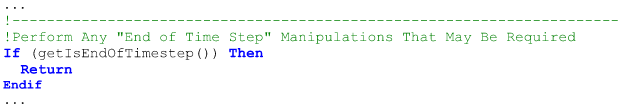
当模拟无错误终止时，系统将在用户点击"yes"或"continue"确认按钮后触发末次操作流程。

● The lines of code here above should be placed before the normal instructions so the return occurs before those instructions are executed.

上述代码行必须置于常规指令之前，以确保返回操作在常规指令执行前完成。



At the end of each time step, each Type in a simulation is recalled. The Type should call the Access Function "getIsEndOfTimestep()". Note that the function will return "true" whether or not a converged solution was found at the current time step. We are keeping the example Type as simple as possible so there will be no manipulations at the end of the time step. Therefore, we can enter the following code at this point in the component:



在每个时间步结束时，仿真中的每个类型（Type）都会被召回。该类型（Type）应当调用访问函数"getIsEndOfTimestep()"。需注意：无论当前时间步是否找到收敛解，该函数都将返回"true"。为使示例类型（example Type）尽可能简洁，此处不会设置时间步结束时的操作流程。因此，我们可以在组件的这个位置输入以下代码：

In more complex components than the one detailed by this example, the "end of time step" call allows a number of different manipulations to be performed such as resetting counters. For users familiar with the concept of manually updating storage variables, the concept of "dynamic" and "static" storage variables were introduced with TRNSYS 17. Refer to section 7.4.4.18. The "end of time step" call is also used by Types that automatically report information to the **simulation summary report (SSR)**. For additional information, refer to section 7.3.10.3.

相较于本示例更复杂的组件中，“end of time step”调用可执行多种操作，例如计数器重置。对于熟悉手动更新存储变量概念的用户，TRNSYS 17版本引入了"动态"与"静态"存储变量的概念，详见章节7.4.4.18。“end of time step”调用机制同样适用于需向仿真摘要报告(SSR)自动提交信息的类型，更多信息请参阅章节7.3.10.3。



It is sometimes advantageous to count the number of times that a particular event has occurred during a given time step. One example might be a controller that counts the number of times that it has calculated a different value of its control signal during a given time step. After a certain number of different decisions, the controller "sticks" and the calculated output state no longer changes. At the end of the time step after all components have converged, it is necessary to reset the iteration counter to zero. This could be done as one of the end of time step manipulations when the call to the getIsEndOfTimestep() Access Function returns "true.”

在特定场景下，统计某事件在单个时间步内的发生次数具有实用价值。例如某控制器需累计其在同一时间步内生成不同控制信号的次数：当决策变更次数达到阈值时，控制器将进入锁定状态（"sticks"），其输出状态不再更新。待所有组件完成收敛后，必须在时间步结束时将迭代计数器归零复位。该操作可通过调用getIsEndOfTimestep()访问函数实现——当函数返回"true"时，即可在时间步结束操作流程中执行计数器重置。



In TRNSYS versions prior to 17.1, another common “end of timestep" manipulation was to update variables that had been stored in the global storage structure during the Type's iterations. With the addition of the concept of “dynamic" storage, this step is no longer needed. Refer to section 7.4.4.18.1.2for information on implementing dynamic storage.

在TRNSYS 17.1之前版本中，常见的" end of timestep "操作涉及更新迭代过程中存储于全局存储结构的变量。随着"动态存储"概念的引入，该操作流程已被淘汰。动态存储的实施指南详见章节7.4.4.18.1.2。



With the release of TRNSYS 18.0 the concept of automatic reports was introduced. If a component is to automatically report either integrated or min/max values, those values need to be updated at the end of each time step. Refer to section 7.3.17.3 for more information about updating automatic report variables.

TRNSYS 18.0版本引入了自动报告机制。若需组件自动上报积分值或最大值/最小值，则必须在每个时间步结束时更新相关数据。自动报告变量的更新规范详见章节7.3.17.3。