

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.

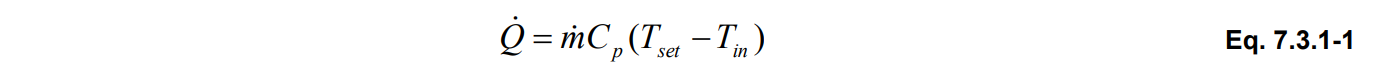
在TRNSYS 16.x及更早版本中，内核通过组件调用语句的参数列表向各组件传递特定信息并回收组件设置的数据。然而，该参数列表无法提供组件所需的全部信息，开发者不得不采用多种拼凑式方法获取额外数据，例如：子程序调用 (subroutines)、公共数据块 (COMMON blocks)、访问函数 (access functions)、USE声明 (USE statements)、数据模块 (data modules)。TRNSYS 17 的架构革新：取消参数列表（argument list）；新增访问函数体系；组件（Types）必须通过访问函数从内核获取全部所需数据。

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 liquid 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:

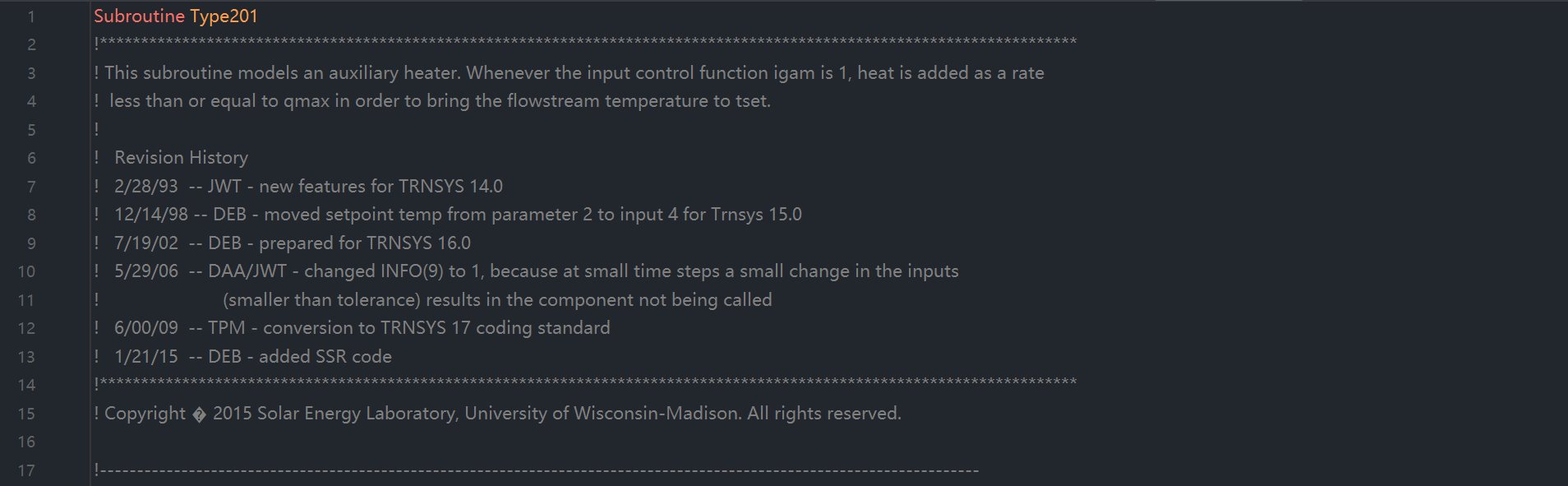
在TYPEn中，编号n作为组件唯一标识符（取值范围1至9999），为避免多组件编号冲突，采用以下管理规程：（n的编号规则详见Document-07-ProgrammerGuide文档Page7-18，此处不再赘述）。

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：



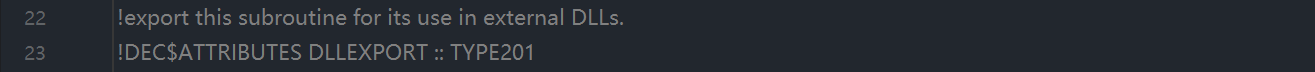
Where n is the Type number. For our example (Type201) we would have the following line added:



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：





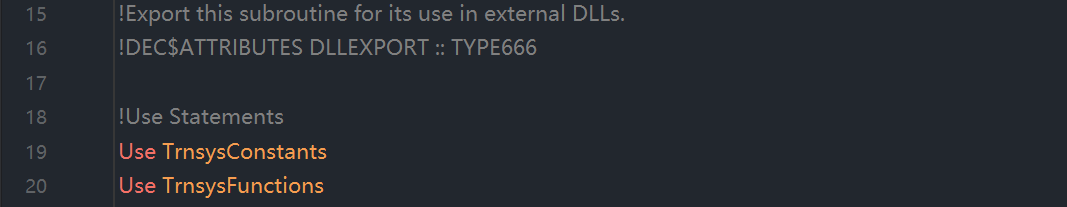
Whereas previous versions of TRNSYS relied heavily on Fortran COMMON blocks, TRNSYS 16 (and later version) Types should access global constants through various Data Modules that are included with the source code. A Data Module is really nothing more than a file that contains a series of variable declarations. 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:



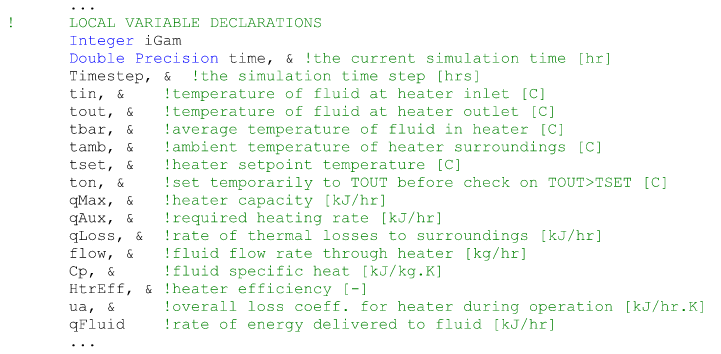
As previously mentioned, it is permissible to list the names of those functions that we wish to access from the TrnsysFunctions module. However, since there is little chance that we will be writing Functions of our own in this component, the chances that the name of a local function and the name of a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

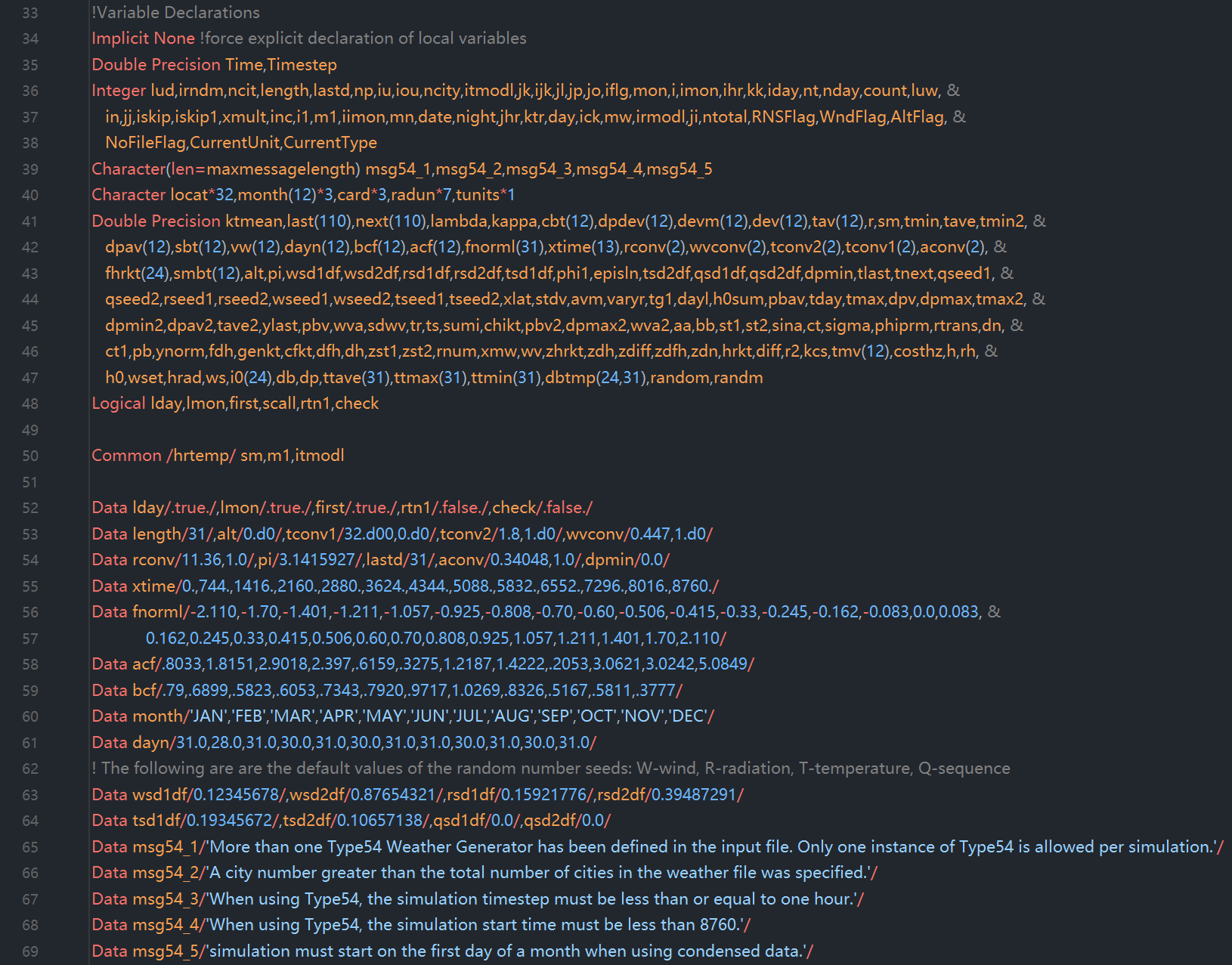


It is advantageous in some Types to set some constants that will be used throughout the Type outside of the executable portion of the code using a DATA statement. The advantage of doing so is that by defining a constant in a DATA statement, that definition is loaded at the simulation start and never 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：



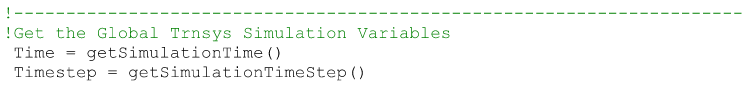
程序截图：



**（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：



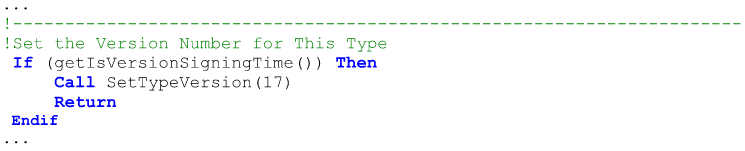
（special calls，特殊调用）

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:

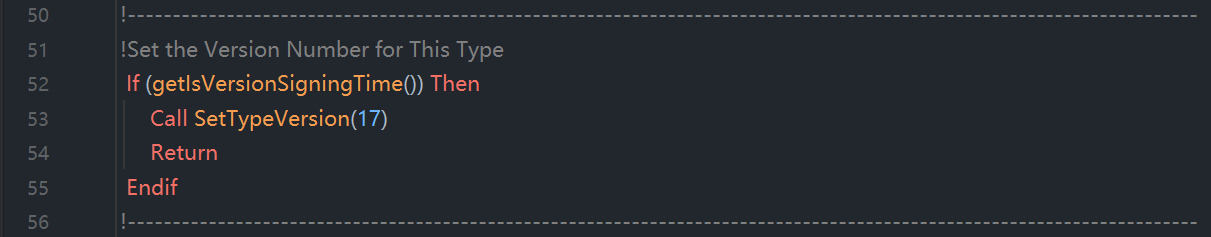
就本液体加热器示例而言，我们正在编写遵循TRNSYS 17+规范的组件，因此必须添加版本签名代码声明本类型符合TRNSYS 17编码规范。请注意v17与v18版本的编码规范完全一致，故下列代码同时适用于这两个TRNSYS版本。需添加的核心代码如下：



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。

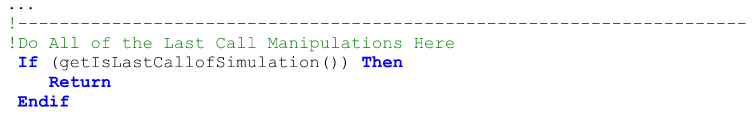
程序截图：



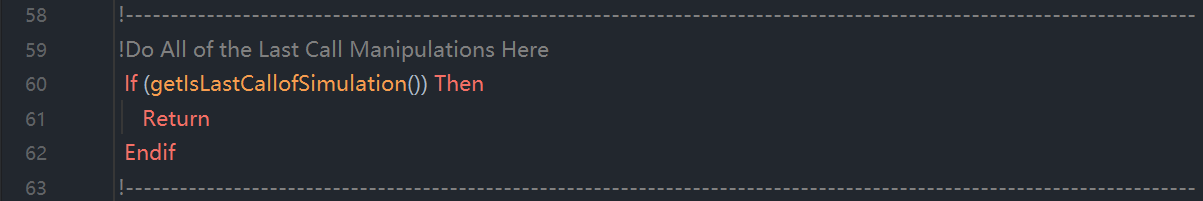
（special calls，特殊调用）

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：

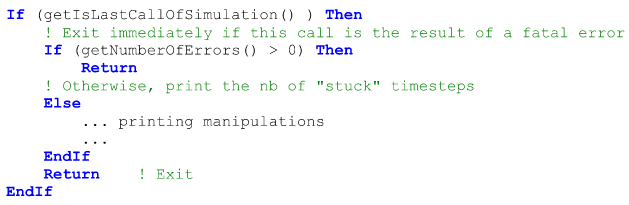


You may want your component to actually do something at the end of the simulation. It could, for instance print a message to the list file or close logical units that were used during the simulation. Standard component Type 22 (Iterative Feedback controller) performs end of the simulation manipulations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

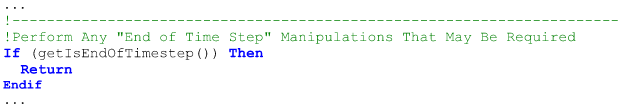
当模拟无错误终止时，系统将在用户点击"yes"或"continue"确认按钮后触发末次操作流程。

● 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。

**说明：SSR是simulation summary report的简称**

（初始化调用）

At the very start of each simulation, after all Types have version signed themselves (see section 7.3.8 or section 7.4.3.3 for more information), each Type should initialize itself. Types should call the “getIsFirstCallOfSimulation()” Access Function to determine when these initializations should take place. There is a very specific set of steps that each Type should take at this point in the simulation. With the release of TRNSYS version 16, the steps taken by a Type during the initialization step changed from what had been done previously. **The most important thing to keep in mind concerning the initialization call is that a Type should NOT read its parameter list, nor should it perform ANY calculation of output values. It should perform the following operations:**

在每次仿真的初始阶段，待所有类型（all Types）完成版本自检后（version signed themselves）（详见章节7.3.8或7.4.3.3），各类型须执行初始化操作（initialize itself）。此时应调用"getIsFirstCallOfSimulation()"访问函数以确定初始化时机，并严格遵循特定步骤序列。需特别注意：自TRNSYS 16版本起，初始化步骤相较此前版本有所变更。**（初始化的调用机制）最关键的原则是：初始化过程中不得读取参数列表，亦不可执行任何输出值计算，仅需完成下列操作：**

Tell the kernel how many PARAMETERS are expected to be found in the input file by calling the

setNumberOfParameters() Access Function as follows:

机制1：通过调用setNumberOfParameters() 访问函数向内核声明参数数量，具体调用方式如下：



Tell the kernel how many INPUTS are expected to be found in the input file by calling the

setNumberOfInputs() Access Function as follows:

机制2：通过调用setNumberOfInputs() 访问函数向内核声明输入变量数量，具体调用方式如下：



Tell the kernel how many DERIVATIVES are expected to be found in the input file by calling the

setNumberOfDerivatives() Access Function as follows:

机制3：通过调用setNumberOfDerivatives()访问函数向内核声明导数数量，具体调用方式如下：



Reserve the required amount of space in the global output array by calling the

setNumberOfOutputs() Access Function as follows:

机制4：通过调用setNumberOfOutputs()访问函数在全局输出数组中预留所需空间，具体调用方式如下：



Set how the Type should be called using a call to setIterationMode(). Most Types will call this Access Function with a value of 1, indicating that the Type should be called every iteration whether or not its input values have changed. Types that are integrators or printers call the function with different values Users programming this kind of Type should refer to section 7.4.3.5 for more information. The call to setlterationMode() is as follows:

机制5：通过调用setIterationMode() 访问函数设定类型调用规则：多数类型应传入参数值"1"，指示无论输入值是否变更，每次迭代均需调用该类型。积分器与打印机类型（例如：Type25）需传入不同参数值，相关开发规范详见章节7.4.3.5。具体调用方式如下：



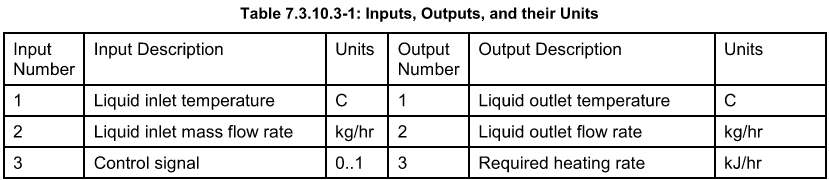
Set the number of static and dynamic storage spots required using a call to the setNumberStoredVariables() Access Function, Refer to section 7.4.4.18 for fundamental information about static and dynamic data storage between time steps. **It is again important to bear in mind that we are not setting initial values of the storage variables at this time, we are merely reserving space for later usage.** For the water heater component, the following statement reserves the required space:

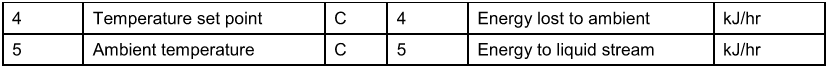
机制6：通过调用setNumberStoredVariables() 访问函数设定静态与动态存储位数量。存储空间的基础理论详见章节7.4.4.18，需特别注意：此时并非设置存储变量初始值，仅作后续使用的空间预留。以热水器组件为例，以下语句实现所需空间预留：



Call the routines that set three-character codes for the units of the INPUT and OUTPUT variables and make sure that the units of the OUTPUTS that are connected to the lNPUTS of this Type are correct. Refer to section 7.4.4.13 for additional information on these three letter codes. Our list of inputs, their units, and our list of outputs and their units is as follows:

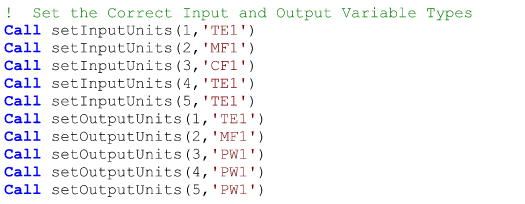
机制7：通过调用专用例程设置输入/输出变量的三字符单位编码，并确保连接至本类型的OUTPUT的单位跟与之关联的INPUT的单位保持一致。三字符编码规范详见章节7.4.4.13，本组件的输入输出单位配置如下：





Referring to Table 7.3.10.3-1 and to Table 7.4.4.13-1 through Table 7.4.4.13-22, the following Access Function calls can be added to the Type, thus setting the units of the first input as a temperature in degrees C, the units of the second input as mass flow rate in kilograms per hour, the third input as a control signal having a value between 0 and 1, etc.

参考表7.3.10.3-1及表7.4.4.13-1至7.4.4.13-22，可在类型（Type）中添加以下访问函数调用（Access Function）来设定变量单位，例如：将首项输入单位设为摄氏度(°C)，次项输入单位设为千克/小时(kg/h)的质量流量，第三项输入设为0-1区间的无量纲控制信号，以此类推。



By calling the setInputUnits() and setOutputUnits() Access Functions, we are allowing TRNSYS to check the users connections between other Types and this one to make sure that they did not inadvertently connect an output with units of power ('PW1' for example) to an input with units of temperature ('TE1'perhaps)

通过调用setInputUnits()和setOutputUnits()访问函数，TRNSYS可**自动校验用户跨类型连接的正确性**，防止误将功率单位输出（如'PW1'）接入温度单位输入（如'TE1'）等错误。

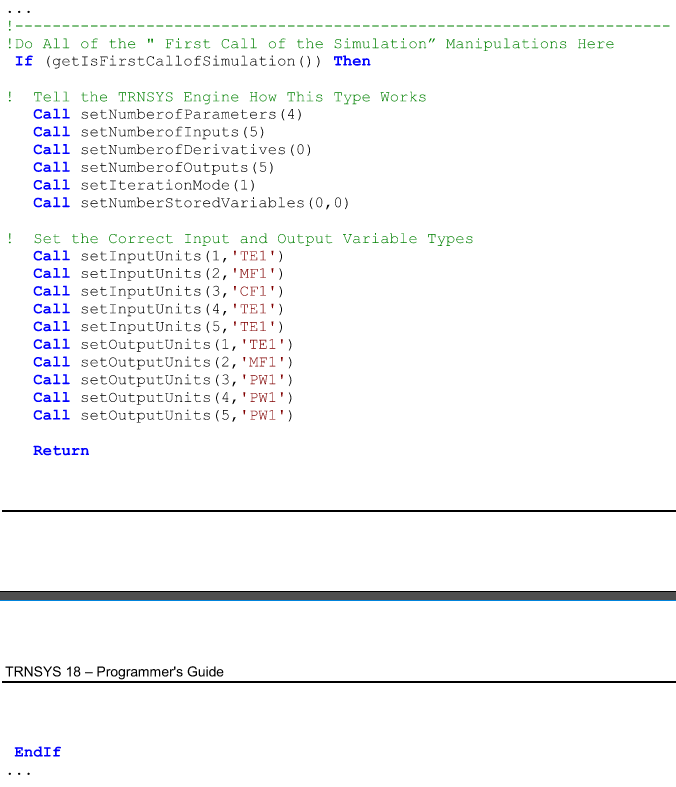
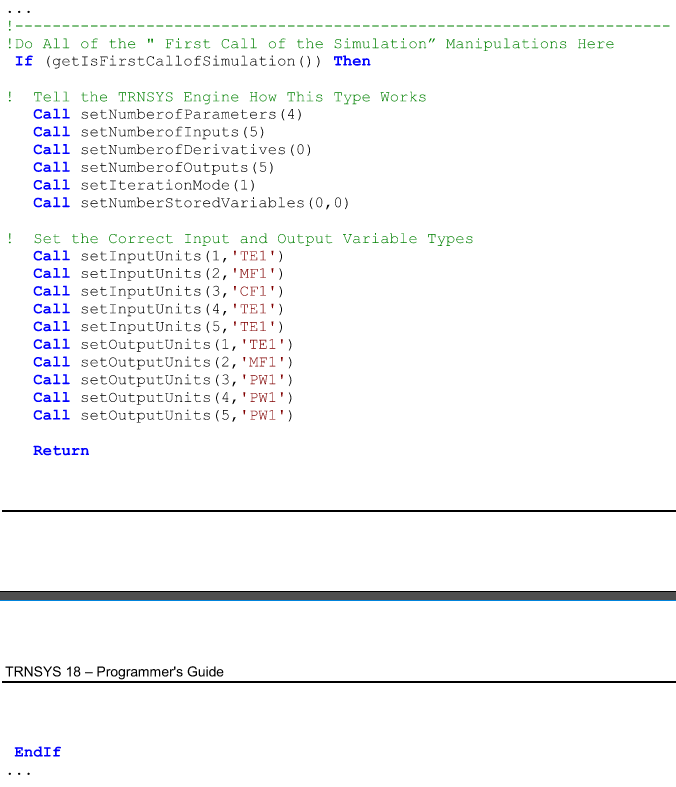
Return control to the TRNSYS kernel. Again, there are NO iterations performed during this time step. Therefore there should be NO calculations performed. To return control, add the following line.

完成配置后需立即将控制权交还TRNSYS内核：此阶段不得执行任何迭代运算，亦不可进行任何计算操作。交还控制权需添加以下代码：（Return代码）

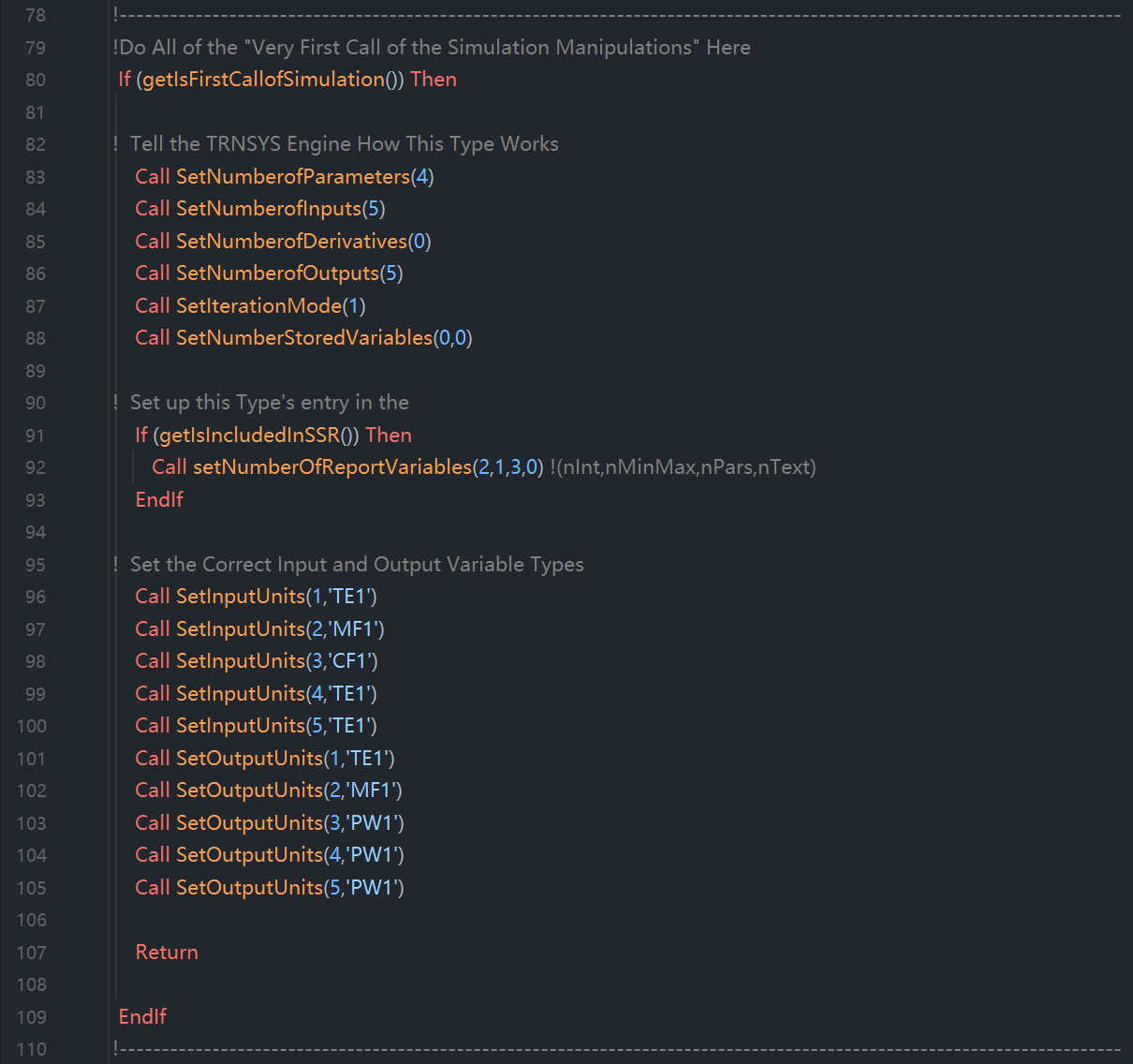


The code required for the above steps in the context of Type201 is as follows. Please note that the following code is fairly generic and can be copied for any Type that does not have a variable number of parameters, inputs, or derivatives allowed in the input file. The user will, of course, have to correctly adjust the arguments to the various Access Functions so that they correspond to the requirements of their Type.

在Type201组件中实现上述步骤的代码如下。需注意：以下代码框架具备通用性，可复用于任何参数/输入/导数数量固定的组件类型。开发者需根据具体需求精确调整各访问函数的参数值，确保其与组件规范完全对应。



程序截图：





During the initialization step any Types that are to be reported in the Simulation Summary Report (\*.SSR) must reserve the number of spots that they wish to later use. For additional information about the required programming, please refer to sections 7.3.17.1 and 7.4.2.101.

在初始化阶段，需在仿真摘要报告(\*.SSR)中上报数据的类型必须预先预留报告位。具体编程实现要求详见章节7.3.17.1与7.4.2.101。



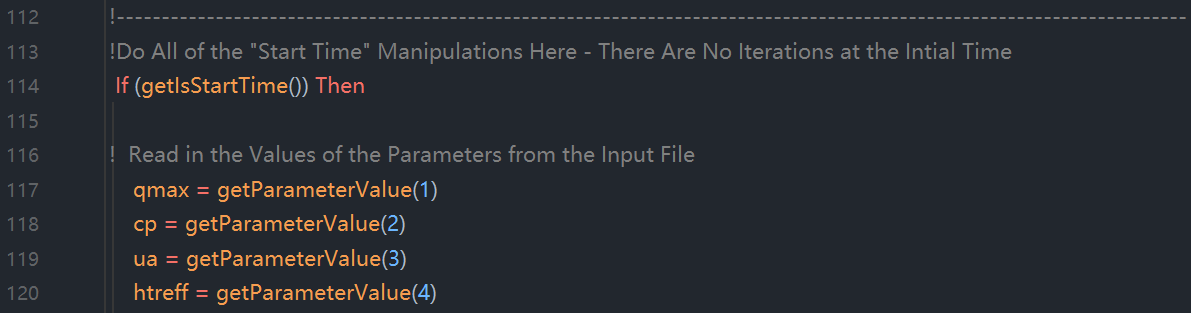
Once the initialization steps (discussed in section 7.3.11 above) were completed, control was returned to the TRNSYS kernel. The next time that each Type is called is when the simulation time is equal to the simulation start time. Throughout much of TRNSYS's history (through version 15) there was no call to components at the simulation start time. After being called when getIsFirstCallOfSimulation() returned “true" for initialization (as described in section 7.3.11) Types were next called at the end of the first time step (TIME = InitialTime + TimeStep). With the release of TRNSYS 16.0, an important distinction was made by calling all Types once after they have been initialized but before time has advanced in the simulation. The point of this initial time call is to perform the following actions:

完成初始化流程（见7.3.11节）后，控制权已交还TRNSYS内核。各类型将在仿真启动时刻（simulation time is equal to the simulation start time）首次被调用——此处需注意版本演进差异：TRNSYS 15及更早版本，无启动时刻调用环节；类型初始化后（getIsFirstCallOfSimulation()返回"true"）；直接跳转至首时步终点调用（时刻=初始时间+时步长）。TRNSYS 16+革命性变更，新增初始时刻调用阶段：执行时机：初始化完成后 → 时间推进前；调用范围：所有类型；核心目的：执行下列关键操作。

First, read the local parameter list and set each parameter to a local variable. Parameter values can (and should) be read only at two places in a Type. The first is during this "Start Time” call before time has progressed. The other is during the "multiple unit manipulations" section (discussed below in section 0). Only reading parameters at these two stages of the simulation saves a great deal of computational time because in this manner, parameters (which do not change with time) are read only when they must be. In each of these two sections, each parameter value is set equal to a local variable name that has been declared as a double precision variable among the Type's local variables as discussed in section 7.3.5. The generalized code for setting a parameter value to a local variable name is 

首先，读取本地参数列表（local parameter list）并将每个参数值赋予本地变量（local variable）。参数读取操作在组件中仅允许（且应当）发生在两个特定节点：其一是在时间推进前的本次" Start Time "调用阶段，其二是在后文章节0所述的" multiple unit manipulations "阶段。将参数读取严格限定在这两个仿真阶段可显著节省计算资源——由于参数值随时间保持不变，此机制确保时不变参数仅在必需时被加载，从而避免冗余操作。在上述两个阶段中，每个参数值均需赋值至预声明的双精度本地变量（变量声明规范详见章节7.3.5），其通用代码实现形式为：

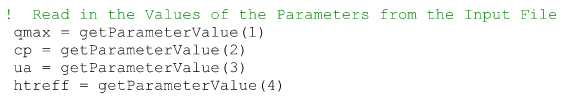
程序截图：



该做法的本质就是将参数值（Parameter Value）赋值给局部变量，如上图所示。

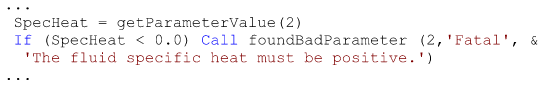
Type201 has four parameters, one each for the maximum heating rate, the specific heat of the working liquid, the overall loss coefficient of the heater, and the heater efficiency. Referring to the code at the end of this section, each of these four parameters is set to a local variable name: QMAX, CP, UA, and HTREFF respectively as follows:

Type201组件包含四个核心参数：最大加热速率、工作流体比热容、加热器整体热损系数以及加热器效率，各参数分别通过以下代码赋值至预定义的本地变量QMAX、CP、UA与HTREFF，具体实现可参考本节末尾的代码示例：

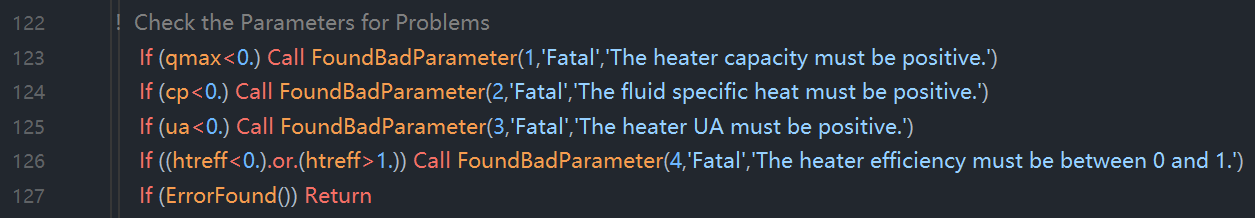


The second step performed during the “start time” manipulations section is to check each of the parameter values for validity. Parameters whose values cannot be negative, or whose values must be between 0 and 1, can be flagged as an error using a call to the foundBadParameter() Access Function. Take, for example, the specific heat of the working liquid for the heater. It does not make any physical sense to allow the user to enter a negative value of specific heat. We could use the following code to detect and report the problem, assuming that the second parameter is the specific heat and has been set to a local variable called SpecHeat.

在“start time”操作阶段执行的第二步是校验各参数值的有效性：针对物理层面不可为负值（如工作流体比热容）或必须处于0至1区间的参数，可通过调用foundBadParameter()访问函数标记为非法输入。以加热器工作流体比热容参数为例，允许用户输入负值比热容在物理层面完全无效，若该参数被定义为第二输入项且已赋值至SpecHeat本地变量，则可采用以下代码实现异常检测与报错：



程序截图：



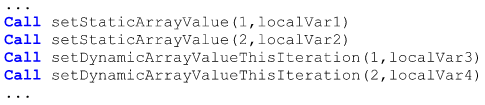
Once foundBadParameter() Access Function has reported the flagged parameter and printed its error message to the simulation log and list files, it returns control to the calling Type but unfortunately does not return an argument that tells the Type that an error message was written. It is good practice for a Type that has called foundBadParameter() with a bad parameter to immediately return control to the TRNSYS kernel without completing its "start time” manipulations in case the bad parameter could cause other problems before the end of the call. It is therefore recommended that once all the parameters have been checked, programmers use the ErrorFound() Access Function to determine whether foundBadParameter wrote any error messages to the simulation list and log files. For more information on the ErrorFound() Access Function, please refer to section 7.4.2.3. The code for returning control in this manner is:



当foundBadParameter()访问函数完成非法参数标记并将错误信息写入仿真日志与列表文件后，其执行流程将返回至调用类型，但需注意的是：该函数不会返回任何指示错误信息已写入的状态参数。为确保代码健壮性，若某类型因参数非法调用foundBadParameter()，应立即中止当前" start time "操作并将控制权交还TRNSYS内核，此举可规避该非法参数在调用结束前引发次生问题。因此建议开发者在完成所有参数校验后，通过ErrorFound()访问函数检测foundBadParameter()是否已向仿真文件写入错误信息（具体函数规范详见章节7.4.2.3），其控制权交还的实现代码如下：

Next, it is necessary to set the initial values of any static or dynamic storage variables that are used by the Type. This is accomplished simply by assigning local variable values to each of the spots in the array that is used to transfer information to and from the storage structure, then calling the setStaticArrayValue() and setDynamicArrayValueThislteration() subroutines. The code used to set four initial storage values might look like the following:

在组件初始化流程中，必须为所有静态及动态存储变量设置初始值。该操作需通过两步实现：首先将本地变量值逐一赋给存储结构传输数组的对应槽位，随后调用setStaticArrayValue()与setDynamicArrayValueThislteration()子程序完成存储初始化。以下代码展示了四个存储变量初始化的典型实现方案：



Since our Type201 does not require any storage variable spots, there is no need to include the above step in this section of the Type201 code.

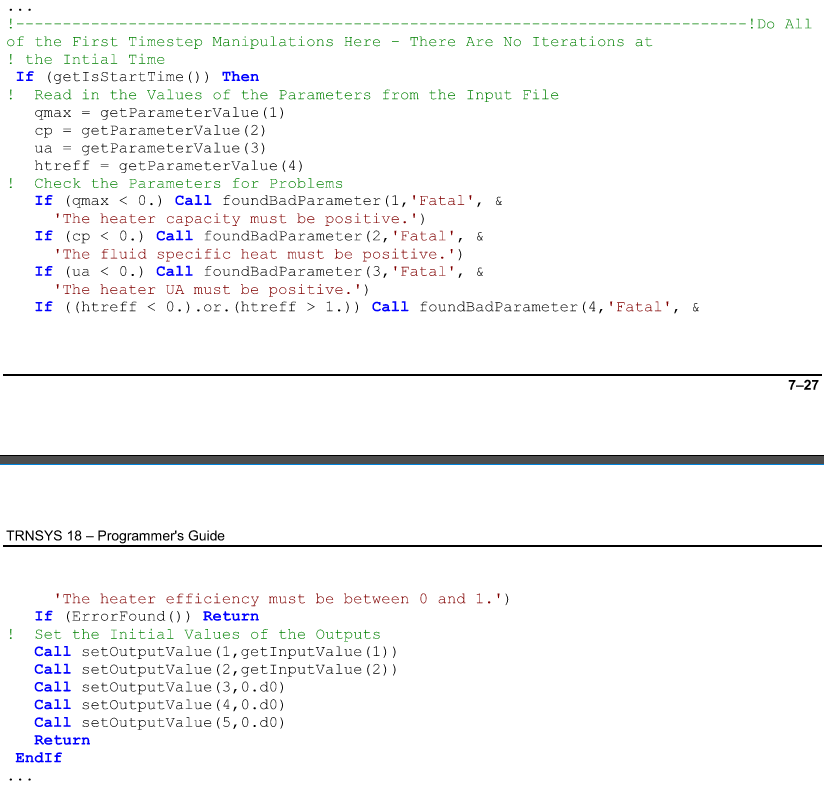
由于Type201组件无需任何存储变量槽位，故无需在代码中实现前述存储初始化步骤。

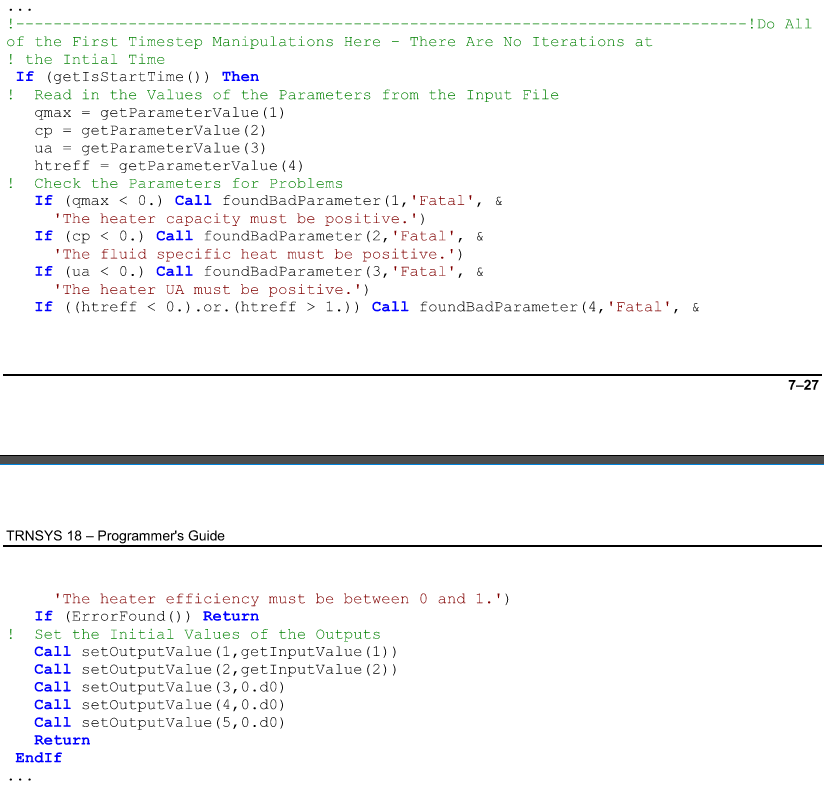
The final step performed during the "Start Time" manipulations section is to set the initial values of the outputs. It is essential that you set initial values here. Your Type should not perform any of its calculations at this point in time. If it does, then it will likely get one time step ahead of where it is supposed to be at this point in the simulation. Because of this restriction on performing calculations, it is often difficult to know what value to choose for an output initial value. A good default value is zero since that usually indicates to the user that the output has not yet been calculated. In the case of devices that have fluid flow through them, the outlet temperature and outlet flow rate of the fluid can often be set to the inlet temperature and inlet flow rate respectively. In the case of Type201, the first two outputs can be set to the input initial values in this manner. The other outputs cannot be calculated easily and so are simply set to zero.

在" Start Time "操作阶段的最终环节必须设置输出初始值，此时严禁执行任何计算操作，否则将导致组件状态超前推进一个时步，破坏仿真时序。鉴于计算禁令的存在，输出初始值设定常面临决策困境——推荐采用零值作为默认方案（通常向用户传递"尚未计算"的状态信号）。对于流体设备类组件，出口温度与流量可分别设为入口对应值；Type201的前两项输出即按此原则参照输入初始值设定，其余输出项因计算复杂度直接设为零值。

Below is the entire code for the “Start Time" Manipulations section of Type201.

以下是Type201完整的"启动时刻"操作段代码实现：







During the initialization step any Types that are to be reported in the Simulation Summary Report (\*.SSR) should initialize (set up) the specifics of the values that they wish to report. For specific convections and requirements, refer to section 7.3.17.2

在初始化步骤中，任何需要在仿真总结报告（\*.SSR）中报告的类型均应初始化（set up）其希望输出的数值。具体规范和要求请参阅第7.3.17.2节。

完整的程序截图如下：





Only one more special manipulation section is required before we begin actually simulating the performance of the liquid heater. In this “Multiple Unit" manipulations section, we must reread our components parameter list if and only if there is more than one instance of the component in the user's simulation.

在开始实际模拟液体加热器的性能之前，还需完成一个特殊的操作步骤。在这一“多单元实例”（Multiple Unit）操作部分中，当且仅当用户的仿真模型中存在该组件的多个实例时，必须重新读取该组件的参数列表。

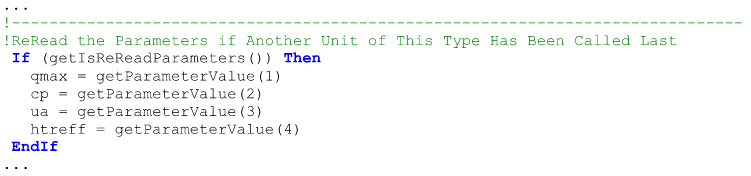
As usual, we begin by querying an Access Function:

与往常一样，我们首先调用一个访问函数（Access Function）：getIsReReadParamters()



If the function returns "true" then we know that we need to reread that parameter list so that we are sure that all local variables are set to the current instance of the Type's parameter lists. The code is as follows:

如果该函数返回“true”，则表明我们需要重新读取参数列表，以确保所有本地变量均与当前组件实例的参数列表保持同步。代码如下所示：



There is no need to recheck the parameters for validity since this was done already during the “Start Time" manipulations section.

由于在“启动时间（Start Time）操作部分”已对参数有效性进行了验证，因此无需重复检查。



At last we have come to the meat of writing new components. It is in this final section that the performance of the component is simulated. Because every component is different, the rules for what steps go in this section become much more flexible. That said the "Every Time Step” manipulations section can be broken up into four basic categories which proceed in order as follows.

终于我们来到了编写新组件的核心部分。在这一最终操作阶段中，组件的性能将被模拟。由于每个组件的设计各不相同，该部分的步骤规范也变得更为灵活。尽管如此，"每个时间步长（Every Time Step）"操作部分仍可划分为四个基本类别，且需按照以下顺序依次执行：

（1）retrieve stored values：检索计算过程中所需的存储值

（2）retrieve input values：

（3）perform calculation：

（4）set storage values：

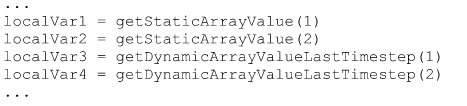
（5）set outputs：

（6）return control：



At each iteration, a call should be made to the getDynamicArrayValueLastTimestep() and getStaticArrayValue() functions to retrieve stored values that are needed in calculations. The call might look like the following:

在每次迭代过程中，应调用 getDynamicArrayValueLastTimestep()（动态数组前一时间步长值获取函数）和 getStaticArrayValue()（静态数组值获取函数），以检索计算过程中所需的存储值。该调用的示例如下：



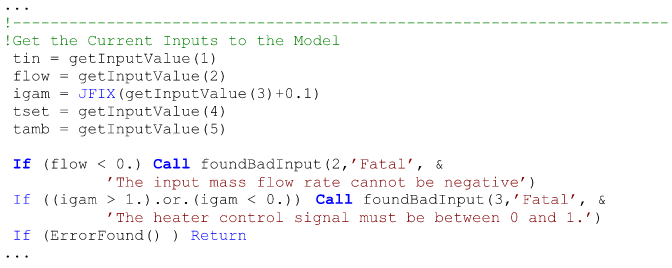


Retrieving input values follows much the same pattern as that of retrieving parameter values. The inputs are available to each Type by means of the getlnputValue() Access Function. The values are typically read to local variable names for convenience and then can be checked for validity in much the same way as parameters.

输入值的获取与参数值的获取方式基本相同。每个组件类型（Type）均可通过 getInputValue() 访问函数来获取输入值。为了便于操作，这些值通常会被读取到本地变量中，之后也可以像参数一样对其进行有效性检查。

Our Type201 has five inputs. However, only two of them have restrictions on their values. Namely, the flow rate should not be negative, and the heater control signal cannot be less than zero or greater than one. The section that sets the inputs to local variables and checks them might look like the following.

我们的 Type201 组件共有五个输入。然而，其中只有两个输入对其数值范围有所限制：具体来说，流量不应为负数，而加热器控制信号不能小于零或大于一。用于将输入值赋给本地变量并进行检查的代码部分可能如下所示：



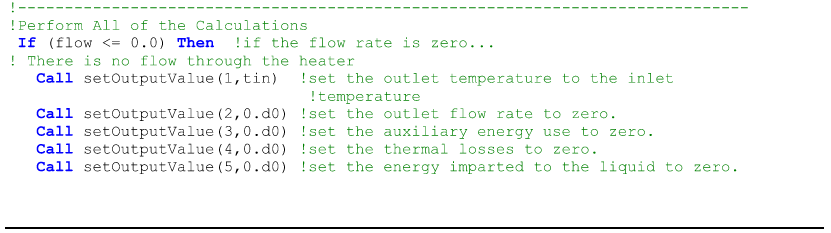
Here the 2 and the 3 in the foundBadlnput() call indicate the input number that is the problem in each case.

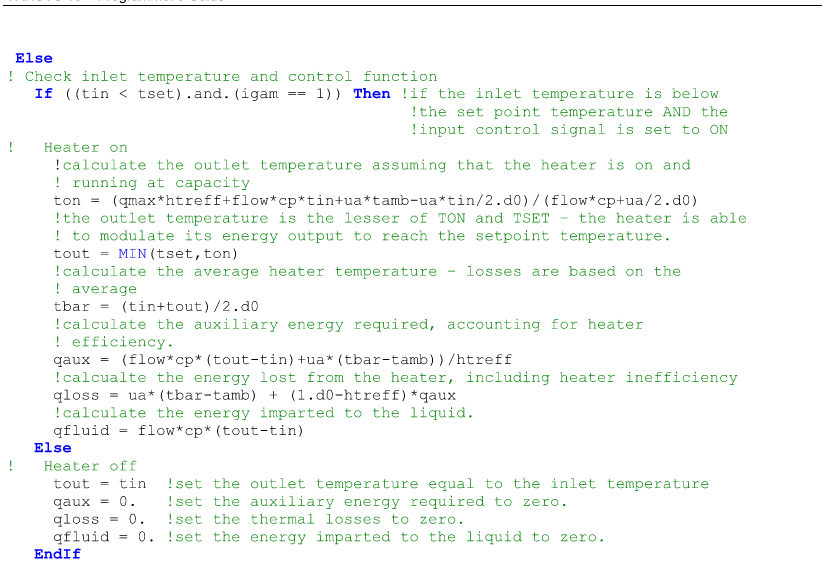
此处 foundBadInput() 调用中的数字 2 和 3 表示在各自情况下存在问题的输入编号。



Once all of the input values have been retrieved, the heater performance calculations and control logic can be performed. Each line in the following code is commented to show its purpose.

一旦所有输入值均已完成获取，即可执行加热器的性能计算与控制逻辑。以下代码中的每一行均通过注释说明其功能目的。







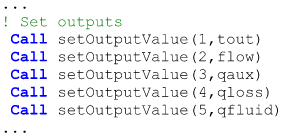
Before exiting the Type at each iteration, it is good practice to update all appropriate storage variables. Obviously, if one or more of the storage variables is an initial value to be used for all iterative calculations at a given time step until convergence has been reached then it is not appropriate to update this value until the “End of Timestep" call (see sections 7.3.10 and 7.4.3.9 for additional information.)

在每次迭代结束组件类型（Type）之前，建议更新所有相关的存储变量。显然，如果某个或多个存储变量被用作初始值，且该初始值需在给定时间步长内所有迭代计算中保持不变（直至达到收敛状态），则不应在“时间步长结束（End of Timestep）”调用之前更新该值（详见第7.3.10节和第7.4.3.9节的补充说明）。



In much the same way that we took the TRNSYS kernel provided parameter and input values and set them to local variable names, we now have to set the calculated local variables to the appropriate TRNSYS global output array. The code to do so for Type201 is as follows:

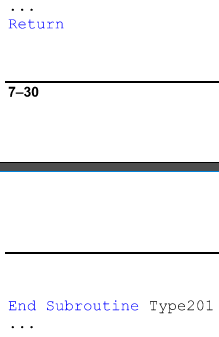
与我们此前将 TRNSYS 内核提供的参数和输入值赋给本地变量的方式类似，现在需要将计算得到的本地变量赋值给对应的 TRNSYS 全局输出数组（Global Output Array）。Type201 的实现代码如下所示：

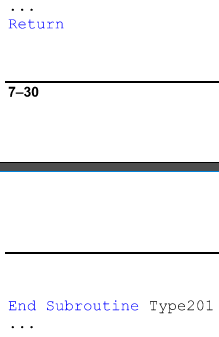




The very last requirement is that a Type return control to the TRNSYS kernel after its computations are complete.

最后一个要求是，在组件（Type）完成所有计算后，必须将控制权交还给 TRNSYS 内核（TRNSYS Kernel）。









Whenever two or more instances of a Type are used in a single simulation, some care must be taken to make sure that the local variables in a Type have the values that are associated with the current Unit. For parameters and inputs, this is not a problem; inputs are read at every iteration and parameters are reread when a new Unit is encountered (see section 0). Values that are calculated from parameters are also no problem; the calculations can be redone at each iteration. What if those calculations are time consuming? It would be nice to be able to do them once and store the result for each Unit (for each instance of the Type in the simulation) and then to retrieve the value instead of redoing the calculation. The concept of “static" storage is used. The Type programmer uses the setNumberStoredVariables() Access Function to reserve space in the global storage structure, and then uses the AccessFunctions setStaticArrayValue() and getStaticArrayValue() as appropriate. Refer to section 7.4.4.18.1.1 for additional information.

在单次仿真中，当一个组件类型（Type）被实例化为两个或更多单元（Unit）时，必须特别注意确保该组件（Type）中的本地变量所保存的是与当前单元（Unit）相对应的数值。对于参数和输入值而言，这通常不会构成问题；输入值会在每次迭代时重新读取，而参数则会在遇到新的单元时重新加载（详见第 0 节）。由参数计算得出的值一般也不是问题；这些计算可以在每次迭代中重复执行。但如果这些计算非常耗时呢？此时我们希望能够仅执行一次计算，并为每个单元（即仿真中该类型的每一个实例）存储结果，以便后续直接读取而无需重复计算。这就引入了“静态”（static）存储的概念。组件开发者通过调用 setNumberStoredVariables() 访问函数，在全局存储结构中预留空间，然后根据需要使用 setStaticArrayValue() 和 getStaticArrayValue() 这两个访问函数来设置和获取静态数组中的值。有关更多信息，请参阅第 7.4.4.18.1.1 节。

In another scenario, Types often need to compute a final value based on an initial value but they need that initial value to be the final value at the previous time step, not the final value at the previous iteration. Then at the end of the time step, the Type needs to take the final value calculated at the last iteration and put it into the “final value at the last time step" spot. This can be accomplished manually using two "static” storage spots or it can be accomplished more automatically using “dynamic" storage. In this scenario, the Type programmer again uses the setNumberStoredVariables() AccessFunction to reserve space in the global storage structure, at the end of each iteration, (s)he then uses the result of a call to the getDynamicArrayValueLastTimestep() function as the basis of calculations and then uses the function setDynamicArrayValueThislteration() to record the calculated result. At the end of the time step, the TRNSYS kernel takes care of moving the stored variables around so that the Type always gets the appropriate one. Refer to section 7.4.4.18.1.2 for additional information.

在另一种场景中，组件类型（Type）通常需要基于一个初始值来计算最终结果，但该初始值必须是上一时间步长的最终值，而非上一次迭代的最终值。因此，在时间步长结束时，组件需要将最后一次迭代计算出的结果保存为“上一时间步长的最终值”，以供下一个时间步长使用。这一功能可以通过手动方式实现，即使用两个“静态”（static）存储位置来分别保存当前值和上一时间步长的值；也可以通过更自动化的方式实现，即使用“动态”（dynamic）存储机制。在该场景下，组件开发者同样通过调用 setNumberStoredVariables() 访问函数在全局存储结构中预留空间。在每次迭代结束时，开发者使用对 getDynamicArrayValueLastTimestep() 函数的调用来获取上一时间步长的值作为计算依据，并使用 setDynamicArrayValueThisIteration() 函数记录本次迭代的计算结果。在时间步长结束时，TRNSYS 内核会自动处理这些存储变量的转移，确保组件始终获取到正确的值。有关更多信息，请参阅第 7.4.4.18.1.2 节。



实现时间步长间的数据存储：在 TRNSYS 中，组件（Type）通常需要在时间步长之间保留某些计算结果，以便用于后续的时间步长。这种机制对于模拟具有状态依赖性的系统（如热能存储、控制系统等）至关重要。实现这一功能的关键在于合理使用 静态存储（Static Storage） 和 动态存储（Dynamic Storage）。

With the release of TRNSYS 18 a user can elect to generate a simulation summary report when a simulation project is run. The simulation summary report (SSR) is an automatically created text file that contains a summary of certain pieces of information about the instances (Units) of Types contained in the simulation. Specifically what information is written to the SSR by a given Type is determined by the Type's author. The user has the ability to include all of a given Unit's information in the SSR or to exclude it all. This section will focus primarily on the source code that must be added to a Type in order for a Unit of that Type to be reported in the SSR.

随着 TRNSYS 18 的发布，用户在运行仿真项目时可以选择生成一份仿真总结报告（Simulation Summary Report, SSR）。该报告是一个自动生成的文本文件，其中包含关于仿真中所使用的组件类型（Type）实例（Unit）的某些信息摘要。具体来说，某一类型向 SSR 中写入哪些信息，是由该类型的开发者决定的。用户可以选择将某个单元的所有信息包含在 SSR 中，也可以选择完全不包含。本节将重点介绍为了使某一类型的单元能够在 SSR 中被报告，需要在该类型源代码中添加的相应代码部分。

The idea of the SSR is not to replace traditional, detailed output reporting such as is available using Type24. 25, and 46. The idea instead is to provide the user with a quick reference by which they can check to make sure that a particular component in their system is performing as expected, is sized correctly, or is operating in the intended mode.

SSR 的设计初衷并非取代传统的、详细的输出报告功能，例如通过 Type24、Type25 和 Type46 所提供的那种详尽的输出功能。相反，它的目的在于为用户提供一个快速参考工具，使用户能够借此检查系统中某个特定组件是否按预期运行、尺寸选择是否合理，或者是否工作在预期的运行模式下。

Generally a Type can provide four kinds of information to the SSR. These are: numerical values, integrated values, minimum / maximum values, and text strings. An example of a numerical value might be the capacity of a heater or the area of a photovoltaic array. These are single values that the programmer thinks might be a useful way to summarize the characteristics of a particular component. They may, but certainly do not have to be, the values of one or more of a Type's parameters. An example of an integrated value might be the power consumed by a pump or the energy delivered to a flow stream by a heat pump. Integration is automatically performed over the entire length of the simulation. An example of a minimum/maximum value might be the efficiency of a fan motor or the COP of a chiller. Minimums and maximums are reported over the entire length of a simulation. A text string may be used to provide a reference for which control mode or which algorithm has been selected in a Type that offers more than one.

通常情况下，一个组件类型（Type）可以向仿真总结报告（SSR）提供四类信息：数值型数据、积分值、最小/最大值 以及 文本字符串。具体如下：

1. 数值型数据（Numerical Values）：

这类信息用于表示组件的某些固定特性或配置参数。例如加热器的容量（Capacity）或光伏阵列的面积（Area）。这些是由开发者认为有助于概括该组件特性的单一数值。这些数值可以是组件参数中的一个或多个，但并非必须如此。

2. 积分值（Integrated Values）：

积分值是指在整个仿真过程中对某一变量进行时间积分后的结果。例如水泵的总耗电量，或热泵传递给流体的能量。TRNSYS 会自动在整个仿真周期内对相关变量进行积分。

3. 最小/最大值（Minimum / Maximum Values）“”

此类信息用于报告某个变量在整个仿真过程中的最小值和最大值。例如风机电机的效率或冷水机组的性能系数 COP（Coefficient of Performance）。

4. 文本字符串（Text Strings）：

文本字符串用于描述组件中某些非数值型的设置选项，例如所选的控制模式或运行算法。这对于具有多种运行模式的组件尤其有用，可以帮助用户快速识别当前使用的是哪一种配置。

There are three sections of code that must be added to a Type in order to allow a user to include a particular instance (Unit) of a Type in the SSR file produced at the end of a simulation. These are the reservation section, the initialization section and the update section.

为了使用户能够在仿真结束时生成的 SSR 文件中包含某一组件类型（Type）的特定实例（Unit），需要在该类型的源代码中添加三个必要的代码部分：预留部分（Reservation Section）、初始化部分（Initialization Section） 和 更新部分（Update Section）。



If its information is to be included in an SSR report, a Type must first reserve the number of numerical value spots, min max value spots, integrated value spots, and text strings it wishes to use. This is accomplished by calling the subroutine setNumberOfReportVariables( ) during Type initialization (covered in section 7.3.11). The subroutine requires four integer arguments each of which corresponding to the number of values of each information type being reserved. The code required to reserve one integrated value spot, two min/max value spots, three numerical value spots and four text string spots is:

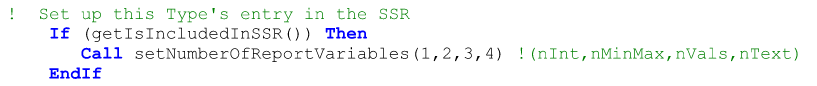
如果某一组件类型（Type）的信息需要包含在 SSR 报告中，则该组件必须首先预留用于存储以下四类信息的空间数量：数值型数据（Numerical Values）、最小/最大值型数据（Min/Max Values）、积分值型数据（Integrated Values）、文本字符串（Text Strings）。

这一功能通过在组件初始化阶段调用子程序 setNumberOfReportVariables() 来实现（详见第 7.3.11 节）。该子程序需要传入四个整数参数，分别对应上述四种信息类型的预留数量。



It should be noted that unlike many of the other steps in Type initialization, reserving space for report variables is optional and is best done only if the user implementing the Type in a simulation has actually asked for a particular Unit number to be reported. Therefore, the call to setNumberOfReportVariables( ) should only be executed if the current unit number has been indicated for reporting. The Type programmer can made a call to the getIsIncludedInSSR( ) function and call the setNumberOfReportVariables( ) subroutine only on a .true. result. The complete code for reserving space in the SSR might look like the following:

要指出的是，与组件（Type）初始化过程中的许多其他步骤不同，预留报告变量空间的操作是可选的，并且只有当在仿真中实际要求某个特定单元（Unit）生成报告时才应执行此操作。因此，setNumberOfReportVariables() 子程序的调用应仅在当前单元号（current unit number）被指定为需报告的情况下才执行。组件开发者可以通过调用 getIsIncludedInSSR() 函数来判断当前单元是否需要被包含在 SSR 报告中，并且仅在返回值为 .TRUE. 时才调用 setNumberOfReportVariables()。



Returning to the example of Type201, let us say that we wish to include the heater's energy consumption and energy delivered over the course of the simulation (i.e. two integrated values), the minimum and maximum heater temperature set point (one min/max value), the heater's capacity, overall loss coefficient, and efficiency (three numerical values). The code to reserve space would look like the following:

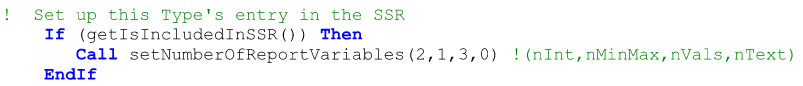
回到 Type201 的示例，假设我们希望在 SSR 报告中包含以下信息：

加热器在整个仿真过程中的耗电量和供热量（即两个积分值）

加热器温度设定点的最小值和最大值（一个最小/最大值）

加热器的容量、总损失系数和效率（三个数值型数据）

为了预留这些报告变量所需的空间，我们需要调用 setNumberOfReportVariables() 子程序，并传入相应的数量参数。

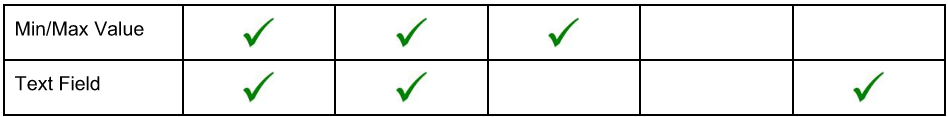




Once space has been reserved in the SSR data structures each spot must be initialized. During initialization, the Type programmer provides a heading for (title/description of) each data value, provides the units of each data value and can provide the values for numerical value fields and text value fields if desired. The following table indicates what information should be provided during initialization for each of the four data types:

一旦在 SSR 数据结构中预留了空间，就必须对每个位置进行初始化。在初始化过程中，组件（Type）开发者需要为每个数据值提供一个标题（即名称/描述），提供每个数据值的单位，并可根据需要为数值型字段和文本型字段提供初始值。下表列出了针对四种数据类型在初始化阶段应提供的信息：





The general code for initializing a numerical value might look something like the following:



The general code for initializing an integrated value might look something like the following:



The general code for initializing a min/max value might look something like the following:



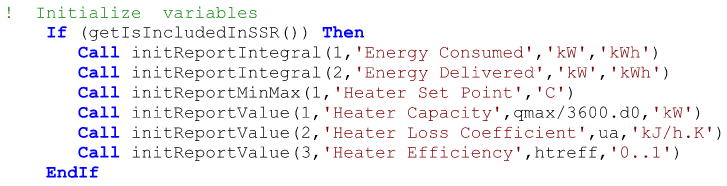
The general code for initializing a text field might look something like the following:



As when reserving space in the SSR data structures, initialization should only be carried out if a particular Unit has been tagged by the user as being included in the SSR. it should also be noted that text and numerical value fields do not have to be initialized during the simulation start time step. It may be that the values the programmer wishes to report are not known until later in the simulation.

与在 SSR 数据结构中预留空间时一样，初始化操作也应仅在用户已将某个特定单元（Unit）标记为包含在 SSR 中的情况下执行。同时需要注意的是，文本型和数值型字段并不必须在仿真的起始时间步长中进行初始化。有些情况下，开发者希望报告的值可能直到仿真的后续阶段才能确定。

The code for Type201's SSR value initialization is,



Note that if one of the SSR variables is initialized twice, TRNSYS will generate a fatal error.

The four functions referenced in this section are further detailed in sections 7.4.2.89 through 7.4.2.92

请注意，如果某个 SSR 变量被初始化两次，TRNSYS 将会生成一个致命错误。

本节中提到的四个函数在第 7.4.2.89 至 7.4.2.92 节中有更详细的说明。



Two of the four SSR variable types must be updated throughout the simulation: integrated variables and min/max variables. Typically the update will be performed at the end of a time step. The update is not automatic and must be initiated by the Type programmer. During the update, the values of the local variables that are being reported must be retrieved and then sent as an argument to the appropriate update function. If the variables being updated and reported are inputs or outputs of the Type, the retrieval consists of using the getInputValue()and / or getOutputValue( ) functions. If, however, the variable being reported is not an input or an output then either the value must be recalculated at the end of time step or the programmer must place the value into the between time step storage structures discussed in section 7.3.16.

在四种 SSR 变量类型中，有两种必须在整个仿真过程中持续更新：积分型变量（integrated variables） 和 最小/最大值型变量（min/max variables）。通常情况下，这种更新操作是在时间步长结束时进行的。更新操作并非自动完成，必须由组件（Type）开发者主动发起。在执行更新时，开发者必须先获取正在报告的本地变量的当前值，并将该值作为参数传递给相应的更新函数。如果被更新和报告的变量是组件的输入或输出变量，则可以通过调用 getInputValue() 和 / 或 getOutputValue() 函数来获取这些值。然而，如果所报告的变量并不是输入或输出变量，则开发者要么需要在时间步长结束时重新计算该值，要么必须将该值存储到第 7.3.16 节中讨论的“时间步长间存储结构”中。

The general code for updating an integrated value might look something like the following:



The general code for initializing a min/max value might look something like the following:



In which index is an integer and in which 'localVariable’ is a double precision variable containing the present value of the variable being updated.

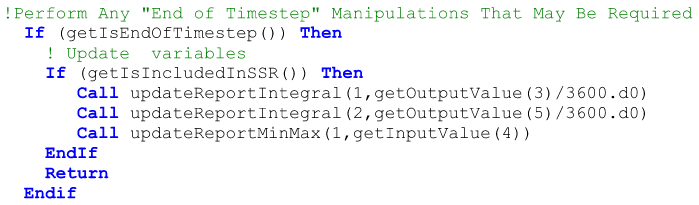
The two functions referenced in this section are further detailed in sections 7.4.2.106 and 7.4.2.107.

The code for Type201's SSR value update is:

其中，index 是一个整数，而 localVariable 是一个双精度（double precision）变量，包含当前正在更新的变量的当前值。

本节中提到的两个函数在第 7.4.2.106 和 7.4.2.107 节中有更详细的说明。

Type201 的 SSR 值更新代码如下：



Note that if one of the SSR variables is updated twice in the same time step, TRNSYS will generate a fatal error.



There are two methods of including a new Type in an external DLL. For information on why a Type should be contained in an external DLL, please refer to the introduction to this manual (section 7.1).

The first method by which a Type may be compiled and linked into an external DLL is to use the TypeStudio. The Type Studio provides a compiler that is integrated into the TRNSYS package and which is set up to create a DLL compatible with TRNSYS. Please refer to sections 7.2.2, 7.2.3, and 7.2.4 for information about the Type Studio and its use.

The second possibility is to use an external FORTRAN compiler to compile Types and link them into a DLL. The advantage of this method is that many Fortran compilers offer extensive debugging features that (among other things) will allow the user to step one line at a time through their Type code viewing the values of internal variables while the simulation is running. The disadvantage of this method is that third party FORTRAN compilers can be expensive and it is unfortunately quite difficult to get the compiler settings correct such that the compiled DLL can communicate with the TRNSYS DLLs. Section 7.6 covers the information needed in order to compile Types and create DLLs using the Intel Visual Fortran compiler.

将一个新的组件类型（Type）包含在外部 DLL 中有两种方法。有关为何应将 Type 放置在外部 DLL 中的信息，请参阅本手册的引言部分（第 7.1 节）。

第一种将 Type 编译并链接到外部 DLL 的方法是使用 TypeStudio。TypeStudio 提供了一个集成在 TRNSYS 软件包中的编译器，该编译器被配置为生成与 TRNSYS 兼容的 DLL。有关 TypeStudio 及其使用的更多信息，请参见第 7.2.2、7.2.3 和 7.2.4 节。

第二种方法是使用外部 FORTRAN 编译器来编译组件类型，并将其链接到一个 DLL 中。这种方法的优势在于，许多 Fortran 编译器提供了强大的调试功能，允许用户在仿真运行时逐行执行 Type 的代码，并查看内部变量的值。然而，这种方法的缺点是第三方 Fortran 编译器可能价格昂贵，而且不幸的是，要正确设置编译器选项以使生成的 DLL 能够与 TRNSYS 的 DLL 正确通信也相当困难。第 7.6 节介绍了如何使用 Intel Visual Fortran 编译器来编译 Type 并生成 DLL 所需的相关信息。



This section gives usage rules for most of the functions and subroutines in the TRNSYS kernel that Types may need to access in the course of a simulation.

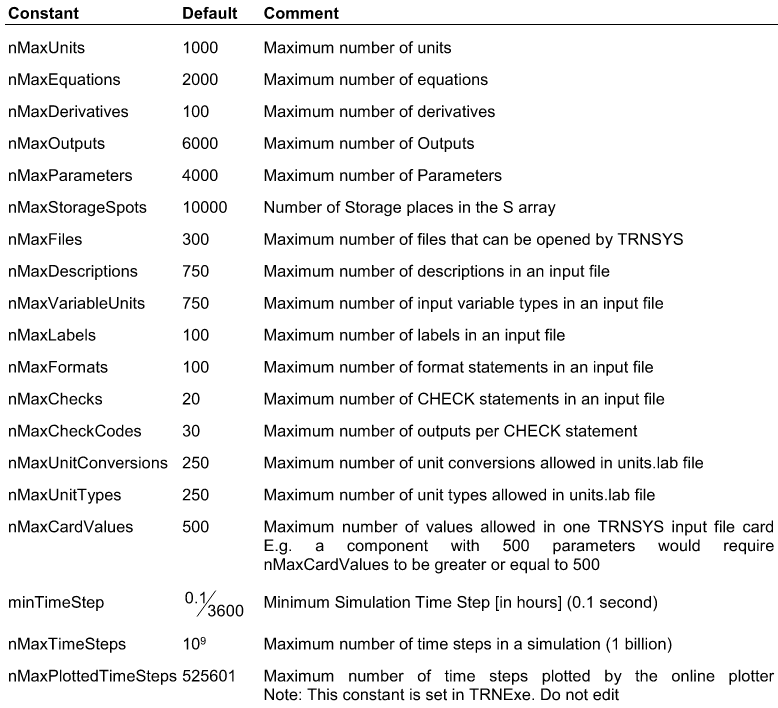
本节给出了组件类型（Type）在仿真过程中可能需要访问的 TRNSYS 内核中大多数函数和子程序的使用规则。

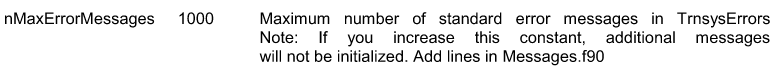
 全局常量

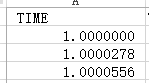
Certain constants in the TRNSYS kernel are defined globally in such a way that they are easy to modify.

TRNSYS内核中的某些常量是以全局方式定义的，以便于修改。



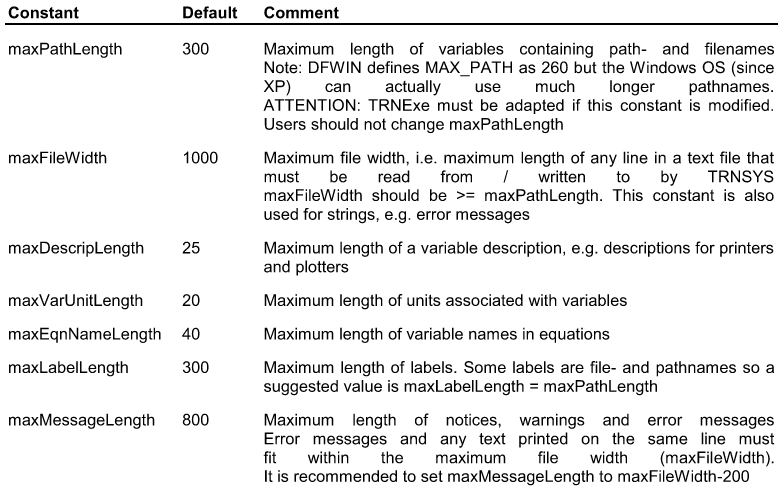




minTimeStep: 0.1/3600  0.1/3600=0.00002778

该部分的内容与TrnsysConstants.f90的内容有不一样







This section lists all functions declared in the "TrnsysFunctions" module. Note that the subroutines used to handle data storage are mentioned here for the record, but detailed explanations on their use can be found in section 7.4.4.18.

本节列出了在 "TrnsysFunctions" 模块中声明的所有函数。请注意，用于处理数据存储的子程序在此仅为记录目的而提及，其使用方法的详细说明可在第 7.4.4.18 节中找到。

Please note that the first group of access functions (getMaxDescripLength(), etc.) provide access to global constants. Those constants are declared in the TrnsysConstants module, so Fortran-written Type scan access them more easily through a "use" statement: use TrnsysConstants. Those access functions are provided for non-Fortran Types.

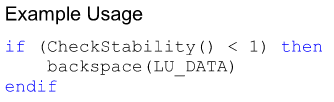
请注意，第一组访问函数（如 getMaxDescripLength() 等）用于访问全局常量。这些常量是在 TrnsysConstants 模块中声明的，因此使用 Fortran 编写的组件类型可以通过 use TrnsysConstants 语句更方便地访问它们。这些访问函数主要是为非 Fortran 编写的组件类型提供的。



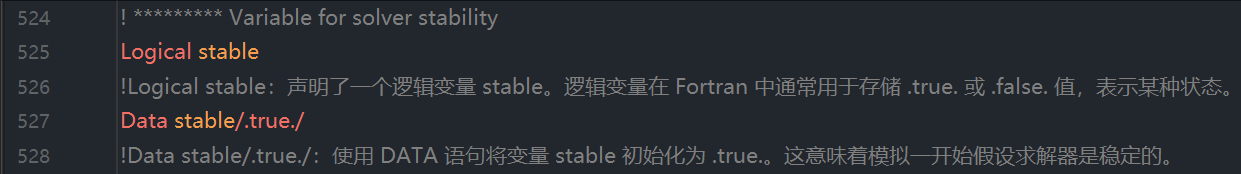


An integer function that returns a 1 if the last time step converged and a 0 if the last time step did not converge. The Check Stability function is used in SOLVER 1, which tries more than one control strategy and then backs up a time step in order to try something else if it did not find a stable solution. TRNSYS data reading components need to know that they should not continue reading the data file but should back up as well.

这是一个整数函数，如果上一时间步长收敛，则返回 1；如果上一时间步长未收敛，则返回 0。该“检查稳定性”（CheckStability）函数被用于 SOLVER 1 中，该求解器会尝试多种控制策略，如果未找到稳定解，则会回退一个时间步长以尝试其他方法。此时，TRNSYS 中的数据读取组件需要知道它们也应当回退，而不是继续从数据文件中读取数据。



TrnsysData.f90文档中定义了stable逻辑变量



TrnsysFunctions.f90文档中的Function CheckStability()

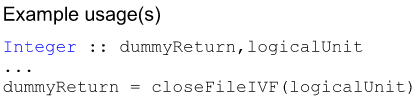


疑问：通过什么方式来修改stable的值



An integer function that closes the file associated with a specified logical unit number. The function always returns a value of 1. This function is typically used by Types that are located in an external DLL(i.e. not in the TRNDll) but which need to read an external file that has been associated with the Unit by means of an ASSlGN statement. With certain Fortran compilers, since the file has already been opened by the TRNDll, the external DLL is prevented from also opening it. This function gives the user the ability to tell the TRNSYS kernel to close the file. The calling Type would then open the file itself and read the data. NOTE: files that are intended to be read (and interpolated) by the InterpolateData (formerly Dynamic Data) routine do not need to make use of this function since it is a TRNSYS kernel routine that will be performing the file read.

这是一个返回整数值的函数，用于关闭与指定逻辑单元号相关联的文件，该函数始终返回值 1。此函数通常被那些位于外部 DLL（即不在 TRNDll 中）中的组件类型（Type）使用，这些组件需要读取一个通过 ASSIGN 语句与该单元（Unit）关联的外部文件。在某些 Fortran 编译器环境下，由于该文件已被 TRNDll 打开，导致外部 DLL 无法打开它。此时，该函数允许用户通知 TRNSYS 内核关闭该文件。调用该函数的组件随后可以自行打开该文件并读取数据。注意：对于那些打算由 InterpolateData（以前称为 Dynamic Data）例程（routine）进行读取（和插值）的文件，无需使用此函数，因为该例程属于TRNSYS 内核的一部分，将由内核自动完成文件的读取操作。

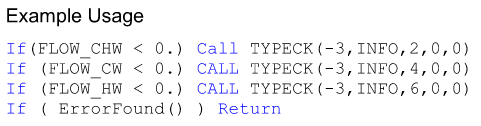






Logical function that returns a value of "TRUE" if TRNSYS errors have been found. This function is most useful in Types that call one or more of the TRNSYS Utility subroutines (such as TYPECK, PSYCHROMETRICS, or STEAM) which may call find and flag errors. For example, say a user written Type calls TYPECK to indicate that one of the lNPUTS had an inappropriate value. TYPECK would call the TRNSYS utility subroutine MESSAGES, which would print out the error, would log that an error occurred and would return control to TYPECK, TYPECK would in turn return control to the Type, which can then avoid the remainder of its calculations by accessing the ErrorFound function. An example follows in which calculations cease if an illegal value of the input variables FLOW CHW, FLOW CW, or FLOW HW is found.

这是一个逻辑函数，如果 TRNSYS 中发现了错误，该函数将返回值为 "TRUE"。此函数在调用了TRNSYS 实用子程序（如 TYPECK、PSYCHROMETRICS 或 STEAM）的组件类型（Type）中特别有用，这些子程序可能会检测并标记错误。例如，假设用户编写的某个 Type 调用了 TYPECK，用于指出其中一个输入值不合法，TYPECK会进一步调用TRNSYS 实用子程序 MESSAGES，后者将打印出错信息、记录错误发生，并将控制权返回给 TYPECK。随后 TYPECK 将控制权交还给调用它的组件类型，此时该组件可以通过调用ErrorFound 函数来判断是否发生了错误，从而跳过后续的计算步骤。以下是一个示例：当输入变量FLOW CHW、FLOW CW或FLOW HW的值非法时，计算过程将立即终止。



example示例包含两个调用的函数：TYPECK（暂未找到出处）和ErrorFound

TrnsysData.f90文档中定义了ierror逻辑变量



TrnsysFunctions.f90文档中的Function ErrorFound()



疑问：通过什么方式来修改ierror的值

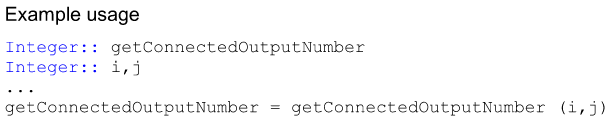


An integer function that returns the output number connected to the jth input of Unit i. When using the Powell's Method solver (SOLVER 1), you should call the getConnectedOutputNumberS1() function instead of this one.

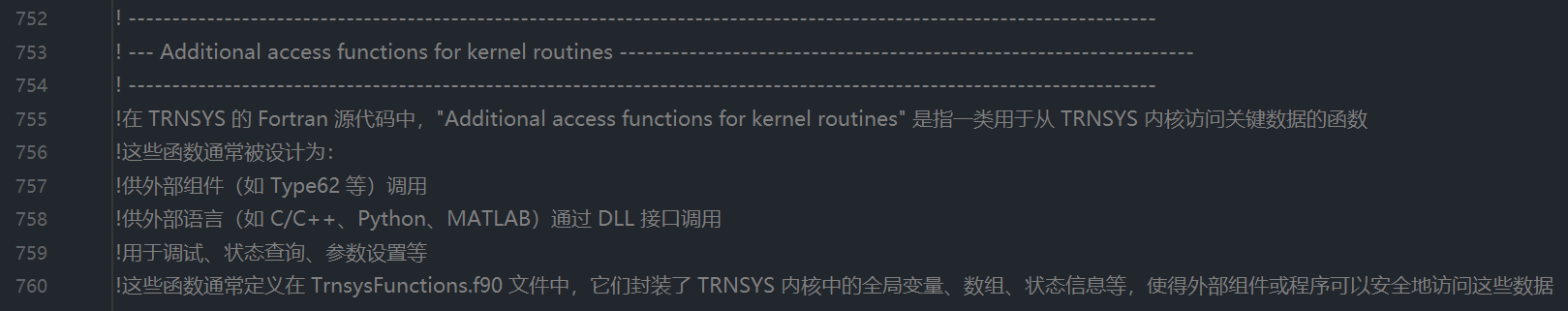
这是一个整数函数，用于返回与第i个单元（Unit）的第j个输入端口相连接的输出端口号。在使用 Powell's Method 求解器（SOLVER 1） 时，应调用 getConnectedOutputNumberS1()函数来代替该函数。

getConnectedOutputNumber(i, j)功能实现，利用了TrnsysData.f90中的inp()、xin()、incon()。

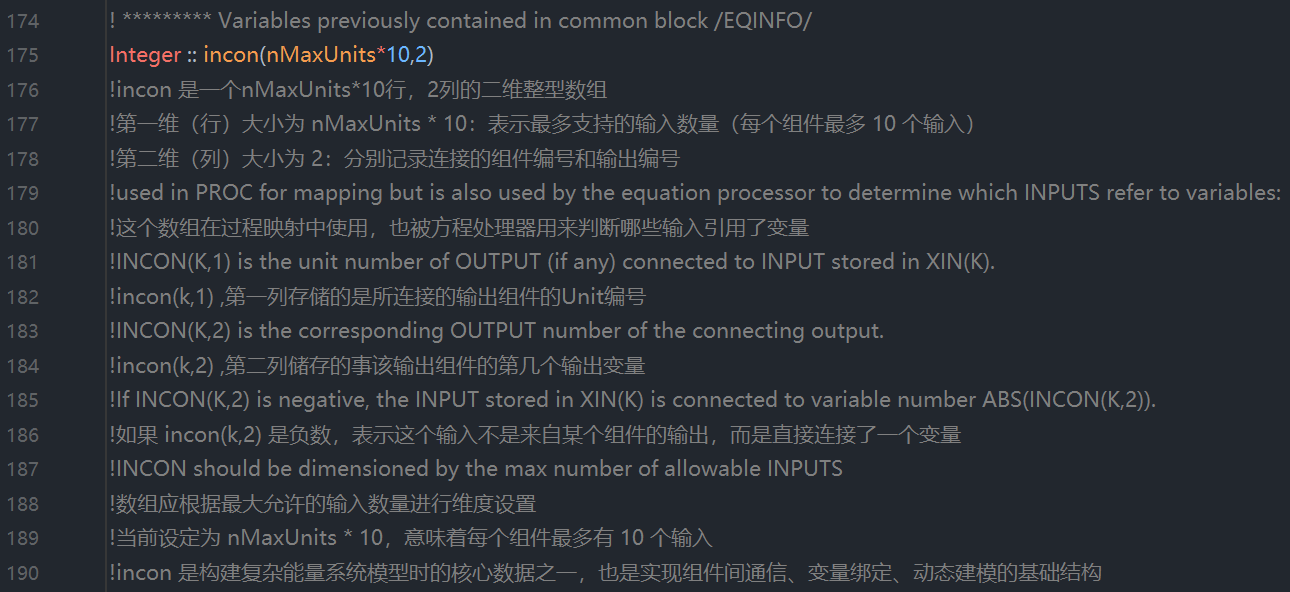
**当Successive Substitution solver (SOLVER 0)时，采用该function**



function getConnectedOutputNumber(i, j)属于additional access functions for kernel routines.



在getConnectedOutputNumber(i, j)程序中使用了incon（TrnsysData.f90），以下是incon的解释：



例1：假设有3个Units，1号Unit有3个输出项，2号Unit有4个输出项，3号Unit有3个输出项，那么incon的数组结构为：

01（1,1），第一个1代表Unit1，第二1代表Unit1的第一个输出

02（1,2），第一个1代表Unit1，第二2代表Unit1的第二个输出

03（1,3），第一个1代表Unit1，第二3代表Unit1的第三个输出

04（2,1），第一个2代表Unit2，第二1代表Unit1的第一个输出

05（2,2），第一个2代表Unit2，第二2代表Unit1的第二个输出

06（2,3），第一个2代表Unit2，第二3代表Unit1的第三个输出

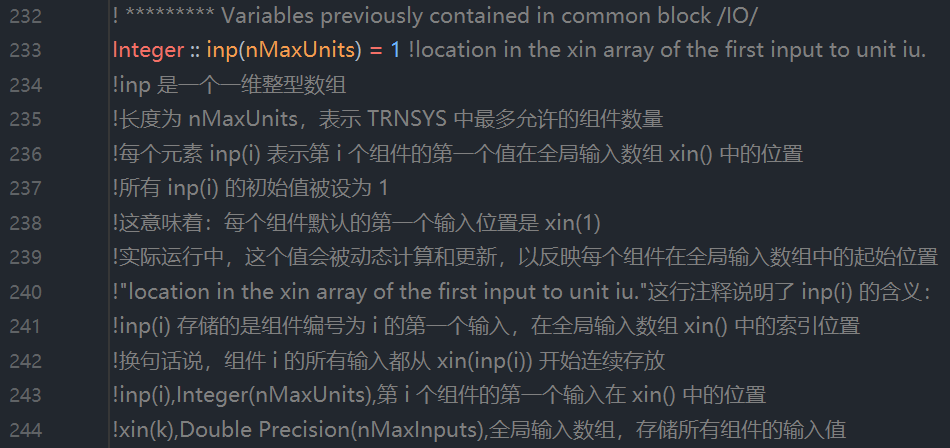
07（2,4），第一个2代表Unit2，第二4代表Unit1的第四个输出

08（3,1），第一个3代表Unit3，第二1代表Unit1的第一个输出

09（3,2），第一个3代表Unit3，第二2代表Unit1的第二个输出

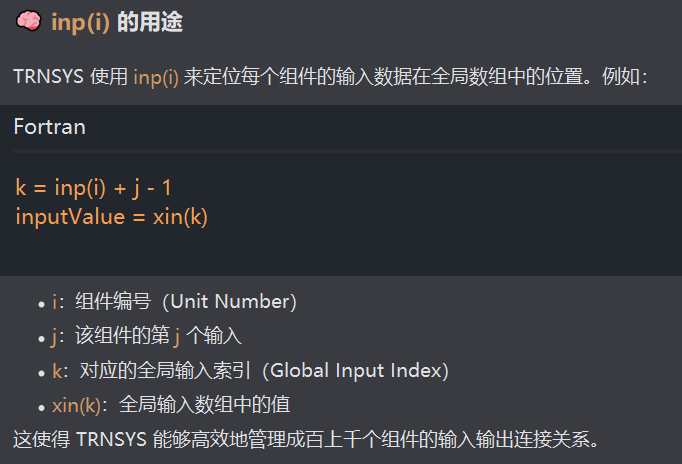
10（3,3），第一个3代表Unit3，第二1代表Unit1的第三个输出

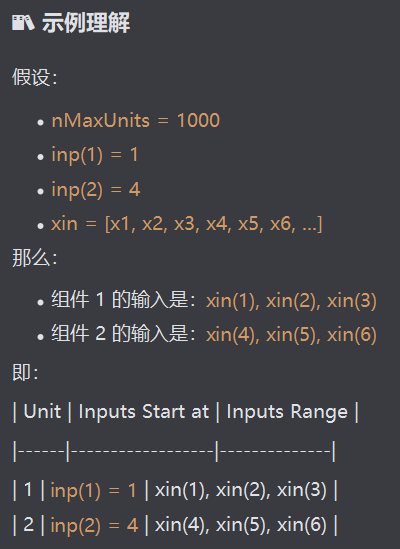
在getConnectedOutputNumber(i, j)程序中使用了inp（TrnsysData.f90），以下是inp的解释：



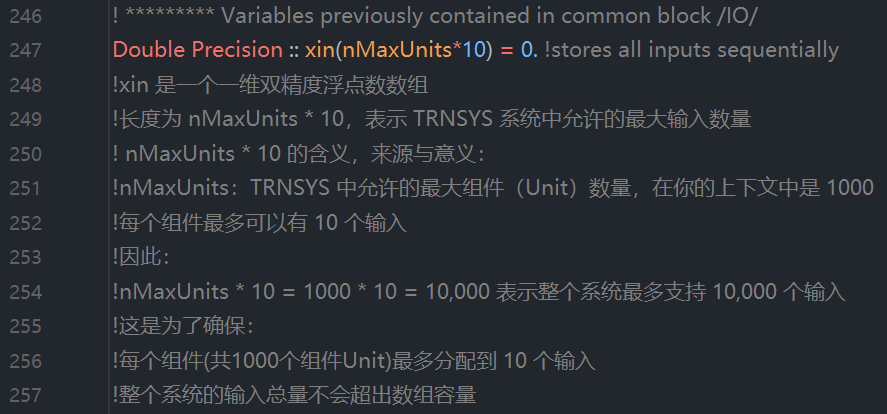
示例：







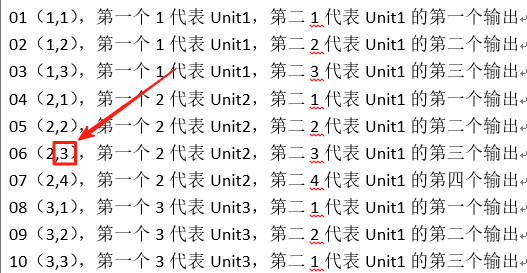
对于inp()的理解还需要配合xin()一起理解，以下是xin()解释



以下是function getConnectedOutputNumber(i, j)的程序及说明：



参考例1的数据，如果getConnectedOutputNumber(2,3)，那么k=inp(2)-1+3，其中，inp(2)=4，k=4-1+3=6，getConnectedOutputNumber=incon(6,2)= [06（2,3），第一个2代表Unit2，第二3代表Unit1的第三个输出]对应的值。存储位置如红色箭头所指的位置。



（使用 Powell's Method 求解器（SOLVER 1），略）

An integer function that returns the output number connected to the jth input of Unit i. When using the Successive Substitution solver (SOLVER 0), you should call the getConnectedOutputNumber() function instead of this one.

一个返回整数值的函数，用于获取与单元 i 的第 j 个输入端口相连的输出端口号。当使用逐次替代求解器（SOLVER 0）时，应调用 getConnectedOutputNumber() 函数来代替该函数。

以下内容来自06-TRNEdit的6.3.15（开始）



TRNSYS is outfitted with two methods for solving the coupled system of algebraic and differential equations that model a given system: the "successive substitution" method and "Powel's” method.

TRNSYS 配备了两种求解用于描述特定系统的代数与微分方程耦合系统的方法：“逐次替代法（Successive Substitution）” 和 “鲍威尔Powell 方法（Powell's Method）”。

A SOLVER command has been added to TRNSYS to select the computational scheme. The optional SOLVER card allows the user to select one of two algorithms built into TRNSYS to numerically solve the system of algebraic and differential equations. The format of the SOLVER card is

在 TRNSYS 中，新增了一条 SOLVER 命令用于选择计算方案。该可选的 SOLVER 卡允许用户从 TRNSYS 内置的两种算法中选择一种，以数值方式求解代数和微分方程组。SOLVER 卡的格式如下：



where k is either the integer 0 or 1. If a SOLVER card is not present in the TRNSYS input file, SOLVER 0 is assumed. If k = 0, the SOLVER statement takes two additional parameters, RFmin and RFmax (see section ):

其中 k 是整数 0 或 1。如果 TRNSYS 输入文件中没有 SOLVER 卡，则默认使用 SOLVER 0。若 k = 0，则 SOLVER 语句还需要两个附加参数：RFmin 和 RFmax（参见相关章节）：



The two solution algorithms are:

● 0: Successive Substitution

● 1: Powell's Method (Powell, 1970a and 1070b)

Descriptions of these algorithms can be found in the references and most other numerical methods books. They are briefly described in the next sections.

**逐次替代法（可以查看高斯赛德尔迭代法）**

With successive substitution, the outputs of a given model are substituted for the inputs of the next model in the system. The performance of that next model is recomputed and its outputs are then substituted for the inputs of the next model. This substitution continues at a given time step until all connected outputs have stopped changing (i.e. their change is smaller than the limits fixed by the TOLERANCES statement). At that point the TRNSYS kernel deems that convergence has been reached and proceeds on to simulate the next time step.

在使用逐次替代法（Successive Substitution）时，一个给定模型的输出会被作为系统中下一个模型的输入进行。接下来的模型会重新计算其性能表现，并将其输出再替代为下一个模型的输入。这一替代过程在当前时间步长内持续进行，直到所有相连的输出值不再发生变化（即它们的变化量小于由 TOLERANCES 语句设定的限值）。此时，TRNSYS 内核判定已达到收敛，并继续进行下一时间步长的仿真。（在一个时间步，上一个模型的输出作为下一个模型的输入，当所有相连接模型的输出值都满足收敛判定条件，则认为该时间步的计算完成；然后进行下一个时间步的计算）

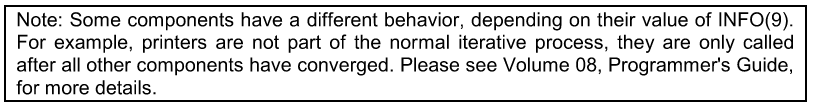
说明：逐次替代法（Successive Substitution）是TRNSYS中默认使用的求解方法（SOLVER 0），适用于大多数线性或弱非线性的系统。在每个时间步长内，TRNSYS 会按照组件之间的连接顺序依次计算每个组件的输出，并将这些输出作为后续组件的输入反复迭代（迭代机制是什么？）。当迭代过程中所有关键变量的变化都小于预设的容差（TOLERANCES）时，认为系统在该时间步长内已经“收敛”，仿真进入下一个时间步长。如果在规定次数内无法达到收敛，则可能会触发错误或导致求解器回退时间步长并尝试其他策略（例如使用 Powell's Method）。

此段内容可参考《TRNSYS 程序员指南》（Programmer’s Guide）第7.5节关于求解器的详细说明，以及《TRNEdit用户手册》中对 TOLERANCES 命令的介绍。

**疑问：每个时间步内的计算机制是什么？**

Generally speaking, TRNSYS calls all components at least once per time step. Then, the TRNSYS solver keeps track of which components must be called during the same time step: only components for which at least one input has changed beyond the fixed tolerances are called. When all components have converged, or when the maximum number of iterations has been reached (see the LIMITS statement), TRNSYS continues to the next time step.

一般来说，TRNSYS在每个时间步长内至少会调用所有组件一次。随后，TRNSYS的求解器会跟踪哪些组件需要在同一个时间步长内再次调用：只有当某个组件的至少一个输入值的变化超过了预设的容差限值时，该组件才会被重新调用。当所有组件都达到收敛，或者达到了最大迭代次数（参见 LIMITS 语句）时，TRNSYS 将继续进入下一个时间步长的仿真。



Although relatively simple, the successive substitution computational scheme used in most TRNSYS simulations has proven to be reliable and efficient for simulating systems with energy storage such as solar domestic hot water systems, buildings, and HVAC systems. These systems typically have less than 50 coupled differential equations and 100 simultaneous nearly-linear algebraic equations with few recyclic loops and controller decisions. The limitations of the computational scheme become apparent when TRNSYS is used to solve sets of non-linear algebraic equations without differential equations. Equations of this type occur in systems for which the energy storage is negligible, such as for a photovoltaic array directly coupled to a load or a refrigeration system operating at steady-state conditions. The successive substitution solution method does not efficiently solve non-linear algebraic equations and may, in fact, not be able to find a solution if the equations are highly non-linear. If the algebraic loops that cause the numerical problems are clearly identified, the ACCELERATE statement can be used to solve the problem. In other cases, adding numerical relaxation to solver 0 can improve its robustness and speed, depending on the type of numerical problems which is involved. Both methods are described here below.

尽管相对简单，TRNSYS 大多数仿真中所使用的逐次替代计算方案已被证明在模拟带有蓄能装置的系统（如太阳能生活热水系统、建筑系统和暖通空调系统）时是可靠且高效的。这类系统通常包含不到50个耦合的微分方程和100个同时求解的近线性代数方程，并且仅有少量反馈回路（recyclic loops） 和控制决策（controller decisions）。然而，当TRNSYS被用于求解不含微分方程的非线性代数方程组时，该计算方案的局限性就变得明显了。此类方程常见于蓄能可以忽略的系统中，例如光伏阵列直接连接负载的情况，或处于稳态工况下的制冷系统。逐次替代法在求解这类非线性代数方程时效率较低，实际上，如果方程高度非线性，甚至可能无法找到解。如果导致数值问题的代数回路能够被明确识别，则可以使用 ACCELERATE语句来解决该问题。在其他情况下，向求解器 0 添加数值松弛（numerical relaxation）也可以根据具体涉及的数值问题类型来提高其鲁棒性和速度（提高求解器的鲁棒性）。这两种方法将在下文中分别进行介绍。



An ACCELERATE command was added to TRNSYS version 13 to improve convergence in problems with recyclic information flow. The ACCELERATE (see Section 6.3.10) command allows the user to break a selected INPUT-OUTPUT connection and replace it with a single-variable Newton's method solution algorithm. Although useful in many circumstances, the ACCELERATE command may be unsatisfactory for two reasons; 1) it requires the user to identify the appropriate INPUT-OUTPUT connection and 2) it implements a single-variable solution method when in many situations, a multiple-variable method is required. Another way in which numerical convergence problems have been handled in past versions is by the user coding convergence-enhancing techniques within the component models. For example, combined-component models have been developed for TRNSYS wherein the non-linear equations describing two or more pieces of equipment are solved internally in a single component model. Although combined-component’ models may eliminate numerical problems, they reduce component modularity and require the user to implement solution techniques, thereby defeating the original purpose of TRNSYS.

TRNSYS13版本中新增了一条ACCELERATE命令，用于改善存在反馈信息流（recyclic information flow）问题时的收敛性。该ACCELERATE命令（详见第 6.3.10 节）允许用户断开某个选定的输入-输出连接，并将其替换为一个基于单变量牛顿法（single-variable Newton's method）的求解算法。尽管在许多情况下该命令非常有用，但它也可能存在以下两个方面的不足：

1）需要用户手动识别：用户必须自行判断并指定需要断开的输入-输出连接；

2）方法局限性：它在很多情况进行单变量求解方法，但是，往往需要多变量求解技术来更有效地解决问题。

在过去版本中，处理数值收敛问题的另一种方式是：用户在组件模型内部编写增强收敛性的技术。例如，在 TRNSYS 中开发了“组合组件”（combined-component）模型，其中将描述两个或多个设备的非线性方程在一个组件模型内部统一求解。虽然这种方法可以有效消除数值问题，但它降低了组件之间的模块化特性，并要求用户自行实现求解策略，从而违背了 TRNSYS的最初设计初衷 —— 即通过通用求解器自动处理组件间的耦合关系，而非依赖于组件开发者手动干预。

数值松弛因子

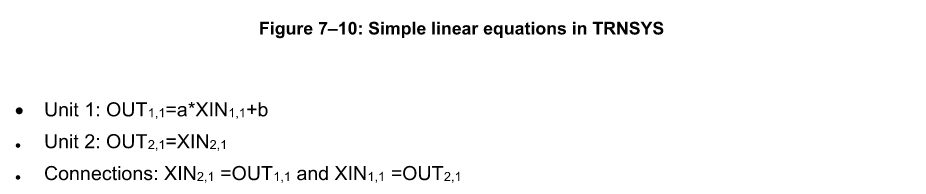
Numerical relaxation has been proven to significantly improve the performance of TRNSYS solver 0 for some classes of problems. The coupling of airflow and temperatures in a building are a typical example of problems that cannot be solved easily using successive substitution, and the implementation of numerical relaxation in the TRNSYS kernel was decided after its successful application in TRNFLOW (Weber et al.2003).

数值松弛（Numerical Relaxation）已被证明可以显著提升TRNSYS求解器 0（Solver 0，即逐次替代法）在某些类型问题中的性能。建筑中空气流动与温度之间的耦合问题就是典型的例子：这类问题使用传统的逐次替代法难以求解。因此，在TRNFLOW中成功应用数值松弛技术后（Weber 等，2003），TRNSYS内核决定引入该技术。

A simple example can be used to illustrate the purpose of numerical relaxation. Consider the following system of equations



Solver 0 will only solve this system for -1 < a < 1. In a TRNSYS simulation, this system of equations will occur if 2 Units are connected in a recyclic information loop:



将Figure 7-10转换为模块图，如下：

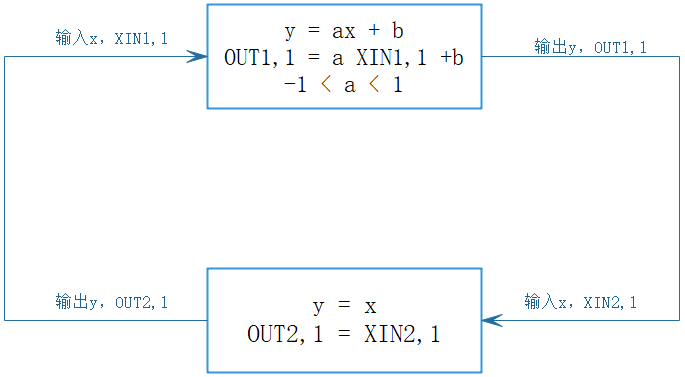
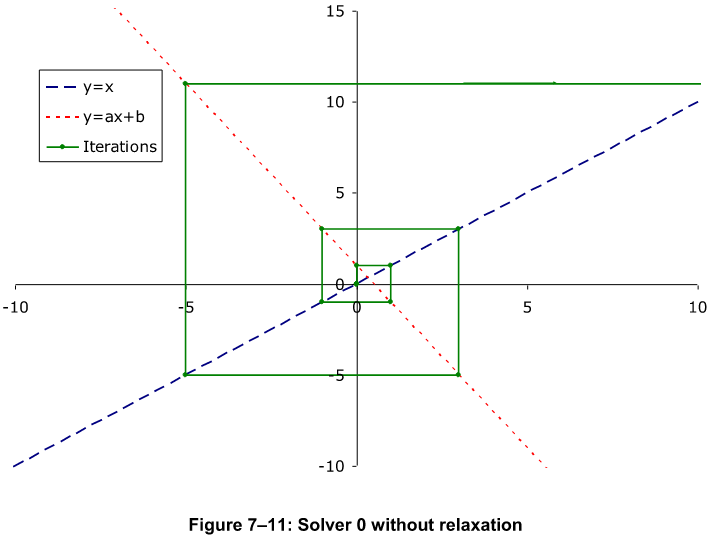


Figure 7-11 shows how Solver 0 would attempt to solve such problems with a = -2 and b = 1. Starting with a guess value of 0, the successive values of the output of Unit 1 are: 1, -1. 3, -5, 11. -21, etc. The solution(x=y=1/3) will never be reached.

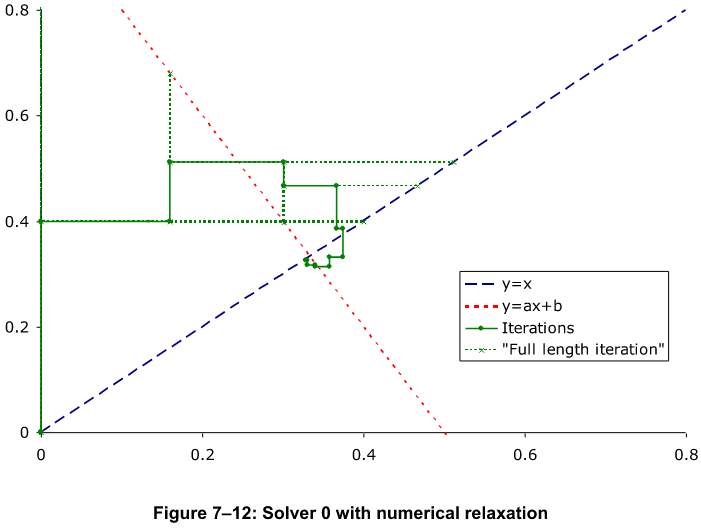


The relaxation method consists in adding some damping to the output values to restrict their change. （松弛方法的核心思想是在输出值的变化过程中引入一定的阻尼作用，以限制其变化幅度。）The general formulation is:



Where RF is the relaxation factor (0 s RF ≤ 1). If RF=1, There is no relaxation (Solver 0). If RF=0, there is no solver at all, OUT is kept constant.（松弛因子的设置）

If we take the same example (a=-2, b=1), the successive substitution will converge to the solution if RF is less than 2/3. For example, if RF=0.4 the successive values of the output are: 0.400, 0.320, 0.336, 0.333(see Figure 7-12).



Note that if a > 1, it is not possible to find a positive value of RF that will lead to convergence. It is necessary to use a negative RF.



The relaxation factor is chosen for each output independently and then the output is modified according to. The implemented rules to modify the relaxation factor were proposed by EMPA:



In other words, the relaxation factor is reduced (\*0.5) when the solution oscillates and it is increased (\*1.5) when the solution keeps progressing in the same direction.（对收敛进行加速和减速的作用）

（略）





The ASSlGN statement allows the assignment of files to logical unit numbers from within the TRNSYS input file. Files that are assigned to a logical unit number using an ASSlGN statement will be opened by the TRNSYS kernel and therefore may cause problems if a Type or routine in an external DLL attempts to open them or read from them. The kind of problem that can occur depends on what compiler was used to compile the TRNDll and the Type in the external DLL. The DESIGNATE keyword is available for users wishing to assign a logical unit to an external file without having the TRNSYS kernel open the file. The format for the ASSlGN statement is

ASSIGN语句允许用户在TRNSYS输入文件中将文件分配给逻辑单元号（Logical Unit Number）。通过 ASSIGN 语句分配给某个逻辑单元号的文件将由TRNSYS内核打开，因此如果某个外部 DLL 中的组件类型（Type）或子程序试图打开或读取这些文件，可能会导致问题。可能出现的问题取决于用于编译TRNDll和外部DLL中组件的编译器种类。为了解决这一问题，TRNSYS 提供了DESIGNATE关键字，供希望将逻辑单元号分配给外部文件但不希望TRNSYS内核打开该文件的用户使用。ASSIGN语句的格式如下：



其中：

**filename** is the full name of the desired file (including path, if necessary); file name must be less than or equal to maxFileWidth characters in length. Spaces are allowed in pathnames as long as the entire path is contained in quotes. Quote marks are not necessary if there are no spaces in the pathname. MaxFileWidth is set in the TrnsysConstants file located in the TRNSYS Source Code directory. maxFileWidth may be modified and a new TRNDll.dll created if necessary. Paths may be relative to the location of the input file.

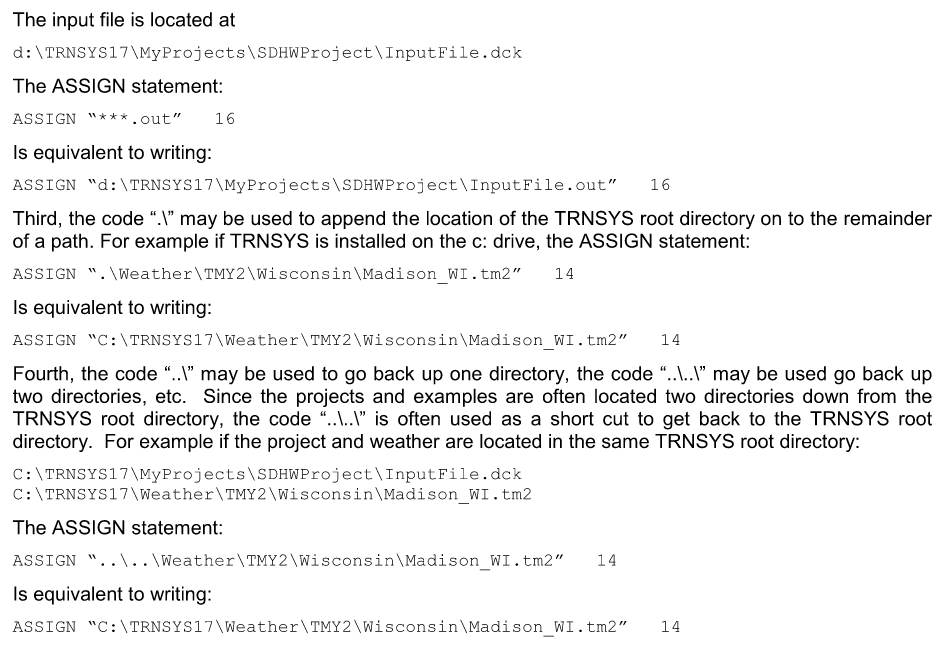
文件名是所需文件的完整名称（如需要，应包含路径）；其总长度必须小于或等于 maxFileWidth 所定义的字符数。只要整个路径被包含在引号中，路径名中可以包含空格。如果路径名中没有空格，则无需使用引号。maxFileWidth 的值是在 TRNSYS 源代码目录中的 TrnsysConstants 文件 里设定的。如有需要，可以修改该值，并重新生成新的 TRNDll.dll 文件。路径可以是相对于输入文件所在位置的相对路径。

**Iu** is the logical unit number to which filename is to be assigned.

lu 是要分配给文件名的逻辑单元号（Logical Unit Number）

Certain other conventions and "short cuts" apply to ASSIGN statements. First, if the path or file name contains spaces, the entire path must be enclosed in double quote marks. Second, the user may replace everything in the path (excepting the file extension) with the code "\*\*" In this case, the name and location of the input file will be appended on to the user specified file extension. For example:

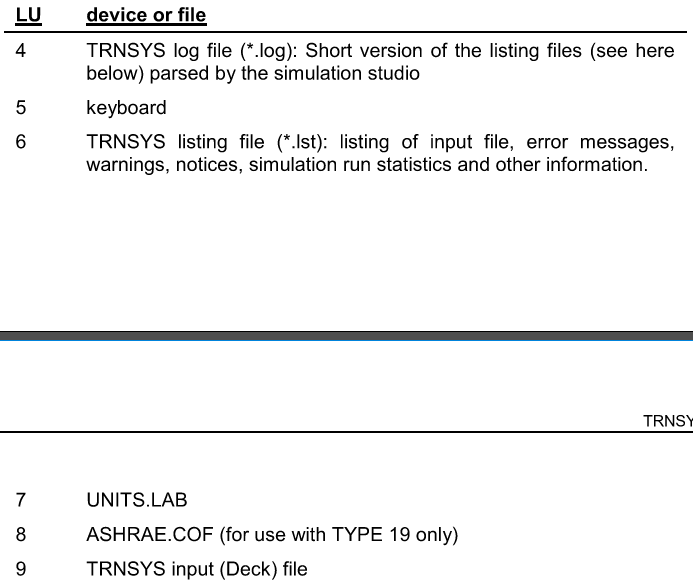
某些其他的约定和“快捷方式”也适用于ASSIGN语句。首先，如果路径或文件名中包含空格，则整个路径必须用双引号括起来。其次，用户可以使用代码 "\*\*" 替换路径中的所有内容（文件扩展名除外）。在这种情况下，输入文件的名称和位置将被附加到用户指定的文件扩展名上。例如：

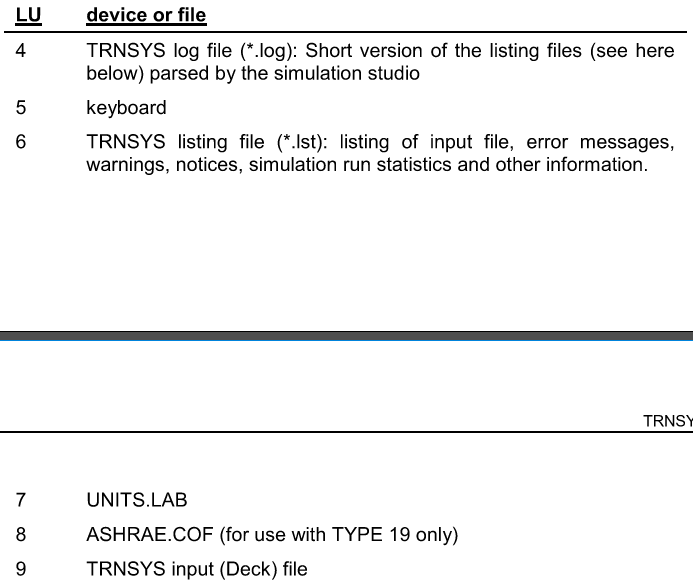


As many ASSIGN statements as allowed in Windows may be used in a TRNSYS input file and they may be placed anywhere before the END statement. The logical unit 6 is ASSIGNED automatically by the TRNSYS 17 kernel and is used for the list file, where information pertinent to a given simulation is written.

在TRNSYS输入文件中，可以使用Windows所允许数量的ASSIGN语句，并且这些语句可以放置在END语句之前的任意位置。逻辑单元号6由TRNSYS 17内核自动分配，并用于列表文件（list file），该文件中记录了与当前仿真相关的信息。

TRNSYS uses several logical unit numbers:





Additionally, the user must specify logical unit numbers when configuring output components, TYPE 9, and components that need to read data files. The re-use of the above listed logical unit numbers should be avoided at all costs.

此外，用户在配置输出组件（如 TYPE 9）、以及需要读取数据文件的组件时，也必须指定逻辑单元号。应极力避免重复使用上述已列出的逻辑单元号。

Some components, such as Type56 and TRNFLow, also use hard-coded Logical units. in general, user files should be directed to LUs greater than 30 (which is done automatically by the Simulation Studio).

For better compatibility with the TRNSYS utility program such as TRNEdit and TRNSED, the following convention should be used when ASSIGNing the output, and plot files within the input file (.dck). The output, and plot files should all have the same name as the input file. For example if the input file is called TEST.DCK and located in the TRNSYS17\test\directory, then the following ASSIGN statements should be used and placed within the input file:

一些组件，例如 Type56 和 TRNFlow，也会使用硬编码的逻辑单元号。一般情况下，用户文件应使用大于 30 的逻辑单元号（Simulation Studio 会自动完成这一分配）。

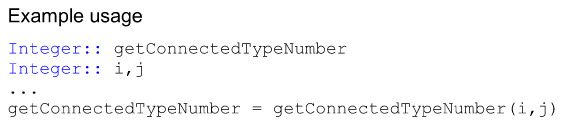
为了更好地与 TRNSYS 工具程序（如 TRNEdit 和 TRNSED）兼容，在输入文件（.dck 文件）中通过 ASSIGN 语句指定输出文件和绘图文件时，应遵循以下命名约定：输出文件和绘图文件的文件名应与输入文件相同。例如，如果输入文件名为 TEST.DCK，并位于 TRNSYS17\test\ 目录下，则应在输入文件中使用如下 ASSIGN 语句：



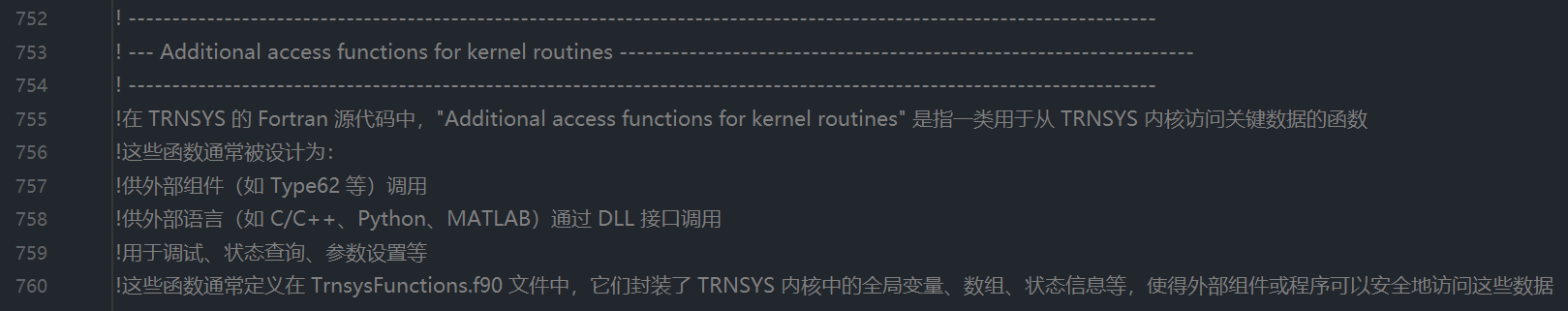
以下内容来自06-TRNEdit的6.3.15（结束）



An integer function that returns the type number of the unit connected to jth input of Unit i. When using the Powell's Method solver (SOLVER 1), you should use the similar function getConnectedTypeNumberS1().

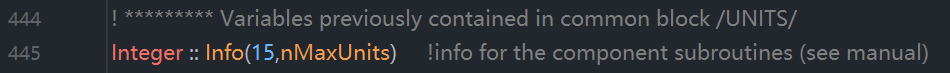


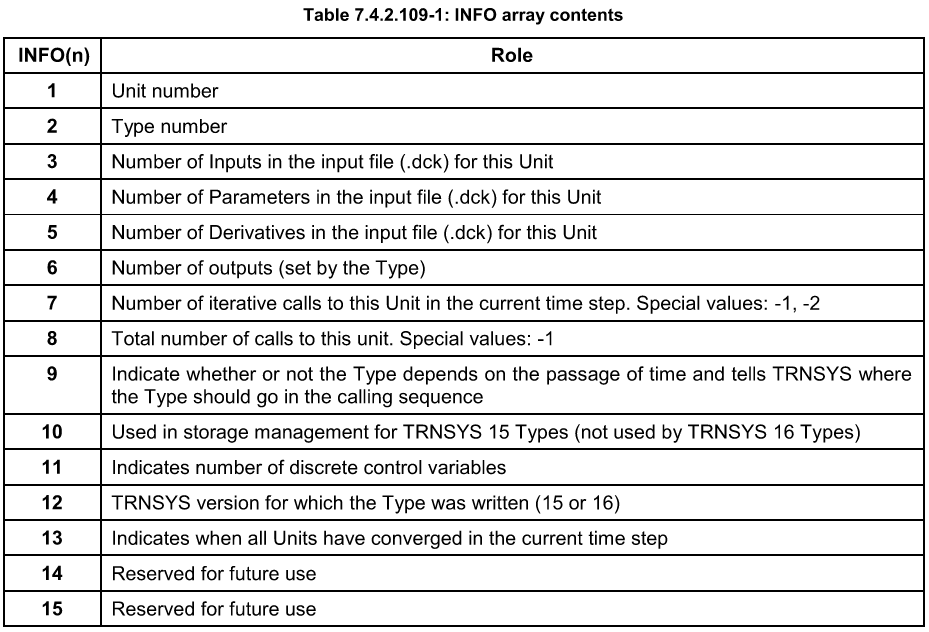
function getConnectedTypeNumber(i, j)属于additional access functions for kernel routines.



在getConnectedTypeNumber(i, j)程序中使用了incon、inp（TrnsysData.f90），详细解释见getConnectedOutputNumber(i, j)。

在getConnectedTypeNumber(i, j)程序中使用了info（TrnsysData.f90）。





Info()的详细内容详见7.4.3

