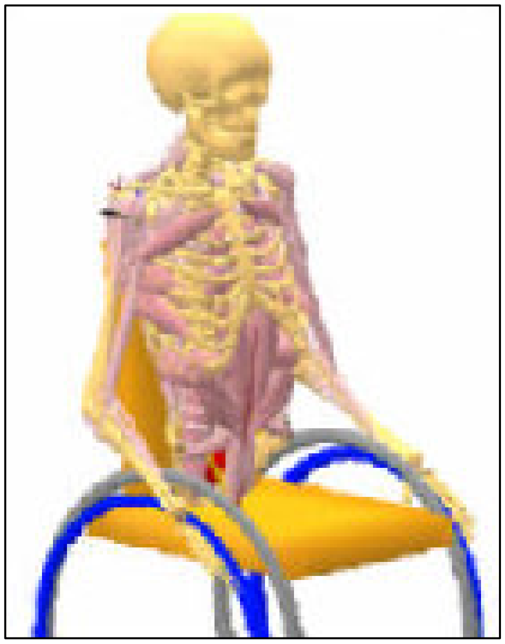
Getting Started With AnyBody

Disclaimer

All information presented in this tutorial has been compiled from AnyBody^{TM} Tutorials Version 5.0.0, December 2010 provided by AnyBody^{TM} Technology.



Tugba Ozedirne

Table of Contents

I. INTRODUCTION

The AnyBody^{TM} software is used to develop full-body musculoskeletal simulations of various daily-life activities by allowing the user to calculate muscle and joint reactions (and more) on a given movement.

The AnyBody^{TM} Modeling System is used in the following situations:

1. Loading and running an existing model
2. Modifying an existing model
3. Constructing an entirely new model

The existing models can be obtained from the AnyBody^{TM} model repository, a library containing predefined models and body parts, which can be obtained from the AnyScript^{TM} Community. Once you download the repository and unpack it, you notice it is comprised of several other files and is organized as follows:

```
AMMRVx.x
    |
    |------- Application------ Beta (unfinished models)
    |                   |
    |------- Body       |----- Examples (finished models)
                        |
                        |----- Validation (models used for validation purposes only)
```

You will primarily be working with the Examples folder, as it contains finished and working models. As such, they serve a good starting for an AnyBody^{TM} user.

II. AnyScript^{TM}

AnyScript^{TM} is the model definition language of AnyBody^{TM} Modeling System. It has a text-based format and an object-oriented structure that is used to construct bodies and the environment the body interacts with.

The syntax is similar to Java, Javascript, or C++ computer programs. Just like any other programming language you can define variables and assign values to them. For instance, you may write:

```
AnyVar a = 10.0;
AnyVar b = a;
```

Folders can also be considered as variables, and you can define a folder as follows:

```
AnyFolder MyFolder = {
   // A whole lot of stuff inside
};
```

However, folders cannot be assigned like a and b above, so the following cannot be performed:

```
AnyFolder MyFolderCopy = MyFolder;
```

In general, you can only do direct assignment of value variables, i.e. variables containing numbers or collection of numbers:

```
AnyVector aa = {1,2,3};
AnyVector bb = aa;
```

But folder assignments are not allowed. However, you can refer to large portions of the model in just one assignment operation. You can do this by using a reference:

```
AnyFolder &MyFolderCopy = MyFolder;
```

The ampersand, '&', in front of the variable name specifies that this is a reference. In this case, MyFolderCopy will point to MyFolder, so whatever you changes you make to MyFolderCopy will also occur in MyFolder.

You can also use include files in AnyScript^{TM}. The purpose to include files is to divide a long file into smaller ones. Typically, you have one main file that contains statements to other files. The syntax is as follows:

```
#include "muscles.any"
```

The # symbol indicates that this is a macro statement and is the only statement that does not require a semicolon in the end. The include statement provides some useful implications:

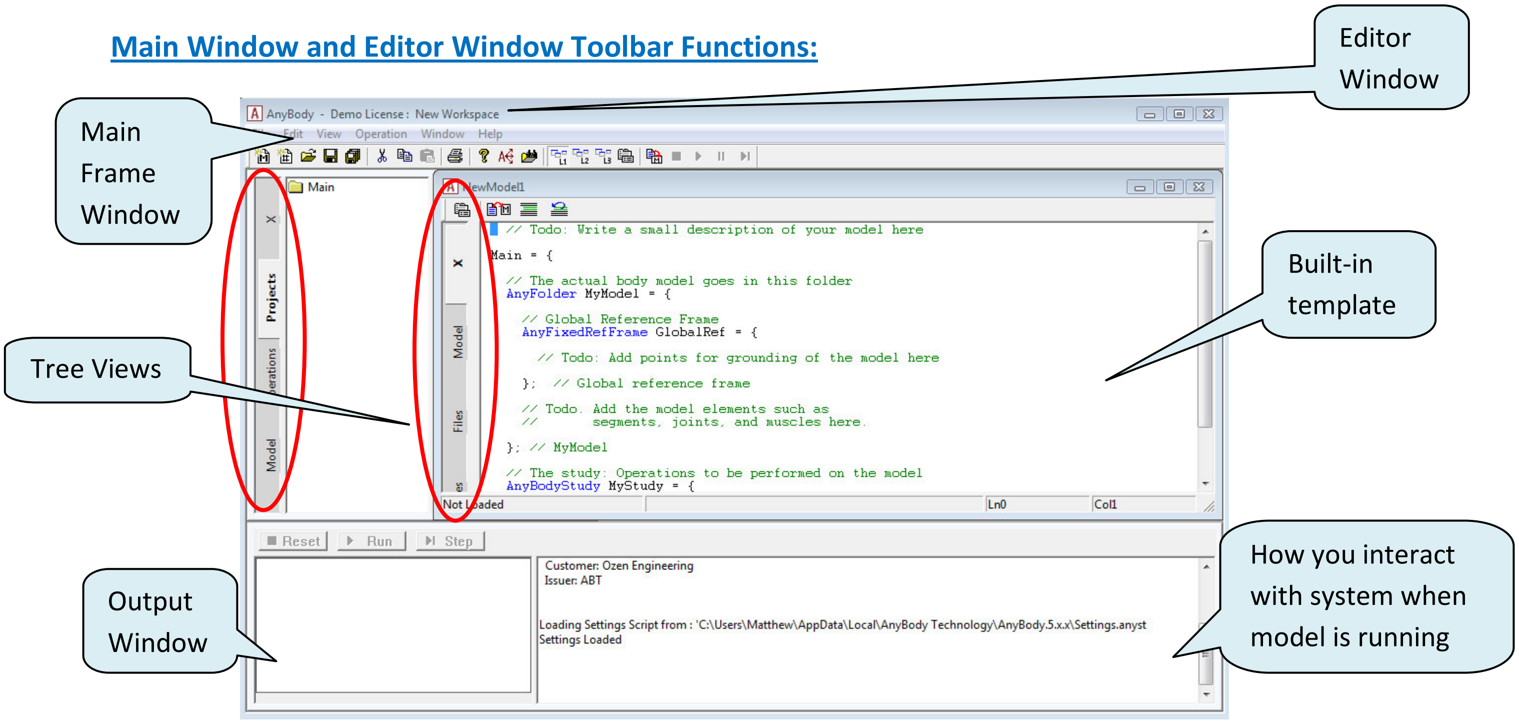
1. Divide large models into smaller and more manageable clusters
2. Create blocks of data that can be reused in your model by putting the data in a file and including it in several places in your main file.
3. Create libraries of files containing elements that you often use in your models, so that you can build a new model by simply included the files you made previously.

III. BASIC INFORMATION

Below is a short description to help you get started using AnyScript^{TM} and navigate through some of the interface features of the AnyBody^{TM} Modeling System.

To create a new AnyScript^{TM} model: Click File -> New Main.

Main Window and Editor Window Toolbar Functions:



```
// Todo: Write a small description of your model here

Main = {

  // The actual body model goes in this folder
  AnyFolder MyModel = {

    // Global Reference Frame
    AnyFixedRefFrame GlobalRef = {

      // Todo: Add points for grounding of the model here

    };  // Global reference frame

    // Todo. Add the model elements such as
    //       segments, joints, and muscles here.

  }; // MyModel

  // The study: Operations to be performed on the model
  AnyBodyStudy MyStudy = {
```

```
Customer: Ozen Engineering
Issuer: ABT


Loading Settings Script from : 'C:\Users\Matthew\AppData\Local\AnyBody Technology\AnyBody.5.x.x\Settings.anyst
Settings Loaded
```

Tree Views:

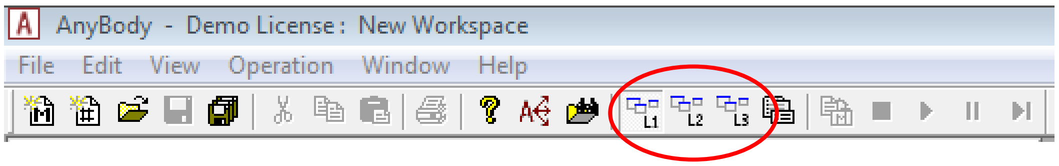
1. Model: shows all objects
2. Operation: shows subset of the model objects and is controlled by the Run, Step, and Reset buttons
3. File: shows all the files in a model

Additional tree views in editor window:

1. Class: contains all classes and assists in inserting the code to create objects
2. Global and Function: shows the globally available elements

Workspaces:

There are three user-defined layouts (L1, L2, L3) located at the top of the main window.


AnyBody - Demo License : New Workspace
File Edit View Operation Window Help
L1
L2
L3

Each layout has a different window setup that you can switch between. Having multiple layouts allows you to organize and preserve your active windows when working on a model. To preserve the window layouts, Click File -> Save Workspace.

Editor Window:

The editor window is where you create your model. To open a new editor window with an empty file, click .

Features:

a. Syntax highlighting: The AnyScript^{TM} editor recognizes predefined class names and highlights them automatically. It does this as you are typing, and it is a great help to avoid misspellings that might later lead to syntax errors. There are two types of highlights in AnyScript^{TM}; blue and green. Preserved words are highlighted in blue and documentation comments are highlighted in green:

AnySeg	The prefix "Any-" indicates reserved words in AnyScript^{TM} and are highlighted in blue.
// AnySeg	You can turn entire blocks of code into a comment by encapsulating it into a pair of /* */ delimiters. Other syntactical forms include: /// ..., /** ... */, ///< ..., and /**< ... */. Another easy way to temporarily remove and re-activate several lines from the file is to block the text and use the two buttons in the toolbar of the Editor window. This automatically places or removes double slashes in front of each line in the block.

b. Auto format: This feature allows you to block a part of the code or the entire file and indent it in one simple step. To perform this operation, Click Edit -> Format Indentation. This is a very useful feature; because it helps you keep your script organized and allow you to spot syntax errors very easily.

Model View Window:

The model view window displays the graphical representation of the model. To open this window, Click Window -> Model View (new). When active, the Model View updates the model as it is being loaded.

Note: Usually when a model is loaded it is not assembled correctly. To resolve this issue, Click Operation Tree -> InitialConditions -> Run. Doing so assembles the model in the correct position because it resolves kinematic constraints.

Model View Toolbar functions:

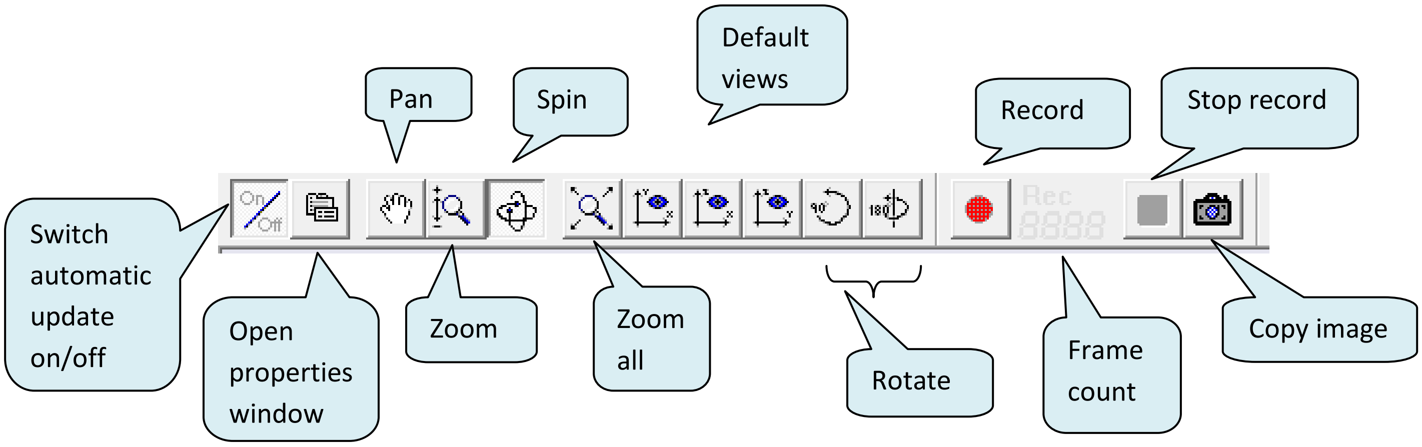
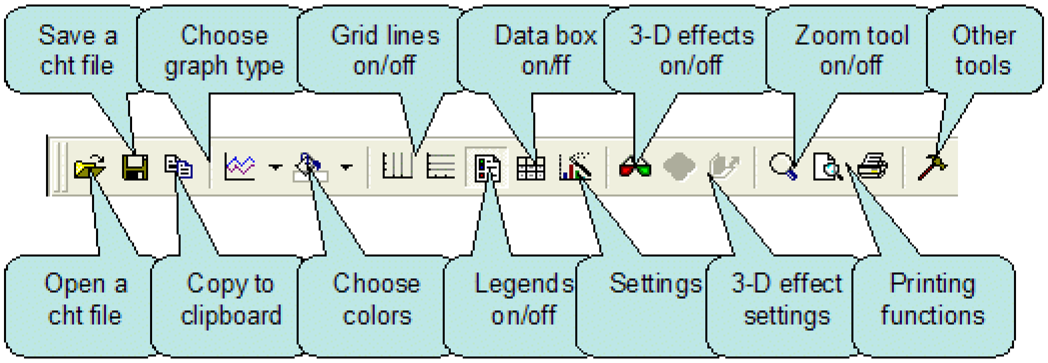


Chart View:

AnyBody^{TM} Modeling System has two types to graphing results; ChartFX and AnyChart View. ChartFX is used for two-dimensional plots, whereas AnyChart View is used for three-dimensional surface plots.

To open a ChartFX, Click Window -> ChartFX (new)

ChartFX toolbars:



How to Load And Analyze a Model:

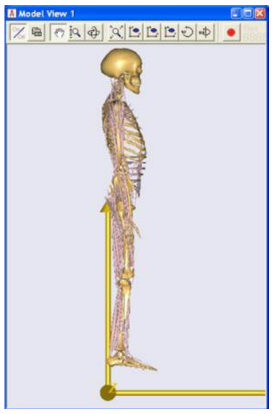
1. To open a file, click File -> Open
2. To load the model. Click
3. To run an analysis, Operations Tree -> InverseDynamics -> Run
4. To view results, Window -> ChartFX (new) -> expand the tree in editor window and click on the nodes to display plots.

Below is an example that incorporates these steps.

1. From the repository, click Applications -> Examples -> StandingModel.main.any

2. Load the model, click

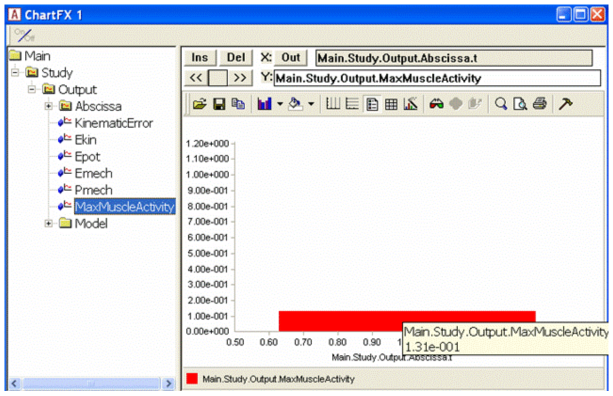
3. Click Window -> Model View (new) to display model:



4. Click Operations tree -> InverseDynamics -> Run

5. Click Window -> ChartFX (new)

6. In the editor model tree, expand Study -> expand Output -> expand Abscissa -> click MaxMuscleActivity:



X: Main.Study.Output.Abscissa.t

Y: Main.Study.Output.MaxMuscleActivity

Main.Study.Output.MaxMuscleActivity
1.31e-001

IV. TYPES OF STUDIES

Studies and operations specify things to do to the model. The study is the "collector", where it collects a model definition and operations that execute the model and displays the results to be investigated. Operations are the things you can do to the model.

Note: studies need to be defined as special classes and you need to declare them manually in the model. This allows you to have multiple studies in the same model. Studies essentially become objects in the model and therefore, you can have as many as you need.

A study is a folder, a pair of braces between which you can add specifications. Whatever you put between the braces becomes part of the study.

Note: when you create a new model by clicking File -> New Main, the system automatically inserts an AnyBodyStudy for you:

```
// The study: Operations to be performed on the model
AnyBodyStudy MyStudy = {
AnyFolder &Model = .MyModel;
Gravity = {0.0, -9.81, 0.0};
};
```

Standard operations of a study:

1. Initial Conditions: Assembles the model correctly by getting rid of kinematic constraints. The model is initialized into the initial positions from load time.
2. Kinematics: Performs the movement you imposed when you defined the model. There is no calculation of forces and the system does not need to be balanced, but the system needs to be "kinematically determinate." Muscles are not needed to run this operation.
    a. An AnyBody^{TM} model is a collection of segments that has 6 movement directions (i.e. degrees of freedom). If you have n segments in a model, you will have 6n degrees of freedom or 6n equations, in mathematical terms. Therefore kinematic analysis solves 6n equations with 6n unknowns. Three situations can arise with these equations:
        i. Kinematically determinate: solvable 6n equations. This is what is needed to run the analysis.
        ii. Kinematically over-determinate: Have more constraints than needed.

iii. Kinematically indeterminate: Have fewer constraints than needed.

Essentially the kinematics operation is an analysis. It assembles the data when it is run, and the results can be investigated through ChartFX. To open, click Window -> ChartFX (new). The results you can analyze include positions (r), velocities (rDot), and accelerations (rDDot).

3. Inverse Dynamics: Simulation of the forces involved in a given movement or posture. This operation utilizes Kinematics, as it requires the correctly defined movement or posture. In addition it requires muscles or motors to drive the model.

V. Muscle Models

Muscles are needed to drive a movement and in AnyBody^{TM} there are three muscle models:

1. AnyMuscleModel: Assumes constant strength of the muscle.
2. AnyMuscleModel2ELin: Bilinear model that takes length and contraction velocity into account.
3. AnyMuscleModel3E: Three element model that takes serial and parallel elastic elements and fiber length and contraction velocity into account.

In the AnyBody^{TM} Modeling System, muscles consist of two computational models:

1. Kinematic model: determines the muscle's path from origin to insertion depending on the posture of the body. This requires finding the length and contraction velocity of the muscle.
2. Strength mode: determines the muscle's strength and passive elastic force depending on the kinematic state of the muscle.

The muscle models can also be linked to different types of muscles by:

1. AnyViaPointMuscle: a muscle that passes through any number of nodes on segments on its way from origin to insertion
2. AnyShortestPathMuscle: a muscle that can wrap over geometries.
3. AnyGeneralMuscle : a more standard actuator-type muscle that can be attached to a kinematic measure.

Below is an example for modeling the simplest muscle type:

```
AnyMuscleModel <ObjectName> = {
      F0 = 0;
};

AnyViaPointMuscle <ObjectName> = {
      AnyMuscleModel &<Insert name0> = <Insert object reference
       (or full object definition)>;
      AnyRefFrame &<Insert name0> = <Insert object reference (or
       full object definition)>;
      AnyRefFrame &<Insert name1> = <Insert object reference (or
       full object definition)>;
      //AnyRefFrame &<Insert name2> = <Insert object reference
       (or full object definition)>;
};
```

VI. Musculoskeletal Modeling

Musculoskeletal modeling is an advanced application of the law of mechanics. In an AnyBody™ model the mechanical elements include:

1. Segments: Used to represent bones and other rigid elements. This class can be inserted into your model by clicking Classes tree -> expanding Class List -> right clicking AnySeg -> Insert Class Template:

   ```
   AnySeg <ObjectName> = {
    //r0 = {0, 0, 0};
     //rDot0 = {0, 0, 0};
     //Axes0 = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
     //omega0 = {0, 0, 0};
    Mass = 0;
     Jii = {0, 0, 0};
     //Jij = {0, 0, 0};
     //sCoM = {0, 0, 0};
     //JaboutCoMOnOff = Off;
    };
   ```

   Note: if you add AnyDrawSeg drw = {}; then you can visualize the segment when you open Model View.

2. Joints: Used to connect segments. Joints act as constraints (i.e. remove degrees of freedom). However, joint constraints are not imposed until you perform an analysis. The SetInitialConditions operation is therefore used for this purpose because it resolves the constraints and assembles the model.

3. Drivers: Used to specify the movement the body will perform.

4. Kinematic Measures: Abstraction representation of kinematic constraints. By incorporating a kinematic measure, you can study your model's development and control it. Below is an example to create a kinematic measure:

   ```
   }; // Jnts folder
     AnyFolder KinematicMeasures = {
     AnyKinLinear WristPos = {
     // These are the nodes that the measure refers to
     AnyFixedRefFrame &Ground = Main.ArmModel.GlobalRef;
     AnyRefNode &UpperArmNode =
       Main.ArmModel.Segs.LowerArm.HandNode;
       Ref = 0;
       };
   ```

```
    }; // KinematicMeasures
```

To drive this measure:

```
AnyFolder Drivers = {
           AnyKinEqSimpleDriver HandMotionXY = {
           AnyKinLinear &Jnt =..KinematicMeasures.WristPos;
           MeasureOrganizer = {0,1};
           DriverPos = {0.4,-0.5};
           DriverVel = {0.2,0.5};
           DriverAcc = {0.0,0.0};
           Reaction.Type = {Off,Off};  // The muscles must
             do the work
          };
}; // Drivers folder
```

5. Forces: Forces applied to the model.

   To add forces to your model, you can insert a template by clicking Classes tree -> expanding Class List -> right clicking AnyForce3D -> InsertClassTemplate:

```
AnyFolder Loads = {
   AnyForce3D <ObjectName> = {
     //F = {0, 0, 0};
     //Flocal = {0, 0, 0};
     AnyRefFrame &<Insert name0> = <Insert object reference
      (or full object definition)>;
   };
 }; // Loads folder
```

VII. Types of Scaling

Scaling of musculoskeletal models is important in order to account for varying sizes of individuals. In AnyBody^{TM}, there are seven scaling laws:

1. ScalingStandard: no scaling, uses standard model size.
2. ScalingUniform: equal scaling in all directions.
3. ScalingLengthMass: scaling takes mass into account.
4. ScalingLengthMassFat: scaling takes mass and fat into account.
5. ScalingUniformExt: input is external measurement, equal scaling in all directions.
6. ScalingLengthMassExt: input is external measurement, scaling takes mass into account.
7. ScalingLengthMassFatExt: input is external measurement, scaling takes mass and fat into account.

VIII. Troubleshooting

While working with AnyScript^{TM}, you might come across errors in your model that will prevent you from analyzing it. In AnyBody^{TM}, you will encounter two types of errors:

1. Load-time errors: the AnyScript^{TM} model contains syntax errors
2. Run-time errors: a successfully loaded model refuses to be analyzed

Below is a list of common load-time errors:

Forgotten semicolon	Every statement in AnyScript^{TM} must be terminated by a semicolon. End braces must also have a semicolon after them.
Unbalanced braces	Braces {} are used to group things together in AnyScript^{TM}. You get error messages when you have too few or too many braces. To avoid this problem, use consistent indentation.
Mix-up of decimal points and commas	The AnyBody^{TM} Modeling System uses decimal points. You will get a syntax error if you type a comma instead of decimal point.
Mix-up of 'O' and '0'	The letter O and the number 0 are different.
Mix-up of 'l' and '1'	The letter l and the number 1 are different.
Inconsistent use of capitals	AnyScript^{TM} is case-sensitive.
Missing reference operator	You will get the error message ‘Folder assignment expected for this object’ if you type: `AnyFolder MyFolderCopy = MyFolder;` You get this error message because you need an ‘&:’ `AnyFolder &MyFolderCopy = MyFolder;`
Missing expected members	The following: `AnySeg arm = {` `Mass = 12;` `};` will produce the error message ‘Obligatory initialization of member :

	AnyVec3 Jii is missing’ because the property Jii must be given a value.
	
	The following:
	
	`AnySeg arm = {`
	`t = 12;`
	`};`
	
	leads to the error message ‘t : Initialization denied’ because ‘t’ is a protected variable. The value is given by the system not the user.