# Debug Model Assembly from Assy File

## Table of Contents

## 1: Introduction

In this section of the TKC manual we will illustrate the use of the TKC Assembly loader to debug an Assembly. Assembly files can be separated in action blocks, each block represents a separate part of the complete Assembly model.

This functionality allows users to:

1) Load an Assembly file piece-by-piece;
2) Test the model in the current state of completion, including deleting and recreating objects;
3) Continue loading the assembly file to a next block;

Using this approach, the assembly is functionally used as a model creation template which is used to check models and strongly supports interactive improvement of toolkit based model components.

## 2: Open Assembly Loader Dialog

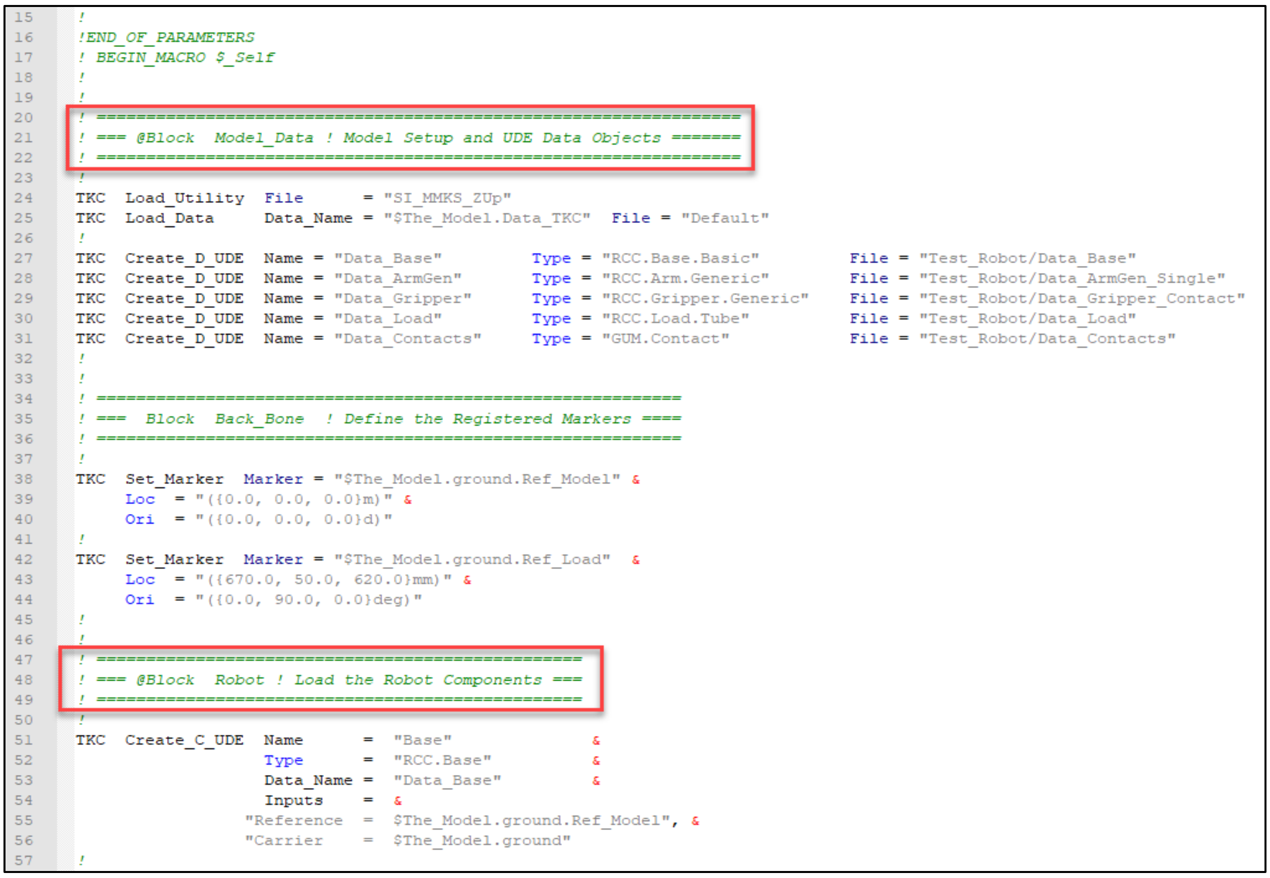
The generic Assembly loader is opened from an empty model. The dialog Assy_File field is set to *Test_Robot* and the Assy_Lib List option is set to *User* to allow loading this specific Assembly. The contents of the Assembly file can be printed by pressing the Assy_File button or by opening it in any available text editor.

The image below illustrates the syntax of denoting Blocks in the *Test_Robot* Assembly file. The marked lines 21 and 48 are the start and end lines of the *Model_Data* Block. Line 48, the start line of block *Robot* is automatically the end line of the previous block *Model_Data.*

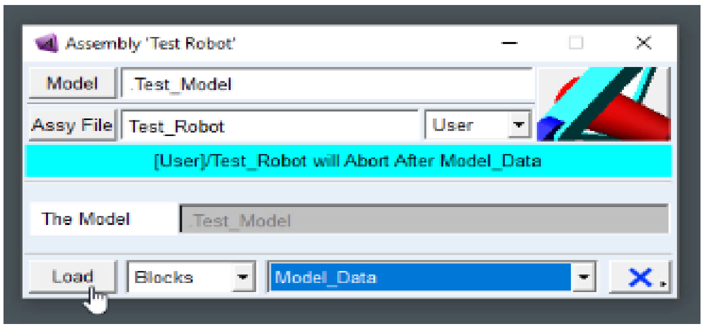
The minimal syntax of a valid Block Line is: *' ! @Block Block_Name ! Block Comment'*

Note 1: To ADAMS, a Block Line is a comment line. Blocks are parsed using a separate Python parser to create a dedicated Assembly file which executes the block dependent commands.

Note 2: Line 35 is not a valid Block Line due to the missing '@' character before the *Block* keyword.


```
15   !
16   !END_OF_PARAMETERS
17   ! BEGIN_MACRO $_Self
18   !
19   !
20   ! ==================================================================
21   ! === @Block  Model_Data ! Model Setup and UDE Data Objects =======
22   ! ==================================================================
23   !
24   TKC  Load_Utility  File      = "SI_MMKS_ZUp"
25   TKC  Load_Data     Data_Name = "$The_Model.Data_TKC"  File = "Default"
26   !
27   TKC  Create_D_UDE  Name = "Data_Base"        Type = "RCC.Base.Basic"         File = "Test_Robot/Data_Base"
28   TKC  Create_D_UDE  Name = "Data_ArmGen"      Type = "RCC.Arm.Generic"        File = "Test_Robot/Data_ArmGen_Single"
29   TKC  Create_D_UDE  Name = "Data_Gripper"     Type = "RCC.Gripper.Generic"    File = "Test_Robot/Data_Gripper_Contact"
30   TKC  Create_D_UDE  Name = "Data_Load"        Type = "RCC.Load.Tube"          File = "Test_Robot/Data_Load"
31   TKC  Create_D_UDE  Name = "Data_Contacts"    Type = "GUM.Contact"            File = "Test_Robot/Data_Contacts"
32   !
33   !
34   ! ============================================================
35   ! ===  Block  Back_Bone  ! Define the Registered Markers ====
36   ! ============================================================
37   !
38   TKC  Set_Marker  Marker = "$The_Model.ground.Ref_Model" &
39        Loc  = "({0.0, 0.0, 0.0}m)" &
40        Ori  = "({0.0, 0.0, 0.0}d)"
41   !
42   TKC  Set_Marker  Marker = "$The_Model.ground.Ref_Load"  &
43        Loc  = "({670.0, 50.0, 620.0}mm)" &
44        Ori  = "({0.0, 90.0, 0.0}deg)"
45   !
46   !
47   ! ================================================
48   ! === @Block  Robot ! Load the Robot Components ===
49   ! ================================================
50   !
51   TKC  Create_C_UDE  Name      =  "Base"            &
52                      Type      =  "RCC.Base"        &
53                      Data_Name =  "Data_Base"       &
54                      Inputs    =  &
55                    "Reference  =  $The_Model.ground.Ref_Model", &
56                    "Carrier    =  $The_Model.ground"
57   !
```

During the execution of lines in an Assembly file, the criterium for running code inside a Block is the absence (as a model object) of the first TKC object listed inside a Block.



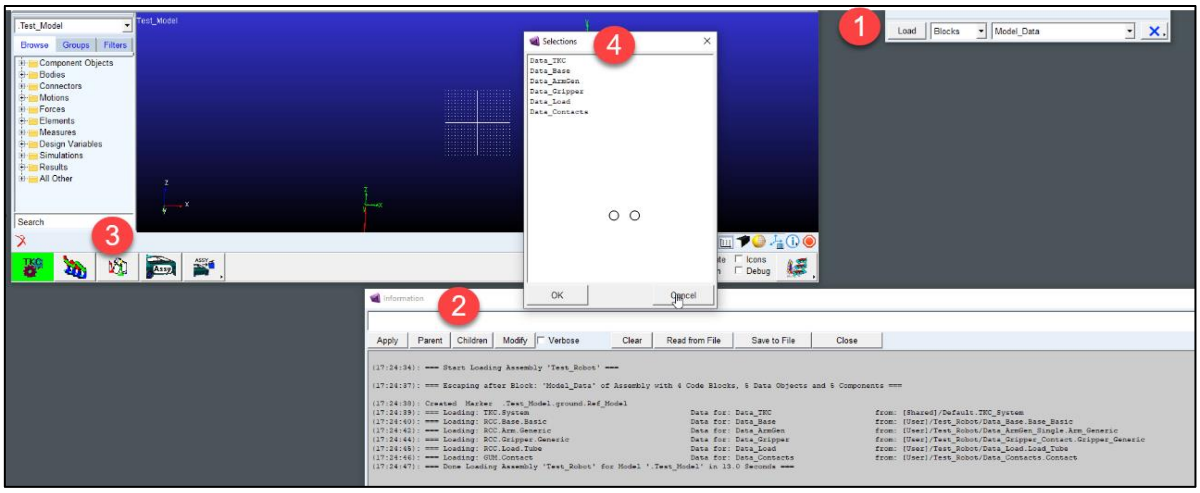
Model: Test_Model
Assy File: Test_Robot | User

[User]/Test_Robot will Abort After Model_Data

The Model: Test_Model

## 3: Load and Check the *Model_Data* Assembly Block

Thus, the code assumes block *Model_Data* to be completed when object *Data_Base* (see line 27) exists in the model. The code will then jump to the start of the next Block *Robot* which will be executed only when no model object *Base* is found. The image above shows the Assembly dialog before creating the model objects until the end of the *Model_Data* Block. Obviously, in the *Model_Data* block, no TKC components are defined in the model but only data elements for the model components.



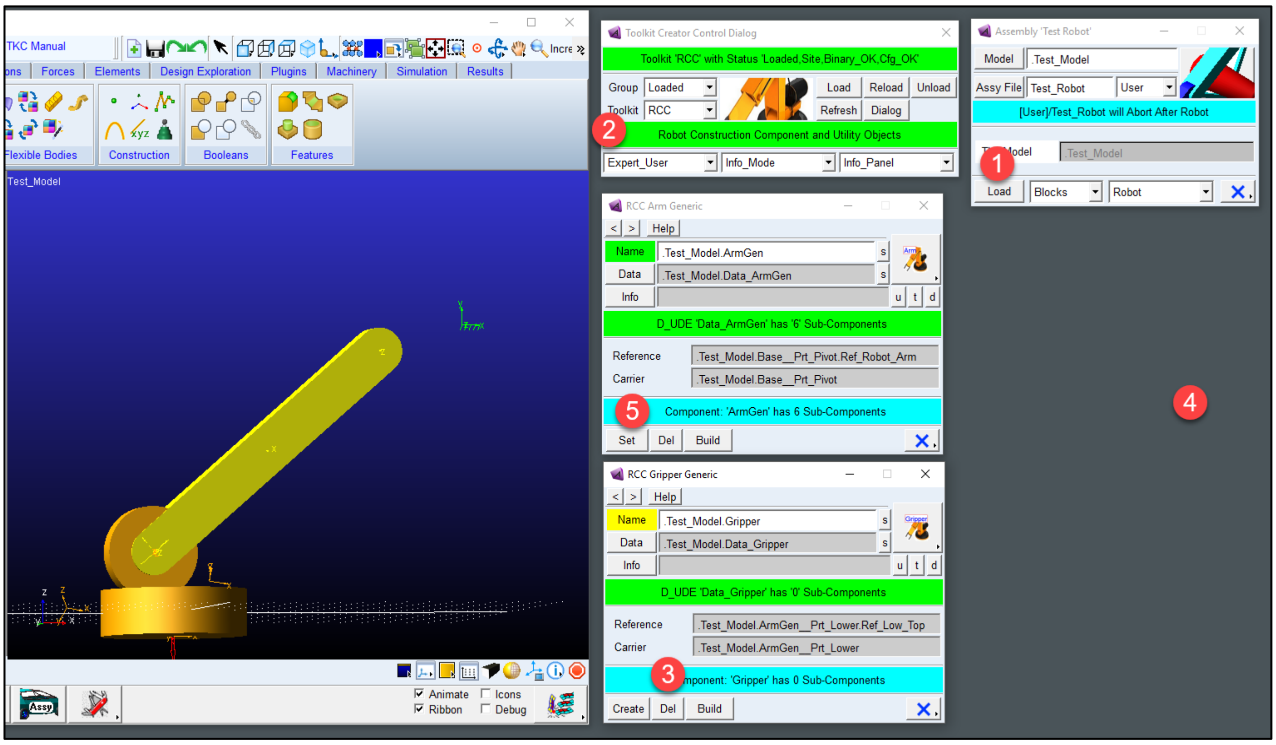
```
Data_TKC
Data_Base
Data_ArmGen
Data_Gripper
Data_Load
Data_Contacts
```

```
(17:24:34): === Start Loading Assembly 'Test_Robot' ===

(17:24:37): === Escaping after Block: 'Model_Data' of Assembly with 4 Code Blocks, 6 Data Objects and 6 Components ===

(17:24:38): Created  Marker   .Test_Model.ground.Ref_Model
(17:24:39): === Loading: TKC.System                     Data for: Data_TKC              from: [Shared]/Default.TKC_System
(17:24:40): === Loading: RCC.Base.Basic                 Data for: Data_Base             from: [User]/Test_Robot/Data_Base.Base_Basic
(17:24:42): === Loading: RCC.Arm.Generic                Data for: Data_ArmGen           from: [User]/Test_Robot/Data_ArmGen_Single.Arm_Generic
(17:24:44): === Loading: RCC.Gripper.Generic            Data for: Data_Gripper          from: [User]/Test_Robot/Data_Gripper_Contact.Gripper_Generic
(17:24:45): === Loading: RCC.Load.Tube                  Data for: Data_Load             from: [User]/Test_Robot/Data_Load.Load_Tube
(17:24:46): === Loading: GUM.Contact                    Data for: Data_Contacts         from: [User]/Test_Robot/Data_Contacts.Contact
(17:24:47): === Done Loading Assembly 'Test_Robot' for Model '.Test_Model' in 13.0 Seconds ===
```

The above image shows the process during and after executing this Block: (1) after executing the Assembly dialog, (2) a list of data objects is created which can be edited by (3) clicking on the data stack icon to show the (4) Data objects selection menu. Markers *Ref_Model* and *Ref_Load* used as references for the robot Base part and the *Load* respectively were also defined in this code block.

## 4: Load, Delete and Update *Robot* Components

Similarly, model components required for the robot parts are defined by the *Robot* Block of the Assembly. While designing or updating the macro code of the RCC toolkit, users can use the Block functionality as Follows: (1) Load objects from an Assembly, (2) set User Mode to Expert_User in the *Control_Dialog*, (3) delete a chain of components, (4) edit RCC macro files or add functionality and finally, (5) re-create components in the same sequence as listed in the Assembly file.



Toolkit 'RCC' with Status 'Loaded,Site,Binary_OK,Cfg_OK'
Group: Loaded
Toolkit: RCC
Robot Construction Component and Utility Objects
Expert_User | Info_Mode | Info_Panel

Model: .Test_Model
Assy File: Test_Robot | User
[User]/Test_Robot will Abort After Robot
Model: Test_Model
Blocks: Robot

RCC Arm Generic
Name: .Test_Model.ArmGen
Data: .Test_Model.Data_ArmGen
D_UDE 'Data_ArmGen' has '6' Sub-Components
Reference: .Test_Model.Base__Prt_Pivot.Ref_Robot_Arm
Carrier: .Test_Model.Base__Prt_Pivot
Component: 'ArmGen' has 6 Sub-Components

RCC Gripper Generic
Name: .Test_Model.Gripper
Data: .Test_Model.Data_Gripper
D_UDE 'Data_Gripper' has '0' Sub-Components
Reference: .Test_Model.ArmGen__Prt_Lower.Ref_Low_Top
Carrier: .Test_Model.ArmGen__Prt_Lower
mponent: 'Gripper' has 0 Sub-Components

Note 1: In the robot model, object *Gripper* is connected to object *ArmGen*. Therefore, to update the definition of *ArmGen* we first have to delete *Gripper* and then *ArmGen* and recreate these objects in reverse order. Meanwhile, the component dialogs must not be closed as they contain the parameters of components.

Note 2: In the image shown, *Gripper* is deleted which is reflected by the yellow Name label on the dialog. The fields data in the dialog are remainders from the previous instance of Gripper.

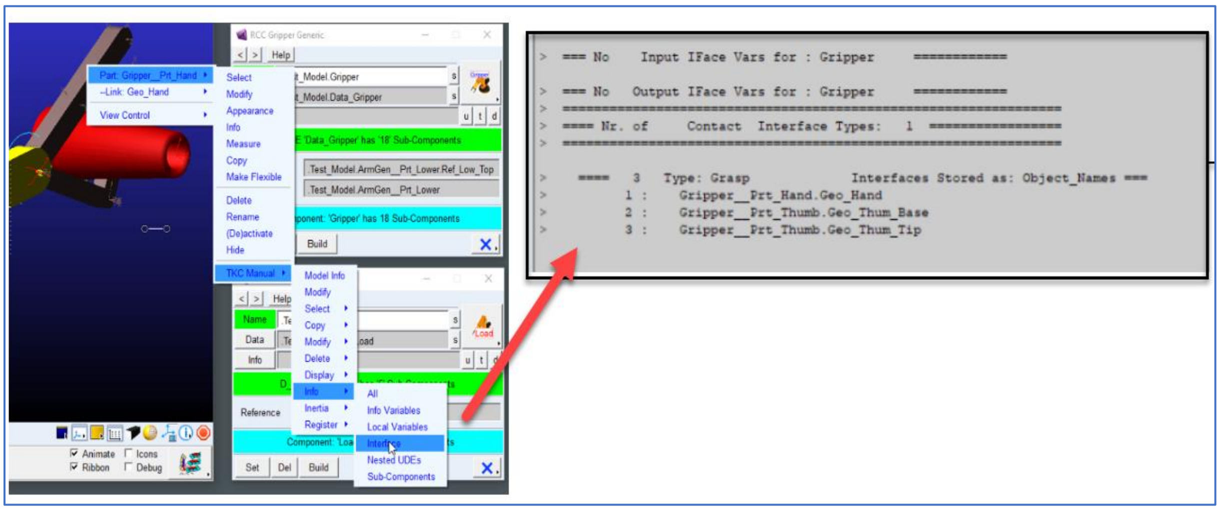
Note 3: See section *Generic versus Specific Components* for more info on the *Generic* Sub-Type

## 5: Create *Load_Contacts* Block and explain Contact Interface

Selecting the *Load_Contact* list option and executing the Assembly dialog will start the creation of the *Load* object and the contacts between *Load* and *Gripper*. The textual info generated by the Assembly loader shows that three contacts are created.

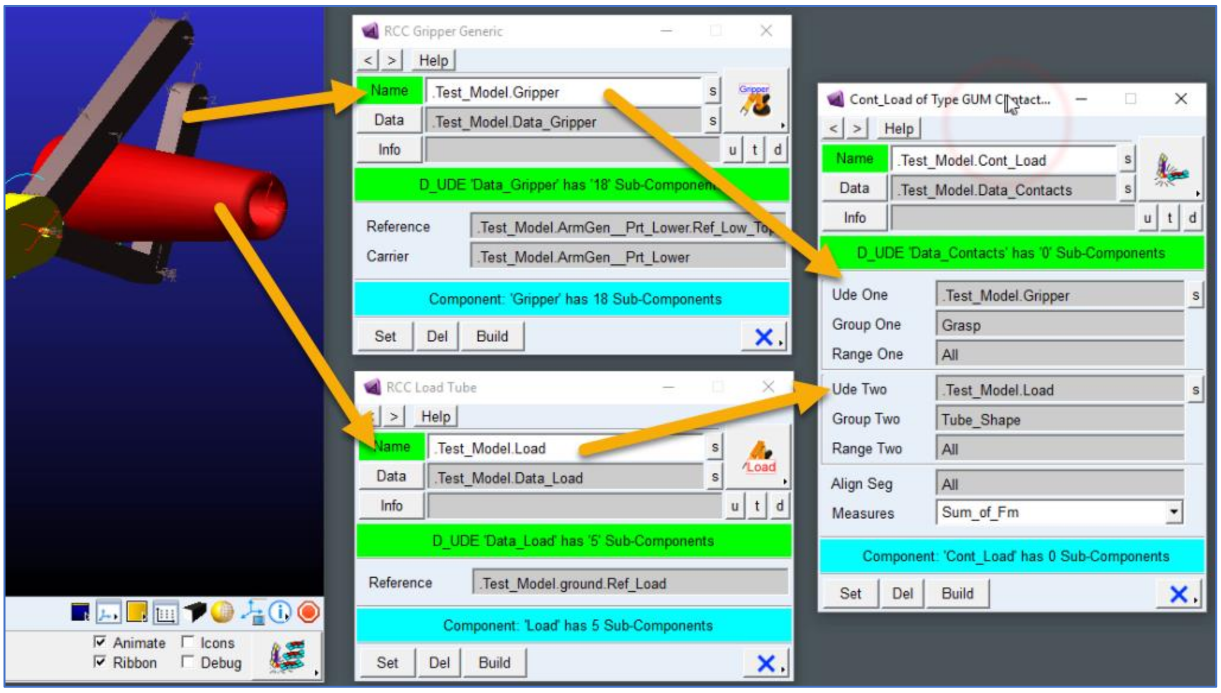
The method used for the definition of contacts is as follows:

- Both *Load* and *Gripper* define some Contact Interface objects by a call to the TKC Utility *Manage_IFace* embedded in the component topology macro or sub-components.
- For debugging, interface objects information can be listed with TKC pop-up menus on components by selecting: *-Info, -Interface*. In most cases, the lists can also be retrieved from a component child object (such as part *Gripper__Prt_Hand*)
- The lists show geometries in a labeled contact group of the component. In the image below, a contact list labeled *Grasp* is shown for the *Gripper* object containing three geometries on parts *Gripper__Part_Hand* and *Gripper__Prt_Thumb*.



```
>  === No    Input IFace Vars for : Gripper     ============

>  === No   Output IFace Vars for : Gripper     ============
>  ===============================================================
>  ==== Nr. of     Contact  Interface Types:   1  ================
>  ===============================================================

>    ====    3   Type: Grasp              Interfaces Stored as: Object_Names ===
>             1 :    Gripper__Prt_Hand.Geo_Hand
>             2 :    Gripper__Prt_Thumb.Geo_Thum_Base
>             3 :    Gripper__Prt_Thumb.Geo_Thum_Tip
```

- In the *Test_Robot* Assembly, a *GUM.Contact_2UDES* contact component named *Cont_Load* is defined between contact group *Grasp* of *Gripper* and contact group *Tube_Shape* of *Load*. Per default, all objects in *Grasp* will contact all objects in *Tube_Shape*. The creation dialog of *Cont_Load* is shown in the image below showing the required input parameters to the component.



RCC Gripper Generic

Name: .Test_Model.Gripper
Data: .Test_Model.Data_Gripper
Info:
D_UDE 'Data_Gripper' has '18' Sub-Componen
Reference: .Test_Model.ArmGen__Prt_Lower.Ref_Low_To
Carrier: .Test_Model.ArmGen__Prt_Lower
Component: 'Gripper' has 18 Sub-Components

RCC Load Tube

Name: .Test_Model.Load
Data: .Test_Model.Data_Load
Info:
D_UDE 'Data_Load' has '5' Sub-Components
Reference: .Test_Model.ground.Ref_Load
Component: 'Load' has 5 Sub-Components

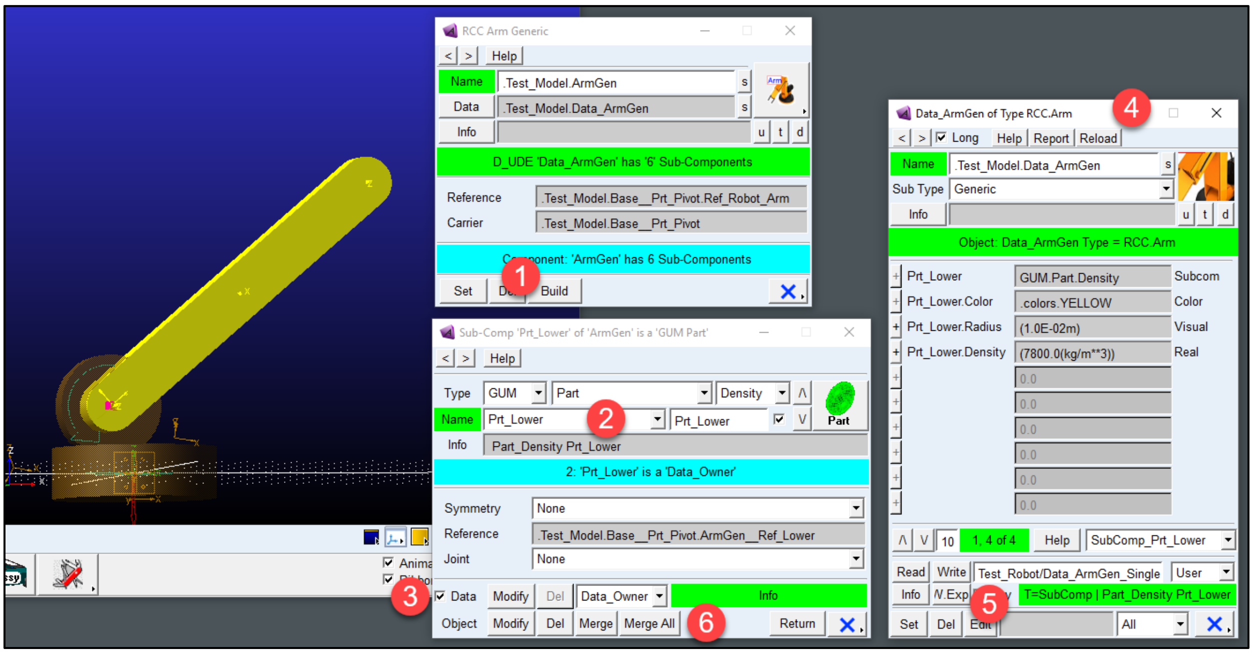
Cont_Load of Type GUM Contact...

Name: .Test_Model.Cont_Load
Data: .Test_Model.Data_Contacts
Info:
D_UDE 'Data_Contacts' has '0' Sub-Components
Ude One: .Test_Model.Gripper
Group One: Grasp
Range One: All
Ude Two: .Test_Model.Load
Group Two: Tube_Shape
Range Two: All
Align Seg: All
Measures: Sum_of_Fm
Component: 'Cont_Load' has 0 Sub-Components

- In case parameters *Range_[xxx]* and *Align_Seg* are set to values not equal to 'All', more advanced range filters and alignment options are applied in GUM contact components. The prime reason for these parameters is to reduce the number of contacts defined in case of many contacting objects.

## 6: Generic versus Specific Components

In the example listed, the components are of sub-type *Generic*. This sub-type is reserved for components which are defined using Sub-Components only instead of using a hard-coded topology macro. Thus, the macro *Arm_Generic.mac_cre* in Toolkit *RCC* is an empty macro, as objects in the Generic Robot Arm are generated from the *Build* method in the creation dialog of object *ArmGen*.

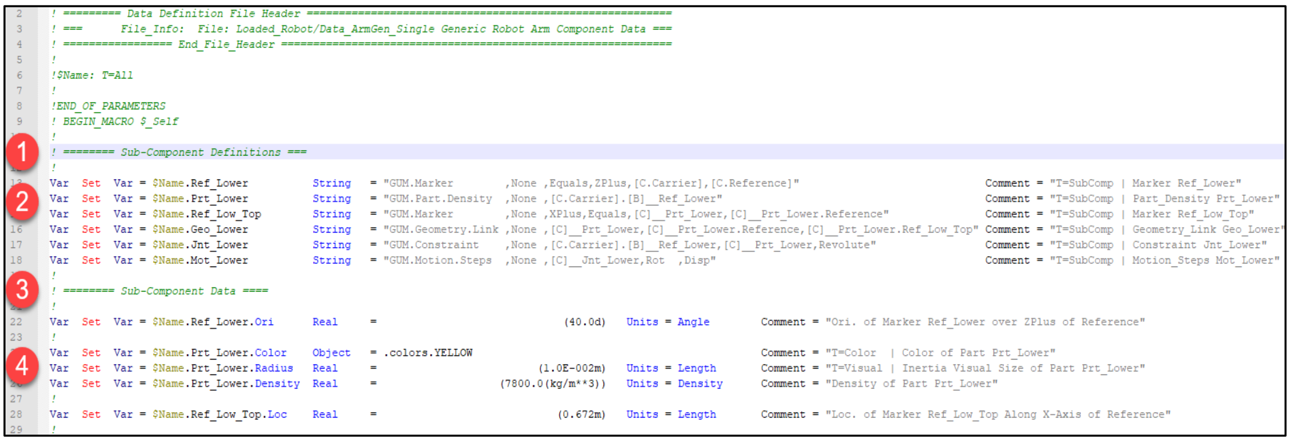


RCC Arm Generic
Name: Test_Model.ArmGen
Data: Test_Model.Data_ArmGen
D_UDE 'Data_ArmGen' has '6' Sub-Components
Reference: Test_Model.Base__Prt_Pivot.Ref_Robot_Arm
Carrier: Test_Model.Base__Prt_Pivot
Component: 'ArmGen' has 6 Sub-Components
Set
Build
1
Sub-Comp 'Prt_Lower' of 'ArmGen' is a 'GUM Part'
Type: GUM | Part | Density
Name: Prt_Lower | Prt_Lower
Info: Part_Density Prt_Lower
2: 'Prt_Lower' is a 'Data_Owner'
Symmetry: None
Reference: Test_Model.Base__Prt_Pivot.ArmGen__Ref_Lower
Joint: None
Data | Modify | Del | Data_Owner | Info
Object | Modify | Del | Merge | Merge All | Return
2
3
6
Data_ArmGen of Type RCC.Arm
Long | Help | Report | Reload
Name: Test_Model.Data_ArmGen
Sub Type: Generic
Object: Data_ArmGen Type = RCC.Arm
Prt_Lower | GUM.Part.Density | Subcom
Prt_Lower.Color | .colors.YELLOW | Color
Prt_Lower.Radius | (1.0E-02m) | Visual
Prt_Lower.Density | (7800.0(kg/m**3)) | Real
10 | 1, 4 of 4 | Help | SubComp_Prt_Lower
Read | Write | Test_Robot/Data_ArmGen_Single | User
Info | T=SubComp | Part_Density Prt_Lower
Set | Del | Edit | All
4
5

The image above illustrates the steps in editing sub-component objects for the Generic Arm: (1) Click on *Build* to (2) open the sub-component manager dialog, (3) select the Data Toggle button to (4) show the Sub-Component data dialog. This data dialog shows the specific design variables of the current sub-component type and sub-type.

The image shows that we are working on the Part Sub-Component called *Prt_Lower.* As shown, *Prt_Lower* is a sub-component of type *Part* and sub-type *Density* defined in toolkit GUM. In this density based part sub-component, mass and inertia are defined by its volume and density.

At any time in defining sub-components, the component data can be stored in a data file. In this case, the name used for this file is (5): *[User]/_Data/Test_Robot_Data/Data_ArmGen_Single.Arm_Generic*.

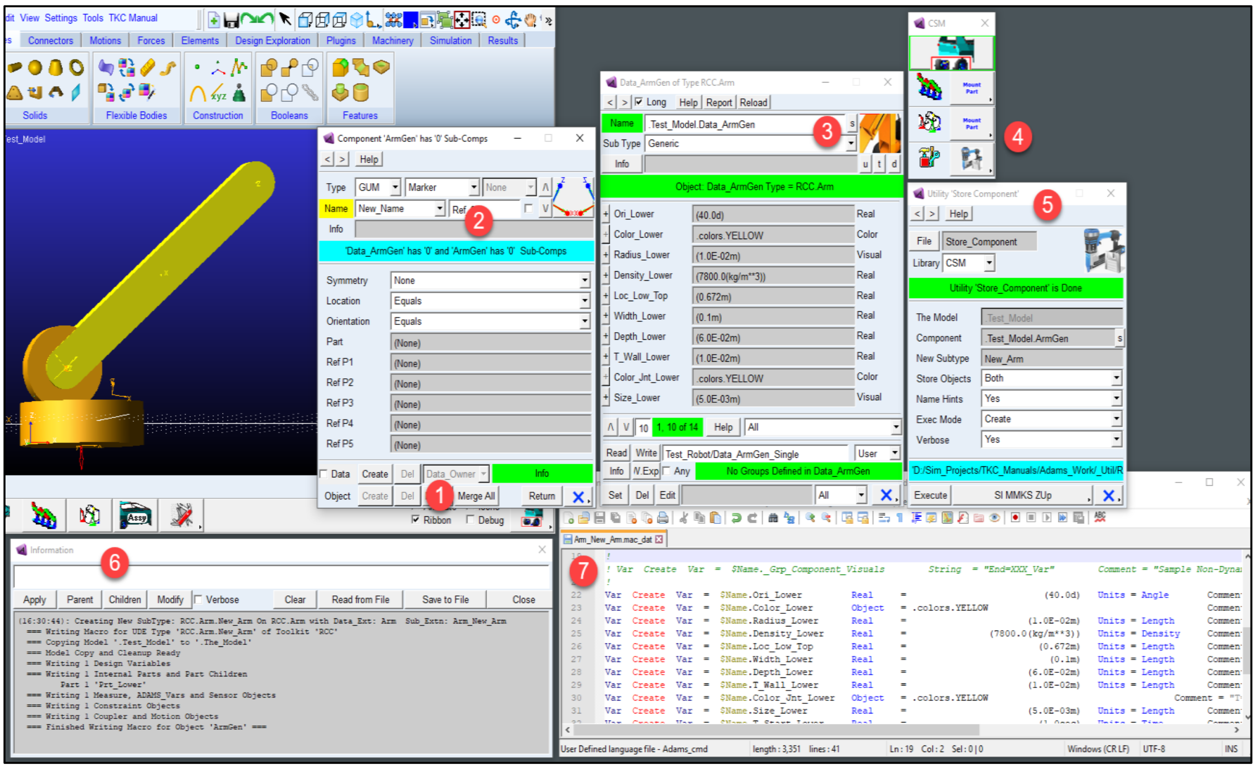


```
! ========= Data Definition File Header ==============================================
! ===       File_Info:  File: Loaded_Robot/Data_ArmGen_Single Generic Robot Arm Component Data ===
! ================= End_File_Header ==================================================
!
!$Name: T=All
!
!END_OF_PARAMETERS
! BEGIN_MACRO $_Self
!
! ======== Sub-Component Definitions ===
!
Var  Set  Var = $Name.Ref_Lower         String   = "GUM.Marker        ,None ,Equals,ZPlus,[C.Carrier],[C.Reference]"                    Comment = "T=SubComp | Marker Ref_Lower"
Var  Set  Var = $Name.Prt_Lower         String   = "GUM.Part.Density  ,None ,[C.Carrier].[B]__Ref_Lower"                             Comment = "T=SubComp | Part_Density Prt_Lower"
Var  Set  Var = $Name.Ref_Low_Top       String   = "GUM.Marker        ,None ,XPlus,Equals,[C]__Prt_Lower,[C]__Prt_Lower.Reference"       Comment = "T=SubComp | Marker Ref_Low_Top"
Var  Set  Var = $Name.Geo_Lower         String   = "GUM.Geometry.Link ,None ,[C]__Prt_Lower,[C]__Prt_Lower.Reference,[C]__Prt_Lower.Ref_Low_Top" Comment = "T=SubComp | Geometry_Link Geo_Lower"
Var  Set  Var = $Name.Jnt_Lower         String   = "GUM.Constraint    ,None ,[C.Carrier].[B]__Ref_Lower,[C]__Prt_Lower,Revolute"         Comment = "T=SubComp | Constraint Jnt_Lower"
Var  Set  Var = $Name.Mot_Lower         String   = "GUM.Motion.Steps  ,None ,[C]__Jnt_Lower,Rot  ,Disp"                              Comment = "T=SubComp | Motion_Steps Mot_Lower"
!
! ======== Sub-Component Data ====
!
Var  Set  Var = $Name.Ref_Lower.Ori     Real     =                              (40.0d)   Units = Angle      Comment = "Ori. of Marker Ref_Lower over ZPlus of Reference"
!
Var  Set  Var = $Name.Prt_Lower.Color   Object   = .colors.YELLOW                                              Comment = "T=Color  | Color of Part Prt_Lower"
Var  Set  Var = $Name.Prt_Lower.Radius  Real     =                           (1.0E-002m)   Units = Length     Comment = "T=Visual | Inertia Visual Size of Part Prt_Lower"
Var  Set  Var = $Name.Prt_Lower.Density Real     =                       (7800.0(kg/m**3))  Units = Density    Comment = "Density of Part Prt_Lower"
!
Var  Set  Var = $Name.Ref_Low_Top.Loc   Real     =                              (0.672m)   Units = Length     Comment = "Loc. of Marker Ref_Low_Top Along X-Axis of Reference"
!
```

The image above shows the top section of this data file with: (1) the definition lines of the sub-components, (2) one of these sub-components is part object *Prt_Lower*. The data (3) of the sub-components is stored further down in the data file where (4) illustrate the specific graphics and inertia data for *Prt_Lower*

Summarizing, a *generic* component sub-type is defined with sub-components for which both topology and data are stored in the component instance data file only. On the other hand, both topology and default data values of a *specific* sub-type of a component are defined in the *.mac_cre and *.mac_dat files of the Toolkit.

Using the *Merge All* button on the sub-component builder dialog, a *generic* component can be converted completely to a *specific* component.  Note that this conversion must be followed by a save operation of component topology and data to ensure storage as specific objects in the toolkit code of a new sub-type of this component.



| | | |
|---|---|---|
| Ori_Lower | (40.0d) | Real |
| Color_Lower | .colors.YELLOW | Color |
| Radius_Lower | (1.0E-02m) | Visual |
| Density_Lower | (7800.0(kg/m**3)) | Real |
| Loc_Low_Top | (0.672m) | Real |
| Width_Lower | (0.1m) | Real |
| Depth_Lower | (6.0E-02m) | Real |
| T_Wall_Lower | (1.0E-02m) | Real |
| Color_Jnt_Lower | .colors.YELLOW | Color |
| Size_Lower | (5.0E-03m) | Visual |

| | |
|---|---|
| The Model | Test_Model |
| Component | Test_Model.ArmGen |
| New Subtype | New_Arm |
| Store Objects | Both |
| Name Hints | Yes |
| Exec Mode | Create |
| Verbose | Yes |

```
(16:30:44): Creating New SubType: RCC.Arm.New_Arm On RCC.Arm with Data_Ext: Arm  Sub_Extn: Arm_New_Arm
 === Writing Macro for UDE Type 'RCC.Arm.New_Arm' of Toolkit 'RCC'
 === Copying Model '.Test_Model' to '.The_Model'
 === Model Copy and Cleanup Ready
 === Writing 1 Design Variables
 === Writing 1 Internal Parts and Part Children
        Part 1 'Prt_Lower'
 === Writing 1 Measure, ADAMS_Vars and Sensor Objects
 === Writing 1 Constraint Objects
 === Writing 1 Coupler and Motion Objects
 === Finished Writing Macro for Object 'ArmGen' ===
```

```
!
! Var  Create  Var  =  $Name._Grp_Component_Visuals      String  = "End=XXX_Var"       Comment = "Sample Non-Dyna
!
Var  Create  Var  =  $Name.Ori_Lower         Real    =                         (40.0d)   Units = Angle      Commen
Var  Create  Var  =  $Name.Color_Lower       Object  = .colors.YELLOW                                       Commen
Var  Create  Var  =  $Name.Radius_Lower      Real    =                      (1.0E-02m)   Units = Length     Commen
Var  Create  Var  =  $Name.Density_Lower     Real    =               (7800.0(kg/m**3))   Units = Density    Commen
Var  Create  Var  =  $Name.Loc_Low_Top       Real    =                        (0.672m)   Units = Length     Commen
Var  Create  Var  =  $Name.Width_Lower       Real    =                          (0.1m)   Units = Length     Commen
Var  Create  Var  =  $Name.Depth_Lower       Real    =                      (6.0E-02m)   Units = Length     Commen
Var  Create  Var  =  $Name.T_Wall_Lower      Real    =                      (1.0E-02m)   Units = Length     Commen
Var  Create  Var  =  $Name.Color_Jnt_Lower   Object  = .colors.YELLOW                                 Comment = "T
Var  Create  Var  =  $Name.Size_Lower        Real    =                      (5.0E-03m)   Units = Length     Commen
```

The image above shows the complete process of converting and storage:

1. Click the *Merge All* Button on the sub-component manager dialog,
2. After the merge process all sub-components from *ArmGen* are merged into specific code,
3. The data of the component is renamed and transferred to specific variables,
4. To store the component definition in the RCC toolkit, use Utility *CSM.Store_Component*,
5. In the dialog fields of the store component utility, select component name *ArmGen* and use *New_Arm* as the new type name for the sub-component,
6. The different topology and data objects stored in toolkit files are echoed to the Info dialog,
7. The data definition of the specific component is stored in *[RCC]/Arm_New_Arm.mac_dat*. Note the resemblance with the data objects in the data dialog for *Data_ArmGen*.

Note 1: A reasonably intuitive algorithm is used to rename the data objects of the component. As shown, *Data_ArmGen.Prt_Lower.Density* is renamed to *Data_ArmGen.Density_Lower*.

Note 2: Users can rename data and topology objects before using the *CSM.Store_Component* Utility. This allows for complete freedom in naming conventions and structure definition.

Note 3: Users must be aware of the fact that after storing data and topology of *ArmGen* as a *New_Arm* subtype the registry of the *RCC* toolkit has also been updated. The file *[RCC]/_RCC.cfg* toolkit registry file must be stored as well to reflect this and ensure correct reloading of the toolkit from the macro files.

## 7: Perform Simulation and Summary

This completes the section to explain a more advanced method for loading a TKC Assembly. In this section, the loading process was split into different sections, allowing a full model Assembly to be partially loaded and objects in the Assembly to be further expanded and extended before creating the final complete Assembly. In all of these stages, measures can be displayed of the model as defined up to that moment and simulations can be performed to check the correctness of interactions between the model components defined in the model.