



## Utilities for Analysing the Content of JT Files



# USER GUIDE

Document Revision: 2.0

Issued: 02/08/2018

## Contents

Overview of JT .....	3
Working with JT files.....	3
The JT File Architecture .....	3
The JT File System Representation.....	4
Accessing Individual Part Bodies within a Single JT Part File.....	5
Accessing Layer Filters within a Single JT Part File .....	6
JT Layer Filter Specification .....	6
JT Layer Specification .....	6
Example JT File Data Structures .....	6
Determining the Content of JT Data.....	10
JT File Analysis Utilities.....	10
Command Line Syntax for jt_content.cmd.....	11
JT File Layer Filter Analysis .....	11
Command Line Syntax for jt_layers.cmd.....	11

## Overview of JT

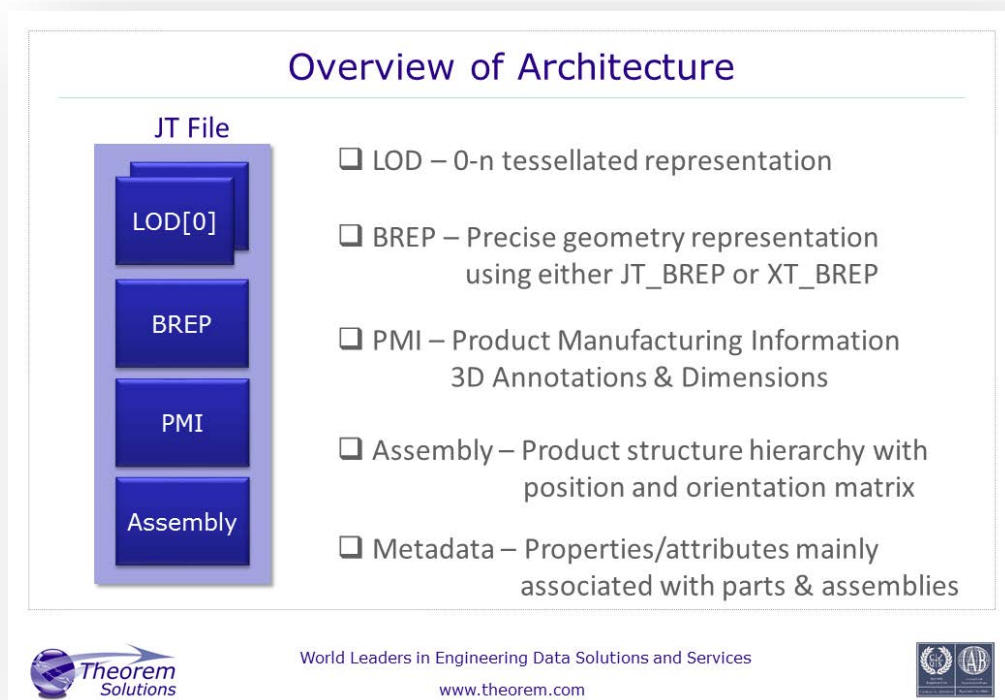
### Working with JT files

The Siemens JT file format is now an ISO standard ISO 14306:2012 and has become widely used in both Aerospace and Automotive companies for sharing 3D engineering data between OEM's and their supply chain. The JT data model allows supports the definition of both tessellated and precise BREP geometry representations. Together with assembly structure, 3D dimensions & annotations as well as non-graphical properties attached to either individual parts, sub-assemblies or the overall top level assembly structure object. Within the world of JT the 3D annotations and dimensions are also known as PMI (Product Manufacturing Information)



### The JT File Architecture

For the tessellated data there will always be one tessellated level of detail (LOD) however there may be more. Each of the LOD's are indexed from LOD(0) being the finest level of tessellated representation to LOD(n) being the least tessellated representation.



With regard to the precise BREP definition, the JT file supports two different definitions for precise geometry. These are:

**JT\_BREP** – The older JT precise geometry representation which is no longer preferred. However there are a number of files still in existence that contain JT data of this format.

**XT\_BREP** – This is the latest JT precise geometry representation. It is based upon the Siemens PARASOLID definition and is the preferred format for precise geometry today.

In addition to the combination of tessellated and precise geometry the JT files support the capability of representing the geometry using the Ultra Lightweight Precise (ULP) format. JT files that are defined with ULP data will not have either LOD or BREP geometry in addition to the ULP representation. The ULP based JT files will be smaller in comparison to the ones with LOD and BREP data. However the geometry representation uses a mathematical representation which closely represents the shape but not as precisely as the conventional BREP formats.

### The JT File System Representation

The JT data can be delivered in a variety of physical file representations each capable of representing the complete JT data structure. These include the following file representations:

**MONOLITHIC** – This is a single JT file that contains the complete JT dataset. Therefore the completed assembly structure as well as all of the geometry for any subordinate parts etc. is delivered within a single file.


**PER\_PART** – The JT PER\_PART representation uses a single JT file to define the complete assembly structure. In addition a folder of the same name as the top level assembly name is also provided. A separate JT file for each of the assembly's subordinate parts is located in the folder. Assembly level PMI is defined within the top level JT assembly file. While part level PMI is stored within the individual JT component files.

**FULL\_SHATTER** – The FULL\_SHATTER representation uses individual JT files for every level of the assembly structure as well as every individual subordinate component file. This is the most flexible representation as the JT data can be directly accessed from any level within the overall assembly structure. However when receiving data from external organizations it's important to know which of the collection of JT files provided is actually the true root node of the overall assembly.

**PLMXML + JT** – Companies that use the Siemens Teamcenter application to manage their CAD data will often output their data using a combination of PLMXML for the assembly structure with a collection of JT files, one for each component part of the assembly.

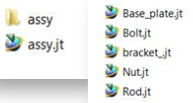
STEP AP242 BOM + JT – The use of STEP AP242 BOM (XML format) + JT is gaining in popularity. The file system representation is similar to that of PLMXML + JT. However the use of STEP AP242 BOM (XML) rather than PLMXML ensures that the data that is being shared conforms to a known ISO standard rather than using PLMXML which is very much implementation format specific.

## File System Representations




assy.jt

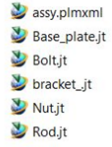
☐ **MONOLITHIC** – a single JT file containing all JT assembly/geometry PMI etc. data




☐ **PER\_PART** – a single JT file for the assembly + a separate JT file for each component part. The component part files are located in a folder of the same name as the assembly file.



☐ **FULL\_SHATTER** – a separate JT file for each level of the assembly hierarchy + a JT file for each component part. All of the data is held in the same folder




☐ **PLMXML** – a PLMXML file for the assembly + a separate JT file for each component parts. All of the data is held in the same folder



World Leaders in Engineering Data Solutions and Services

[www.theorem.com](http://www.theorem.com)



## Accessing Individual Part Bodies within a Single JT Part File

For each subordinate JT component part the standard operation for accessing the geometry is based upon a single item selection. Therefore irrespective of how many independent geometry bodies are in the part they have a single selection capability. For parts that contain multiple separate bodies its possible for the JT data to have been created whereby each of the separate bodies is individually selectable. This is achieved by defining the JT data with an additional attribute (SUBNODE) being applied.

When reading data of this type if you ignore the context of the SUBNODE property the part will have a similar behaviour to an assembly with each of the part bodies being read as an individual part. However when processing the data using the SUBNODE context the bodies will be collated into an individual part representation equivalent to the original source representation.

### Accessing Layer Filters within a Single JT Part File

The JT file format supports the use of JT Layer Filters to logically group different geometry elements into named referenced groupings (Layer Filters). This is used by a number of CAD applications to replicate groupings of geometry similar to those found within the CAD system itself. As well as the named references there is also a reserved Layer Filter named Default which can be used to select geometry associated to the default view of a parts geometry.

### JT Layer Filter Specification

A JT Layer Filter is allocated a JT Reference ID number as well as the Layer Filter reference Name. In addition it will reference one or more JT Layers by their individual JT Reference ID Numbers.

### JT Layer Specification

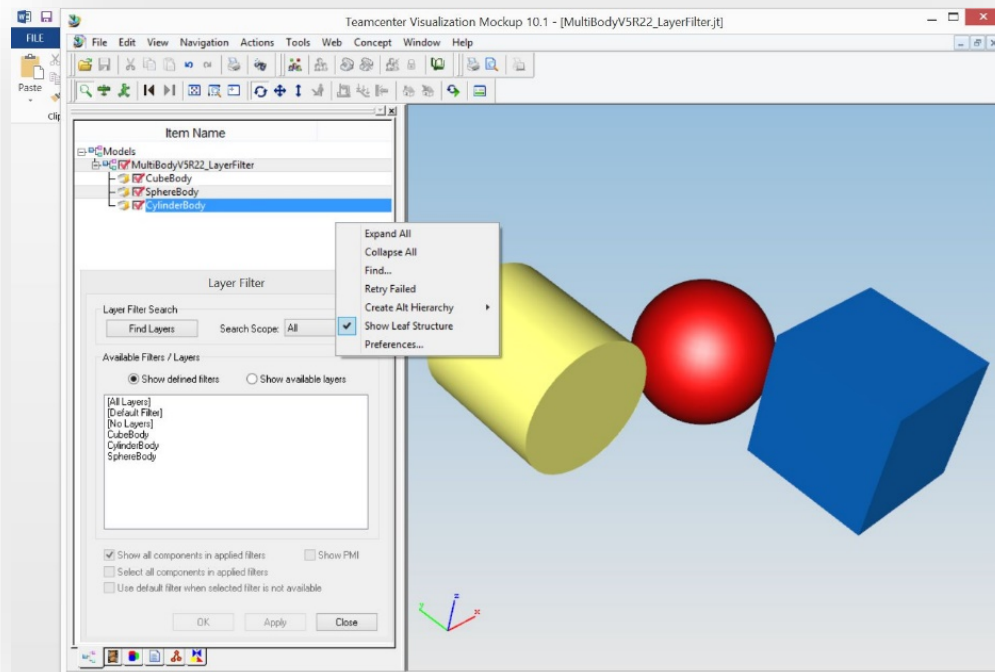
A JT Layer is allocated a JT Reference ID Number. Associated to each JT Layer will be a JT Part and/or individual JT PMI elements. There are occasions when the JT Layer contains no geometry at all. These apparently empty JT Layers may be referenced by individual JT Layer Filters.

### Example JT File Data Structures

The example JT file shown in the picture below is derived from a single CAD part file. However, within the CAD part there were three separate bodies each uniquely named SphereBody, CubeBody & CylinderBody.

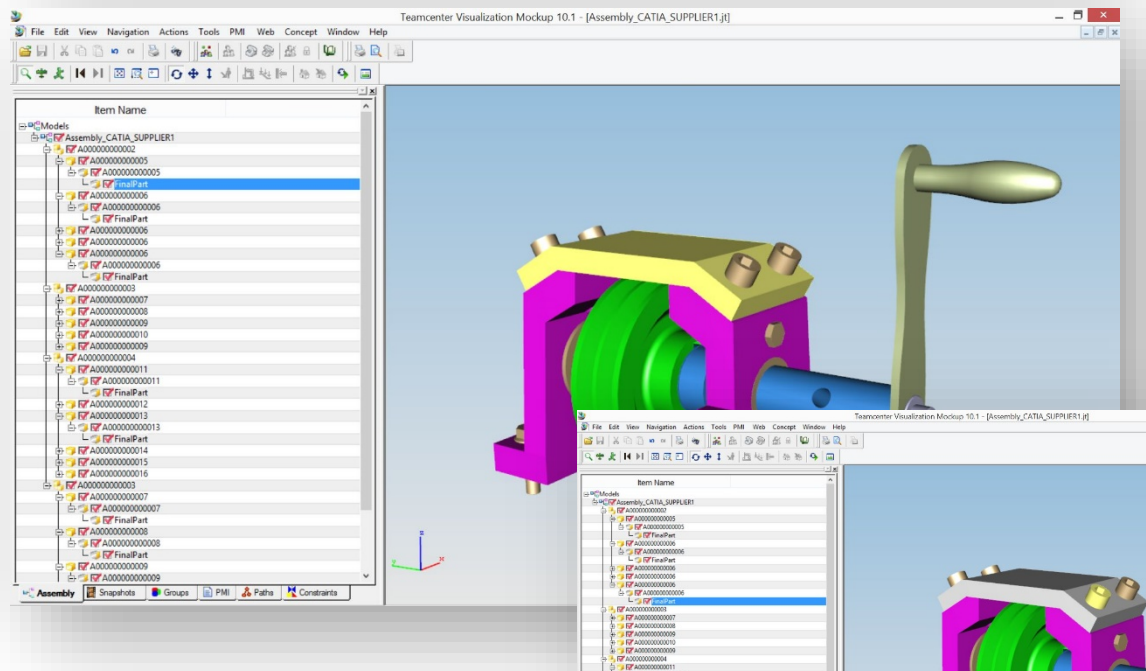
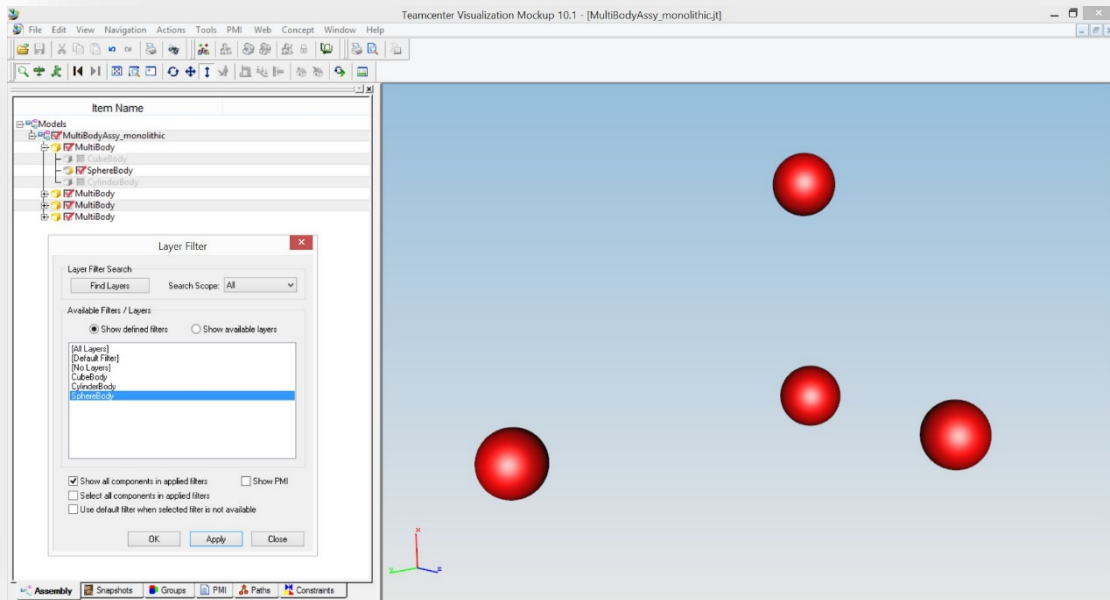
When this file was processed to JT each of the body containers was processed using the SUBNODE option. This created a pseudo JT Part for each of the body containers. This subpart breakdown of the JT data is normally suppressed. However, the JT user can explore this

subpart data by using the “Show Leaf Structure” with the Siemens JT applications. In the example shown there was a JT Layer Filter also created with the same name as the SUBNODE pseudo JT Part. Therefore the JT user could use the JT Action>Layer Filter mechanism as an alternate interface to the Show Leaf Structure to examine the subpart data.





In this next example the same part has been instanced a number of times into a small assembly. Then the Layer Filter interface has been used to change the display to show only one of the subpart bodies. You'll see that although the parts are uniquely instanced the SUBNODE names are common. Therefore selecting one of the Layer Filters will control the display of all instances.



In the final display you'll see an example where the SUBNODE reference names are identical, "FinalPart", and yet the geometry is totally different. This is not uncommon in JT data structures especially when the JT output is being created by organizations that have a process driven focus. In this case the pseudo parts are also equally named and yet the



resultant geometry is totally different. This is something that needs to be taken into account when analysing JT data.

## Determining the Content of JT Data

Due to the fact that there are these multiple methods that could have been employed to author the JT data if you receive JT files from an external source it can be very difficult to determine the actual content of the data you've received. Even when you use a visualization tool, such as the Siemens freely available JT2GO application you cannot always be sure of the detailed technical content of the file you've received.

Therefore Theorem Solutions have made available the following routines to analyse the content of JT files. Once the content has been verified then the most appropriate translation strategy can be determined.

## JT File Analysis Utilities

A number of utilities are provided to enable you to analyse the content of a JT file in batch mode via a command line interface. These utilities are developed using the JT Open Toolkit Libraries and report the analysis of the JT file direct to the screen. Therefore when implementing these utilities into more complex workflows you simply need to redirect the output to a temporary file to analyse the data further if you want to perform conditional actions based on the content of the JT file.

The utilities are all delivered in the folder named **bin** with the translator installation -  
**%TS\_INST%\bin**

### JT File Content Analysis

The utility **jt\_content.cmd** output an overview listing of the contents of the JT file to the screen in a single line of output with 4 values each separated by a semicolon. The values are;

**ASSEMBLY=Yes or No** – Note a JT part file with SUBNODE data would return a value of Yes for this property

**BREP=No, XT\_BREP or JT\_BREP** depending upon the datatype of any precise representation stored in the file

**TESS=Yes or No** – Indicating if the JT file contained any LOD data or not

**PMI=Yes or No** – identifying if the JT file contains any 3D Dimensions or Annotations

## Command Line Syntax for jt\_content.cmd

- **%TS\_INST%\bin\jt\_content.cmd <input\_JT\_filename>**

```
C:\Users\trevor>C:\Theorem\CAD_19.3_MC5JT_WIN.01\bin\jt_content.cmd C:\Users\trevor\Documents\demo_local\CatiaU5\simple_parts\MultiBodyU5R22_LayerFilter.jt
ASSEMBLY=Yes;BREP=XT_BREP;TESS=Yes;PMI=No
C:\Users\trevor>
```

In this example the ASSEMBLY=Yes is shown even though in reality this example JT file is a component. The value of Yes is written due to the SUBNODE data structure of the file.

## JT File Layer Filter Analysis

The utility **jt\_layers.cmd** lists the breakdown of JT Layer Filters defined within the nominated JT file. The output is written to the screen.

## Command Line Syntax for jt\_layers.cmd

- **%TS\_INST%\bin\jt\_layers.cmd <input\_JT\_filename>**

```
C:\Users\trevor>
C:\Users\trevor>
C:\Users\trevor>C:\Theorem\CAD_19.3_MC5JT_WIN.01\bin\jt_layers.cmd C:\Users\trevor\Documents\demo_local\CatiaU5\simple_parts\MultiBodyU5R22_LayerFilter.jt
Layer 0 ID 1033
JtkEntity:: JtkPart : Name: CubeBody
Layer 1 ID 1034
JtkEntity:: JtkPart : Name: SphereBody
Layer 2 ID 1035
JtkEntity:: JtkPart : Name: CylinderBody
Found 3 Layer Filters
LayerFilter 0 Name: CubeBody
Layer 0 ID 1033
LayerFilter 1 Name: SphereBody
Layer 0 ID 1034
LayerFilter 2 Name: CylinderBody
Layer 0 ID 1035
C:\Users\trevor>
```

In this example output you can see that there are three JT Layers, sequence number 0,1 & 2 with the JT Layer Reference ID's 1033,1034 & 1035. This is followed by a count of all Layer Filters in the file, in this example there are 3. This is followed by each of the Layer Filters being defined. The first one has a LayerFilter sequence number of 0 and the Name=CubeBody. This Layer Filter contain just one Layer, ID 1033. The second LayerFilter has the sequence numbers of 1 and the Name=Spherebody. Again this LayerFilter only references a single Layer, ID#1034. Finally the third LayerFilter has the sequence ID#2 and the Name CylinderBody. It is made up of a single Layer, ID#1035