SQL_Database_Definition

The exporting of data from Visual3D requires an external database with a predefined data structure.

There are no restrictions as to which database system is used since connection to it is provided by ODBC connections. C-Motion’s sample code is provided using Visual Basic for Application in Microsoft’s Access database system.

The data structure is normalized to eliminate redundant data entry. The specific data values for signals, metrics, and analog data, however, are stored in a large single string that must be parsed prior to using the data for analysis purposes. This approach was necessary due to the inherent speed limitations a relational database has with storing tremendous amounts of data.

Note that some RDBMS systems may need add-on modules or extensions to support Transforms or Pivot Table capabilities. MS Access has the functionality built-in, so some example code provided by C-Motion may not transfer directly to other systems.)

The picture below illustrates the how the Visual3D signal and event tree structure maps to database tables. The Patient Info table is provided as a sample illustration on where the relationship to patient information may be managed. The BLOB table is where the raw data is stored after the exporting process is done, and a Raw Data table is provided for the parsed results. Sample database definitions and sample routines for parsing and managing data are provided by C-Motion.

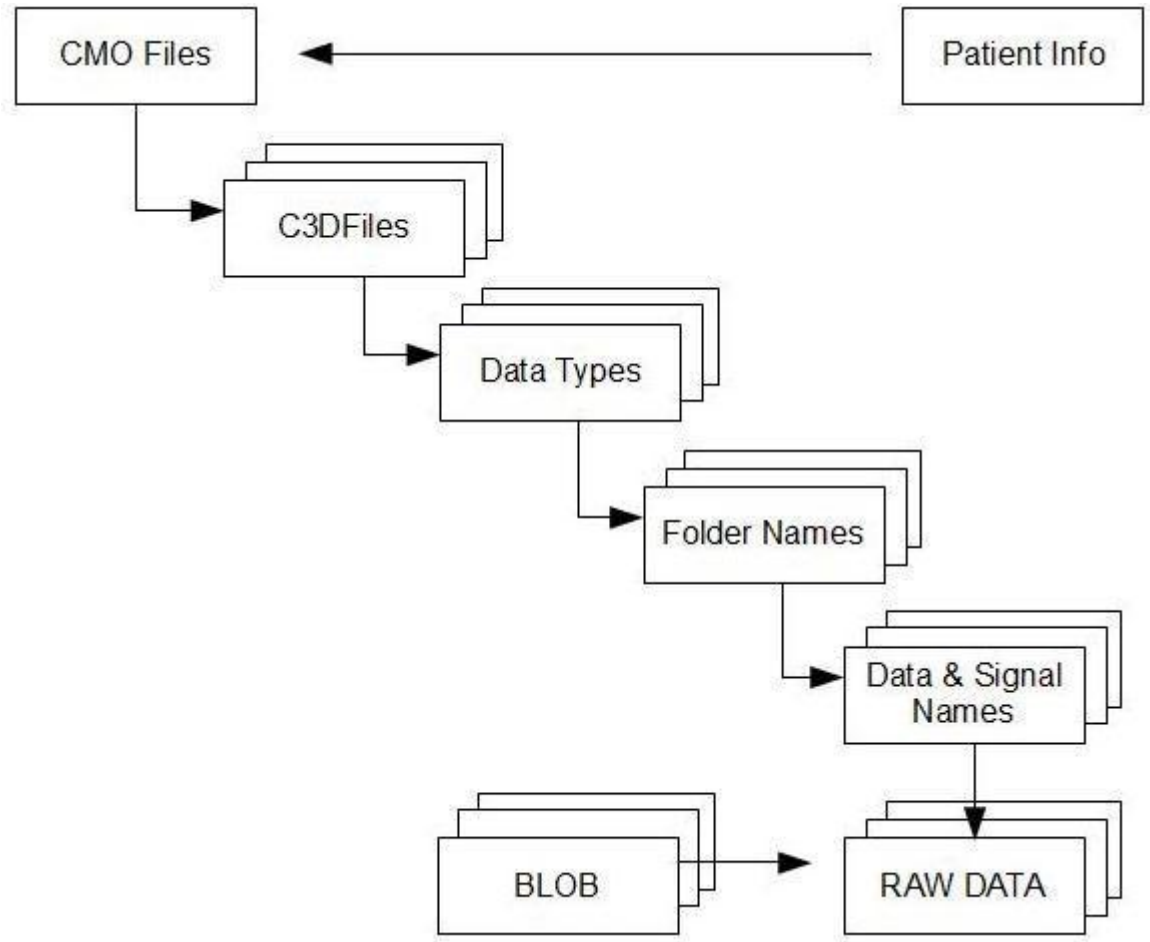

Table Structure Definition

The actual SQL database tables loosely follow the Visual3D data tree layout, and are defined below:

TABLE NAME	DESCRIPTION	COLUMNS
L0_cmo	Contains the CMO file names.	cmoID = Integer, auto-incrementing index cmoName = Text PatientID = link to a patient table (optional)
L1_c3dfile	The C3D files contained in each CMO file.	fileID = Integer, auto-incrementing index cmoID = link to L0_cmo (foreign key) fileName = Text
L2_type	The signal or data TYPE in each C3D file. (Examples are: TARGET, EVENT, ANALOG, LINK_MODEL_BASED, etc.)	typeID = Integer, auto-incrementing index fileID = link to L1_c3dfile type_Name = Text (beware the “type” or “typename” may be reserved words in some RDBMS programs)

Signal Example

L2_events	Events associated with a C3D file. It has event names, frame numbers, and timestamps. No tables link back to this one.	eventID = Integer, auto-incrementing index fileID = link to L1_c3dfile event_label = text event_time = float description = text frame = integer
L3_folder	The folder names holding signals for each type. (Example is ORIGINAL or PROCESSED.)	folderID = Integer, auto-incrementing index typeID = link to L2_type folderName = text
L4_name	Signal names in each folder, (but not the data)	dnameID = Integer, auto-incrementing index folderID = link to L3_folder dataName = text history = long text
L5_framedata	All the actual data (after parsing) for each L4_name. Each raw has a label and a value for each motion capture frame. For example, frame 1 may have 3 rows for X, Y, and Z values. That is why a pivot table is needed for analysis purposes.	frameID = Integer, auto-incrementing index dnameID = link to L4_name framenum = integer frameTime = float stepVal = integer (# dataValues per frame) label = text dataValue = float
L6_framedata	All the actualdata (before parsing) for each L4_name. One row per L4 row of consolidated data and labels stored as a BLOB (or large text string).	L6ID = Integer, auto-incrementing index dnameID = link to L4_name parsed = Boolean (optional yes/no indicator) bigdata = BLOB or long text
Patient (OPTIONAL)	A sample patient table for linking to CMO's. The fields in the samples are derived from old VCM data definitions.	VCM also has a Patient Codes cross-reference table.

For a target signal called "LANK" we would see a row in the database for the signal name in L4_name table with a system generated ID called dnameID and a dataname. For example: 849 "LANK"

In the L6_framedata table, we would then see the raw data associated with "LANK" represented as a row containing a system generated ID, the foreign key linking back to the L4_name table, and the data string. For example, for signal 849 (i.e. "LANK"), we would see 10 frames of data like this:

2847 849 ^1@X@0.106221@Y@0.292641@Z@1.214349@Residual@0.000000@Camera_
bits@0.000000^2@X@0.106105@Y@0.292334@Z@1.214423@Residual@0.000000@Camera_
bits@0.000000^3@X@0.106000@Y@0.292041@Z@1.214490@Residual@0.000000@Camera_
bits@0.000000^4@X@0.105904@Y@0.291758@Z@1.214550@Residual@0.000000@Camera_
bits@0.000000^5@X@0.105815@Y@0.291485@Z@1.214604@Residual@0.000000@Camera_
bits@0.000000^6@X@0.105729@Y@0.291216@Z@1.214653@Residual@0.000000@Camera_
bits@0.000000^7@X@0.105646@Y@0.290951@Z@1.214699@Residual@0.000000@Camera_
bits@0.000000^8@X@0.105563@Y@0.290686@Z@1.214741@Residual@0.000000@Camera_
bits@0.000000^9@X@0.105480@Y@0.290420@Z@1.214782@Residual@0.000000@Camera_
bits@0.000000^10@X@0.105395@Y@0.290150@Z@1.214822@Residual@0.000000@Camera_
bits@0.000000

This approach makes parsing the data fairly straightforward using the built-in parsing tools of today's programming languages. For example, In Visual Basic, using the SPLIT command and Direct Access Object (DAO) methods for managing databases, the code to parse L6_framedata and store the data into the L5_framedata table would look like:

```
Dim recset As DAO.Recordset
Dim frames() As String
Dim frameparts() As String
Dim SQLstr As String
Dim frametime As Double
Dim i As Integer
Dim x As Integer

    'Get the unparsed data signals
    Set recset = CurrentDb.OpenRecordset _
("Select * from l6_framedata where parsed = False;", dbOpenDynaset)
    Do While (Not recset.EOF)
        ' Parse out the frames for this signal - by the ^ delimiter
frames = Split(recset!bigdata, "^")
        For i = 1 To UBound(frames)
            ' Parse out the data for the frame, delimited by @
            ' where framepart(0) is always the frame number and then label-
value pairs follow
            frameparts = Split(frames(i), "@")
        ' Store each signal component
```

Sample SQL scripts to create this table structure are available for SQL Server, MySQL, and MS Access.

Parsing Raw Data

The data exported from visual3D into a database is too complex and dynamic for a rapid database transfer. The overhead associated with SQL Insert commands is quite high, even with optimization. For that reason, the data is formatted into a delimited structure to make intelligent post-process parsing possible. For each signal, a database row is created consisting of the signal identifier (i.e. foreign key) and a big string of raw data. The raw data - whether target data per frame, analog data, or calculated metrics – is retrieved and parsed into the following structure:

```
            For x = 1 To UBound(frameparts) Step 2
                SQLstr = "INSERT into l5_framedata “
         SQLstr = SQLstr & " (dnameID, framenum, label, dataValue) VALUES "
                SQLstr = SQLstr & "(" & recset!dnameID & ", " &
frameparts(0)
         SQLstr = SQLstr & ", '" & frameparts(x) & "', " & frameparts(x+1)
         SQLstr = SQLstr & ");"
                DoCmd.RunSQL SQLstr
            Next
        Next
        recset.MoveNext
    Loop
```

Note that this routine could potentially take a very long time to run since a SQL INSERT is performed for every component of every frame of data.

However, once parsed and inserted into the L5_framedata table, it is very easy and fast to access the data for analysis. This is where pivot tables play an important role. They permit the dynamic storage of data components to be managed since it is impossible to accommodate a data structure for every potential digital and analog motion capture related signal generator.

However, pivot tables can be avoided if only certain information, in a consistant format is needed, then new tables can be created to improve the speed of the parsing process.