The Optimization Of Snapshot Files While Using The Track Drawing Changes Functionality In NX

[1]P.L.Himabindu and [2]Prof Varshapriya JN

[1]Dept of Computer Engineering, VJTI College, Mumbai, India
[2]Dept of Computer Engineering, VJTI College, Mumbai, India

Email: Plhimabindu92@gmail.com and varshapriyajn@vjti.org.in

***Abstract*— In the current scenario the design of machines is becoming very complex. So, identifying any changes that are made during the various revisions of the designing process is very difficult. To track such changes in drawing the track drawing changes functionality is developed in NX. This functionality helps us to compare different part revisions and track basic, reference and dimension changes with other revisions of same part. While tracking the drawing changes a snapshot of the file is created. As these snapshots are huge in size, so it consumes a lot of space on hard disk to store the data. These snapshots include the part file data along with the additional metadata such as version, encoding, modified time and date. So, it is important for this metadata to be compressed while storing the snapshot as increased storage space in the system is causing performance issues. In this paper we will look at the ways in which the snapshot file can be compressed in track drawing changes functionality so as to improve the overall performance.**

Keywords: Snapshot, QAF, Active drawing

I. INTRODUCTION

The Track Drawing Changes functionality is developed in NX to save snapshot data, generate comparison reports, and display change symbols on your drawing. Track drawing changes will enable user to compare the revisions of the drawing against the current snapshot of the file or any other previously saved file. After comparing the files, it will show the user the changes made in the file and will also provide an option whether to allow each individual change or to reject the changes. The snapshot data is used to capture and store key drafting data for each drawing sheet in the current work part. This data is used to generate a comparison report of the differences in the drafting data that results when the part is changed or updated.

In order to perform the track drawing changes functionality a snapshot of the file is saved in the system. This snapshot includes the drawing along with some additional headers such as last modified time, encoding, path etc. As these snapshots are sometimes in GBs in size, so it consumes a lot of space on hard disk to store the data. Hence, it is important for this data to be compressed while storing the snapshot. Compressing the data is essential because of the amount of data that is generated and also to improve the performance efficiency.

II. TRACK DRAWING CHANGE FUNCTIONALITY

The Track Drawing Changes tool in NX Drafting allows user to quickly and easily compare revisions of a drawing to see exactly what changes have been made. Track Drawing Changes is a tool user use to compare an active drawing against a previously saved snap shot of the drawing. The compare function provides the capability to see and document what has changed since the snapshot was last saved, usually when the part was last saved. The Track Drawing Changes command tracks many drafting objects including sheets, views, dimensions, annotations, symbols, centerlines, tables, and drafting sketch curves.

The workflow for using this command would be that a user opens a drawing file as a baseline to compare and then the user selects this command from Track Drawing Changes toolbar. If the snapshot data already exists then the comparison report can be generated directly. In case the snapshot does not exist for the part file then an option is given to the user whether to create a snapshot or not. If user selects yes, then the system will create the snapshot (Key Point Information) data and then the comparison is done. If user selects no, then again, the 'Drawing Compare' dialog will open to let user select another part file.

The track drawing changes command provides two options using which this functionality can be performed. The first option is to compare against a saved snapshot, this option compares the current drawing against snapshot data captured of the same drawing. This method allows the user to determine when to save data. The Track Drawing Changes report function will then compare the current state of the part with the saved snapshot. This method is best used when comparing a previous saved version of the part to a current version. Another method is to compare against the last drawing file update. This method saves a temporary snapshot of the data between executions of the drawing compare report. The idea behind the latter method is to track changes while working on the drawing.

In this, we have provided functionality where user can create snapshot of drawing where we capture important information of annotation. This snapshot file is stored in a hierarchy of structured storages and streams. The information captured in a snapshot includes actual data and some additional overheads. The overheads involved include details such as creation time, encoding, location etc. These overheads sometime increase the file size drastically and cause performance issues. Once the snapshot data is created it is then converted to XML and finally saved in QAF folder. So now based on part file, XML size of snapshot stored in QAF may sometimes increase by around 50%. To overcome this, we will be using compression API which would drastically reduce the QAF size.

III. SNAPSHOT DATA

A snapshot is a point in time information of the data that is captured for comparison. Data that will be captured includes all notes, labels, symbols and dimensions. When you create the snapshot data, it is saved with the part's permanent data. Only one snapshot is saved with a part. The snapshot can be captured in different scenarios depending upon the mode the user is working on. There are different modes such as on demand mode and on save mode. In the on-demand mode, users will create a snapshot when they want. At any time later, they can create a report that will compare the current state of the drawing or part with the snapshot taken on demand. During the on save mode, users may want to compare different versions of the part. The user will open up the part in a new release and run the report. The report will compare the current state of the drawing or part with the snapshot taken when the part was last saved. In this manner, changes that may have occurred as a result of versioning up may be detected. Or the user may run the report right after updating the freshly opened part to see if update causes differences during the two versions. Lastly, the user may want to run the report after the days' work to see what differences have occurred between when the part was last saved and before he saves it. Of course, if the user created a snapshot during the session, it will overwrite the snapshot created when the part was last saved and thus the part will no longer have a snapshot to do version up comparison.

IV. QAF FOLDER

QAF stands for Quick Access Folder, so whenever a part is saved it is stored in the QAF folder. The QAF folder includes the data of the saved part along with the metadata. The part file is stored in QAF folder in binary format whereas the metadata is stored in XML format. The metadata is generated while saving the snapshot increasing the file size. Here the metadata includes fields such as version, encoding, location, creation time etc. This additional metadata added to the saved part file sometimes increases the overall file size by around 50% and hence causing the performance issues.

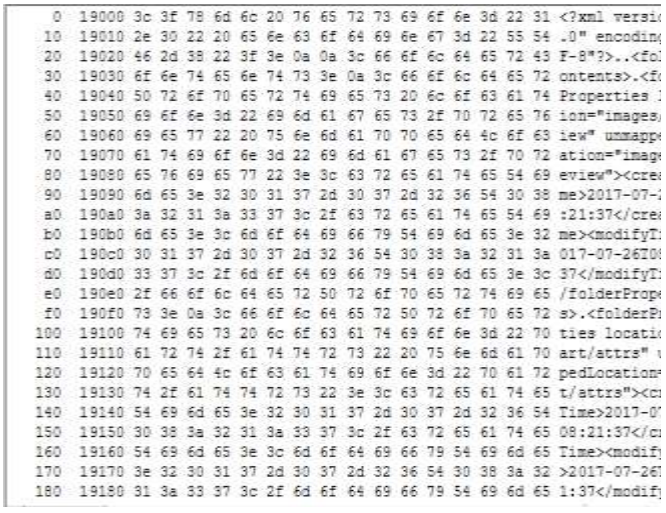

Fig. 1: QAF Folder Data

Figure 1 shows how the snapshot data is stored in the QAF folder. The data stored in the binary format on the left is the actual part file data whereas the meta data is stored on the right in the form of XML tags. The data present in the form of tags is the additional data that is stored while saving the snapshot.

V. WORKING OF TDC FUNCTIONALITY

Track drawing changes functionality allows user to compare current part with other revisions of same part. This helps us to improve product quality and user productivity by identifying the differences in the drawing. Any mismatch or changes in the drawing from the previous saved file can be easily identified. This helps us to easily identify the difference or any new enhancements in the drawing.

In this functionality, the user can select any part for comparison. Once part is selected based on user inputs, system will either read the snapshot data from QAF folder of selected part without opening part or open selected part to create snapshot in case it is not available. Once the actual part and the snapshot data are available then the comparison report is generated. When a comparison report is run, the results are displayed in an interactive dialog box. This report includes details of all the data that has been added, deleted or modified from the previously saved file. We can also view basic or detailed data, to visually identify and understand individual changes, additions, or deletions. The report generated helps us to track the new properties, highlight change symbols and navigate to symbols if any of the entity is deleted. The report generated also provides a chance for the user to determine whether to keep those changes or discard them. And once the change has been reviewed the state can be marked on the drawing.

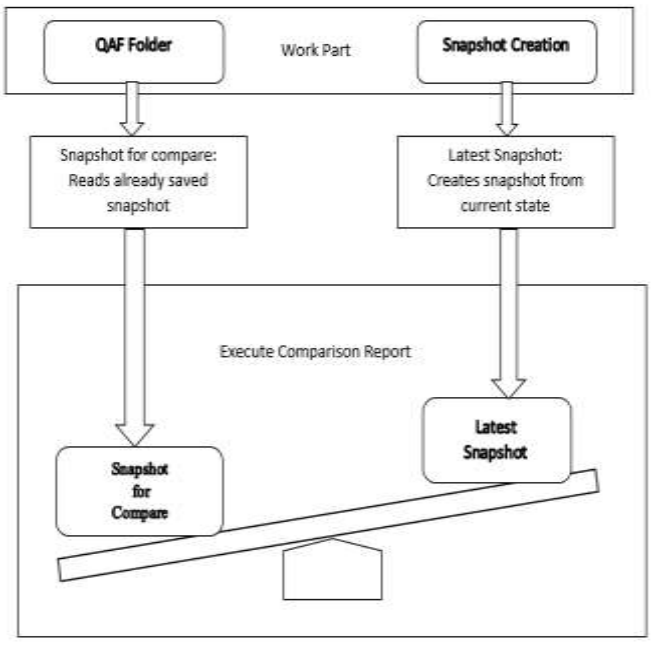

Fig. 2: Comparison Method

VI. Optimisation of snapshot file

Snapshot files are stored in binary and xml file format. The binary data is the actual part file that is stored which contains the information about the drawing and the xml data is the metadata that is added while saving the snapshot. As a part of optimization, the binary data in file will be kept same and the xml part of the file will be compressed. Here the XML part of the file will be compressed for data storage optimization. Different compression levels can be set by compromising between speed and compression. The compression levels can vary from 0 to 9. As the compression level goes up the time taken to compress also increases with the higher levels of compression being from 7 to 9. Depending upon the compression level there is a tradeoff between the compression ratio and performance. Here level one represents the fastest performance and level nine represents the best compression ratio. Most

VII. Algorithm for compression

1. Initially check if part file name is given or not.
2. Then open QAF, if it does not exist then it is created else existing contents will be cleared before writing.
3. Initialize the dataset to NULL for fields such as avail_in, avail_out, next_in and next_out.
4. Prepare compression stream for output and post error message on failure
    a) Obtain data length for compression i.e., length of uncompressed data + checksum length.
    b) Allocate memory to the length of compression stream using calloc function.
    c) Now call deflateInit function to initialize the internal stream state for compression.
    d) If deflateInit function returns an error then the compression stream cannot be setup.
5. Now we can start to compress the data block using deflate function
6. Do
    a) Compress until eof (avail_in>0)
    b) If deflate returns error value then compression failed.
    c) Else compression completed successfully
7. Now add checksum at the end of compression stream to ensure the integrity of the uncompressed data after decoding.
8. After the compression is finished the deflate function returns a compression value.

    If (value==Z_STREAM_END)

    The input is successfully flushed

    Else

    Failed attempt to compress data.

9. The compression stream is flushed by setting the values of compressed data to NULL and data bytes to 0 and also freeing the memory assigned to compressed data.
10. Then dynamically allocated data used by compression stream is cleaned up.
11. The compressed data is then written into the QAF folder successfully.

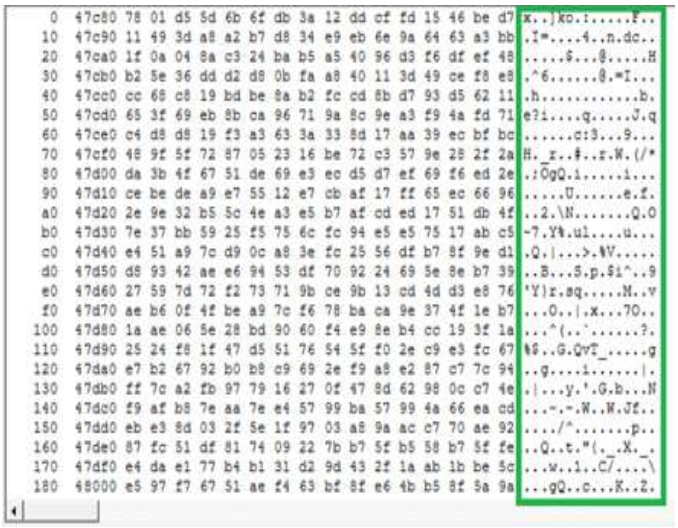

Fig. 3: Compressed QAF Data

Figure 3 shows how the XML data is compressed and is available in the QAF folder. As the size of additional metadata is very large increasing the overall size of the part file so, compression ensures that the size of part files is reduced and thus the storage requirements. Reducing the file size also helps us to open the file more quickly and improving the overall performance.

TABLE I. Part size reduction

Sr No.	Compression Data Fields			
Sr.No.	*Part Name*	*Uncompressed snapshot data size in KB*	*Uncompressed snapshot data size in KB*	*% reduction in snapshot size with compression*
1	NX755_2D_Centerline_HVP_001_dwg_001	6775	611	90.981
2	56000000135093_k2_s_560000001350 93_dr01	450	48	89.333
3	PR2217223_0302AAU00480N_006_1	5615	820	85.396

The table 1 shows the percentage reduction in the snapshot size after the compression algorithm is applied on the data file. For part NX755_2D_Centerline_HVP_001_dwg_001 the uncompressed snapshot file size is 6775KB after the compression algorithm is applied the file size is reduced to 611KB. The percentage reduction in the snapshot file can be given as 90.981% which is very high and can improve a lot of performance.

VIII. CONCLUSION

Implementation of this method allows snapshot data to be stored in a compressed format in the system and also enables the track drawing changes functionality to be more efficiently by the customer. This will also help in identifying the differences in the drawing and also report any issues at the earlier stage. Also optimizing the data that is generated during track drawing changes will help to improve the storage that is used for this functionality. Optimization of the data is essential because of the amount of data that is generated and also to improve the performance efficiency.

IX. REFERENCES

[1] P.Yellamma and Dr.Narasimham Challa, “Performance Analysis of Different Data Compression Techniques,” International Journal of Engineering Research & Technology. Vol. 1, October 2012

[2] https://community.plm.automation.siemens.com/t5/NX-Design-Knowledge-Base/Compare-Drawing-Revisions-Pt-1-Track-Drawing-Changes-in-NX-11/ta-p/403323.

[3] https://github.com/jtkukunas/zlib.

[4] DEFLATE Compressed Data Format Specification - RFC1951: http://www.faqs.org/rfcs/rfc1951.html

[5] James T Kukunas and Vinodh Gopal, “High Performance ZLIB Compression,” White Paper, April 2014