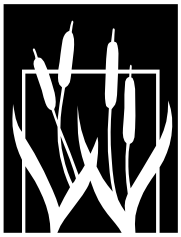

Whitemarsh
Information Systems Corporation

Whitemarsh Metabase Bill of Materials (BOM) and Single File Recursion (SFR) Users Guide

May 30, 2005

Whitemarsh Information Systems Corporation
2008 Althea Lane
Bowie, Maryland 20716
Tele: 301-249-1142
Email: mmgorman@wiscorp.com
Web: www.wiscorp.com

Table of Contents

List of Figures

List of Tables

1 Introduction

The purpose of the database application, Bill of Materials (BOM) and Single File Recursion (SFR) users guide is to provide a simple database application that illustrates the use of two very common data structures in the metabase.

2 Software Installation

BOM-SFR installation is different from the normal Metabase applications. It does not use an SQL DBMS. Just download the self-installing executable file, execute it, and put all the files into three directories: BOMSFR, Calendar Data, and XY Data. The execution of the application is simple: just double-click in the file, BOMSFR.exe.

3 Database Design

The tables associated with this application are in two classes: Bill of Materials, and Single File Recursion

- X
- X Structure
- X Structure Type
- Y

Figure 1 presents the overall database schematic.

- X is a table that represents the atomic parts in a bill of materials.
- X structure represents relationships that exist among parts
- X structure type represents a “typing” of a collection of related parts.
- Y is a table of parts that are related to each other through recursion.

Tables X, X Structure, and X Structure Type related to Bill of Materials, while table Y applies to Single File Recursion. An example of both bill-of-materials (BOM) and single-file-recursion(SFR) is presented in the next section.

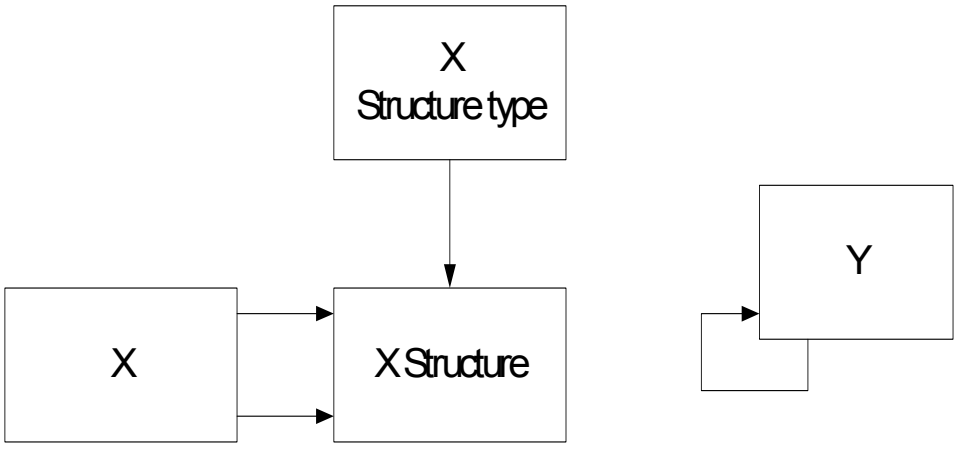

Figure 1. Bill of Materials and Single File Recursion data model.

There are two relationships between X and XS. One represents "contains" and the other represents "is contained in." Suppose there is a bicycle. It has a seat, frame, front fork, and for the purposes of this very simple example, two wheels. Each wheel has a tire, hub, and spokes. If a spoke is a part, then both the following are true:

- Wheel contains spokes
- Spoke is contained in a wheel.

It this case, Wheel and Spoke are both parts, that is, X. The realized relationship between the spoke and the wheel is XS (part structure), and the type of relationship is XST (part structure type). In this case, the type of relationship is containment (contains and is contained in). It might also be control (controls and is controlled by), Divide (divides into, and is divided from). In this bicycle example there is only one wheel part and one spoke part. If this were a manufacturing Bill of Materials, a column in XS would be quantity. But in this example, the columns in X, XS, and XST are quite simple. Table 1 enumerates the columns in each of the X and Y tables along with a brief definition for each.

Table	Columns	Brief Definition
X	Xid.	The identifier of the part
	Xmb.	The name of the part
XS	Xsid	The identifier of the assembly
	XSTid	The identifier of the assembly part type
	ActiveId	The identifier of the part in the active process
	PassiveId	The identifier of the part in the passive process
	Seq	The sequence number of this assembly in an assembly collection
	LevelNumber	The computed level of an assembly where the top level is “1"
	VirtualChildren	An indicator that this assembly represents parts that are employed elsewhere, first.
XST	XSTId	The identifier of the assembly part type
	XSTName	The name of the relationship that encompasses a specific subcollection of XS instances
Y	YId	The identifier of the part
	YSeq	The sequence of a part within a collection of parts that have a common parent
	YLevel	The computed level of an assembly where the top level is “1"
	YName	The name of a parent of a given part
	YParentId	The identifier of a parent of a given part

Table 1. Tables and Columns for Bill of Materials and Single File Recursion

In this very specific example, the data that would exist across the X, XS, and XST tables is presented in the tables that follow.

X Table

Xid	XName
1	Wheel
2	Spoke

XS Table						
XSId	**XSTid**	**ActiveId**	**PassiveId**	**Seq**	**Level Number**	**Virtual Children**
1	1	1	0	1	1	N
2	1	2	1	1	2	N

XST Table	
XSTid	**XSTName**
1	Contain

The "data sentences" read:

X Table: There are two parts, 1 for Wheel and 2 for Spoke.

XS Table: Wheel (ActiveId = 1) is contained in nothing (PassiveId = 0). Spoke (ActiveId = 2) is contained in Wheel (PassiveId = 1). Each has a sequence of 1 in their assembly level of 1 and 2. Neither are virtual children having been included elsewhere first.

XST: Contain. This assembly collection type controls both XS entries.

Now, suppose the front wheel and the back wheel are different (which they are), and one is attached to the fork while the other is just attached to the frame, and finally, they both contain spokes that are the same (which they are), then the tables would have been:

X Table	
Xid	**Name**
1	Back Wheel
2	Spoke
3	Bike
4	Frame
5	Fork
6	Front Wheel

XS Table						
XSId	**XSTid**	**ActiveId**	**PassiveId**	**Seq**	**Level Number**	**Virtual Children**
1	1	1	4	2	3	N
2	1	2	1	1	4	N
3	1	3	0	1	1	N
4	1	4	3	1	2	N
5	1	5	4	1	3	N
6	1	6	5	1	4	Y
7	1	2	6	1	4	N

XST Table	
XSTid	**XSTName**
1	Contain

The level number of the XS row, 7 may appear strange. It would seem like it should be 5. However, because the “part” spoke is first at level 4, and it is merely a “virtual child” of the fork, then its “real” level is 4. Note that the XS row, 6 has the virtual children column marked as “Y.”

that is because its children, the spokes for the front wheel" were first "real" elsewhere, that is, the back wheel.

If the example had been accomplished within a single file recursion, then everything would be the same except for the spokes. The spokes would have had to be duplicated into a set for the front wheel and a set for the back wheel. That is the nature of a hierarchy versus a bill-of-materials. The table for Y would have thus been:

Y Table				
Yid	**Seq**	**Level**	**YName**	**YParentId**
1	1	1	Bike	0
2	1	2	Frame	1
3	1	3	Fork	2
4	1	4	Front Wheel	3
5	1	5	Front Spokes	4
6	2	3	Back Wheel	2
7	1	4	Back Spokes	6

Note the key difference: The spokes, which are presumed the same for both the front and back wheels must be duplicated. But because they are duplicated, both the front-spokes and the back-spokes are "real" and thus can have their own level number. Additionally, the Virtual Children column disappears as there can be no virtual children. For a very small application this difference may not seem worth all the effort. But for applications with thousands of parts that occur commonly in different assemblies, this difference is significant. Under the BOM, the parts never have to be duplicated, and if a change happens to the non-duplicated part, then it automatically affects all assemblies that include that part.

You can always install a hierarchy inside a Bill of Materials, but never the converse.

The remaining sections provide a more detailed example for both BOM and SFR, which is then employed in Section 6 of this guide and is the set of data that is contained in the zip file that contains the BOMSFR database application.

3.1 Bill of Materials

Bill of Materials is a generic name for commonly seen data model pattern. The example in this BOMSFR application is a calendar. The calendar has been somewhat contrived so that it works here, so you need to suspend your disbelief. In the Calendar example, there are the following part classes:

- Year
- Month
- Week
- Day of week.

And in this particular example, there are the years, 2003, 2004, and 2005. The months are: January, February, ..., December. The weeks are Week 1, Week 2, Week 3, and Week 4. The days of the week are Monday, Tuesday, ..., Sunday. The disbelief part here is that every month only contains four weeks. So, or this contrived calendar, there are just 48 weeks. Given that "slight" misrepresentation of a real calendar, there are just the following part counts:

Year	3
Month	12
Week	4
Day of Week	7
Total Calendar Parts	26

Parts, however, do not a calendar make. What makes our contrived calendar is the association of part together in to part structures. The following is an illustration for a single year.

Containing Part	Relationship	Contained Part
2005	Containment Note: Relationship count is 12. That is, 1 year and 12 months.	January February March April May June July\ August September October November December
January February March April May June July\ August September October November December	Contains Note: Relationship count is 48. That is, 12 months and 4 weeks.	Week 1 Week 2 Week 3 Week 4
Week 1 Week 2 Week 3 Week 4	Contains Note: relationship count is 28. That is, 4 weeks and 7 days.	Monday Tuesday Wednesday Thursday Friday Saturday Sunday

Table 2. Bill of Materials Parts for a Contrived Calendar

In the above table, the relationship count is 88 (12 + 48 + 28). In theory, our contrived calendar would have 12 * 4 * 7 days, or 366. The BOM application tree can, however "represent" the calendar in just 88 relationships. That is because when you add a part that contains relationship records to other parts you are also automatically adding the other parts. For example, in our

application, to create another year, say 2004, you merely have to add the part 2004, and the 12 relationships between the newly added year and the 12 months (January, ..., December). Because each month already has four weeks and for each week, seven days associated with it, those are then automatically included in the new year's calendar by reference.

To add the first year requires 88 relationships. Thereafter, a new year is added by adding just one part, that is, a new year such as 2002, and 12 relationships between the newly added year and the 12 months..

That's the good news. Now for the bad news. In a bill of materials structure, the display of the records, that is, the 328 days in the calendar are virtual, not real. Consequently, when the BOM window closes, the apparent 328 days disappear and you are left ONLY with 88 relationship records and the 26 calendar part records. Further, suppose you add a 8th day to week 4, say, QualityTimeDay, which might be the day where we all spend quality time with each other rather than working. Instantly, there will be an 8th day for every week 4 across all months of all the years. Because the days are actually virtual across the calendar.

Because of this virtual characteristic, it is not possible to record information specific to that calendar day. If you were able to record information to, for example Wednesday of Week 2 within January of 2005, then that very same information would appear in every Wednesday of every week of ever month of every year.

3.2 Single File Recursion

Single File Recursions has a key characteristic similar to Bills of Materials, that is, it presents a tree structure. But there is one very important difference: All the records in the Single File Recursion are real. None are virtual. Hence, a real calendar can be created and all the days will really exist. Because the records are all real then information can be recorded within each day. It the example above, there would be 328 records, one for each day. Consequently, when the SFR window closes, the 328 days remain as 328 individual records.

Section 6 illustrates the creation and use of Bills of Materials and Single File recursions. Table 2 identifies these data structures in the metabase system.

Metabase Module	**Bill of Materials**	**Single File Recursion**
Business Information Systems	Business Event Cycle Calendar Cycle	Business Information System
Database Object	None	Database Object

Data Element	Compound Data Element Concept Conceptual Value Domain Data Element Classification Data Element Concept Value Domain Value Domain Values	Business Domain
Document and Form	Document Form	None
Implemented Data Model	None	Column Table
Information Needs Analysis	None	None
Mission Organization Function Person Assignment	None	Database Domain Mission Organization
Operational Data Model	None	DBMS Column DBMS Table
Resource Life Cycle Analysis	Resource Life Cycle Node	Resource
Specified Data Model	None	Entity Subject
View Data Model	View Column	None

Table 3. Bills of Material and Single File Recursions in the Whitemarsh Metabase

4 Reference Data

There is no reference data in this metabase application.

5 Operation

Once the application is installed, it is ready to use. Just invoke the Bill of Materials and Single File Recursion (BOMSFR) program. The application is a traditional windows application.

6 Process Model

The process model consists of two classes of processes:

- Bill of Materials
- Recursive Structures

Figure 2 presents the main menu for the Bill of Materials and Single File Recursion application. The menu items are just two: Bill of Materials and Recursion.

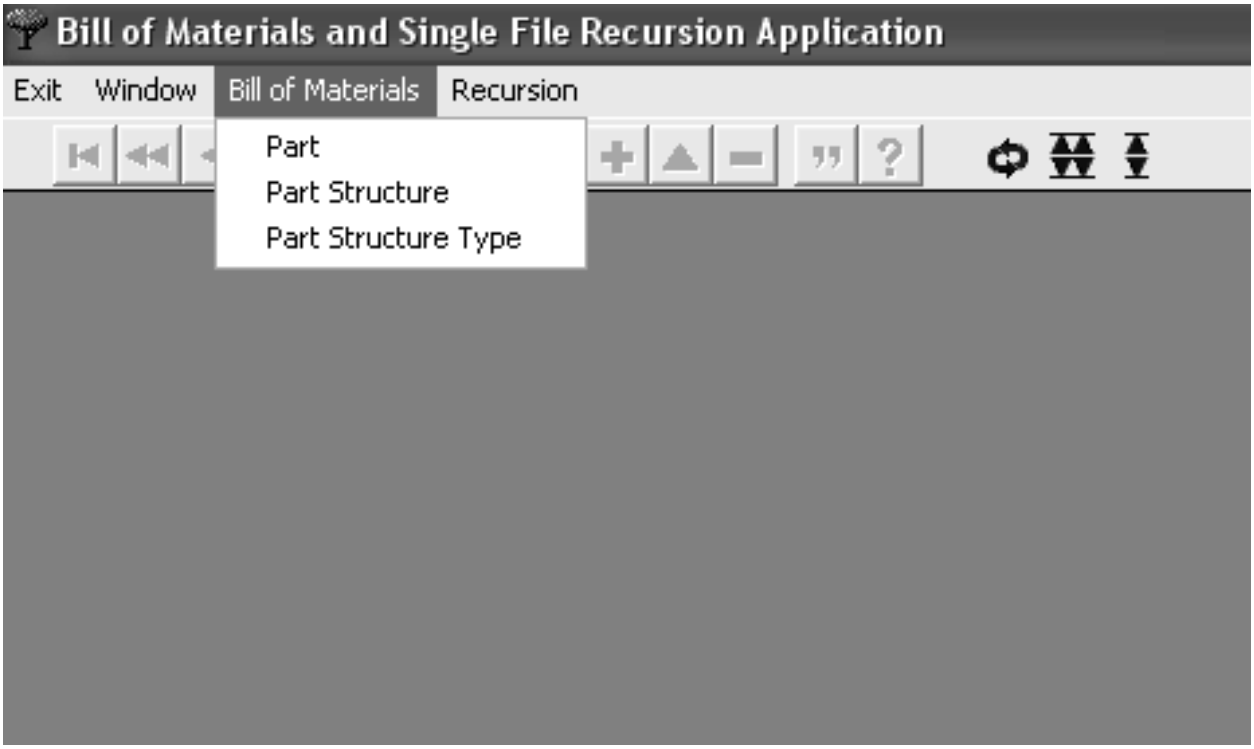

Figure 2. Main menu for Bill of Materials and Single File Recursion application.

Highlight the main menu item and select it. The menu items for Bill of Materials consists of three subitems, Part, Part Structure, and Part Structure Type. The Recursion menu item contains two subitems, Recursive Collection and Reassignment.

6.1 Bill of Materials

The bill of materials consists of three tables: X, XS, and XST. Effectively these are parts, assemblies, and assembly collection types.

6.1.1 Part Processes

Figure 3 presents the list of all parts within a bill of materials. This list appears after the Bill of Materials main menu item is selected and then the Part submenu is selected. The part identifier and the name of the parts are presented in the list. This list contains all the discrete parts, like years, months, weeks and days of weeks. These represent the 26 distinct parts to represent the modified calendar.

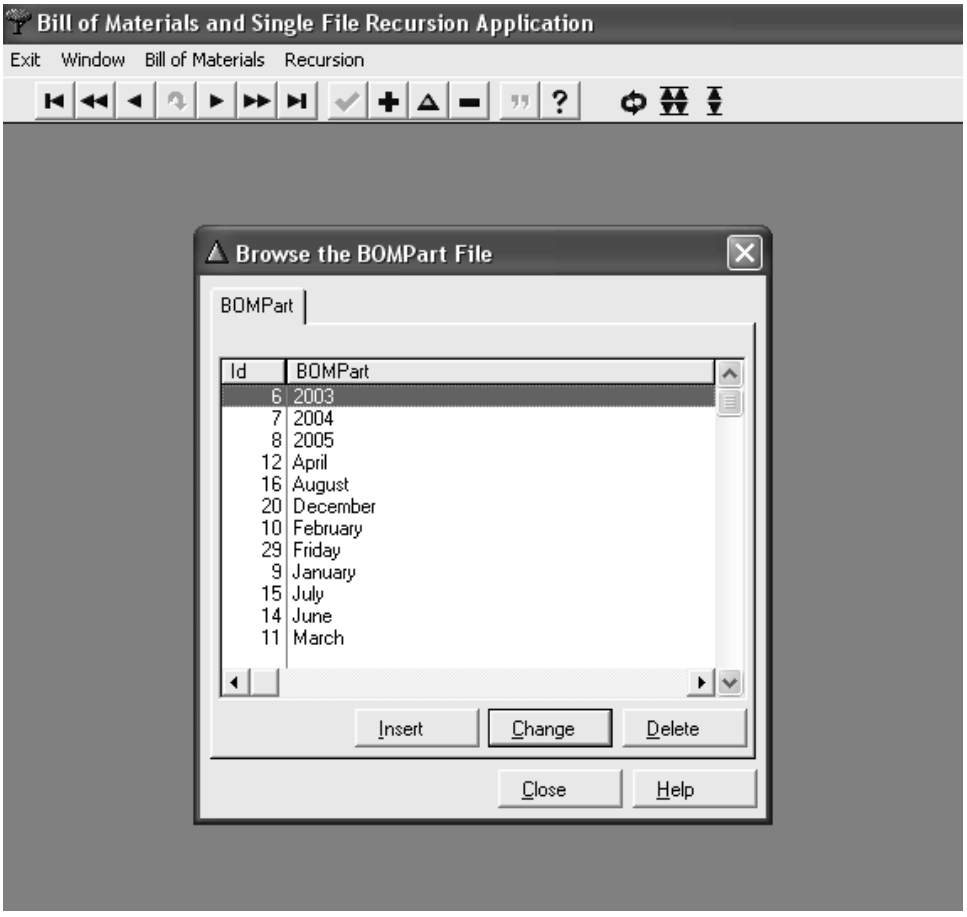

Figure 3. List of all Parts.

To add a new part press Insert. The update screen as illustrated in Figure 4 is presented. Once a part record is entered it is ready to be incorporated into an assembly.

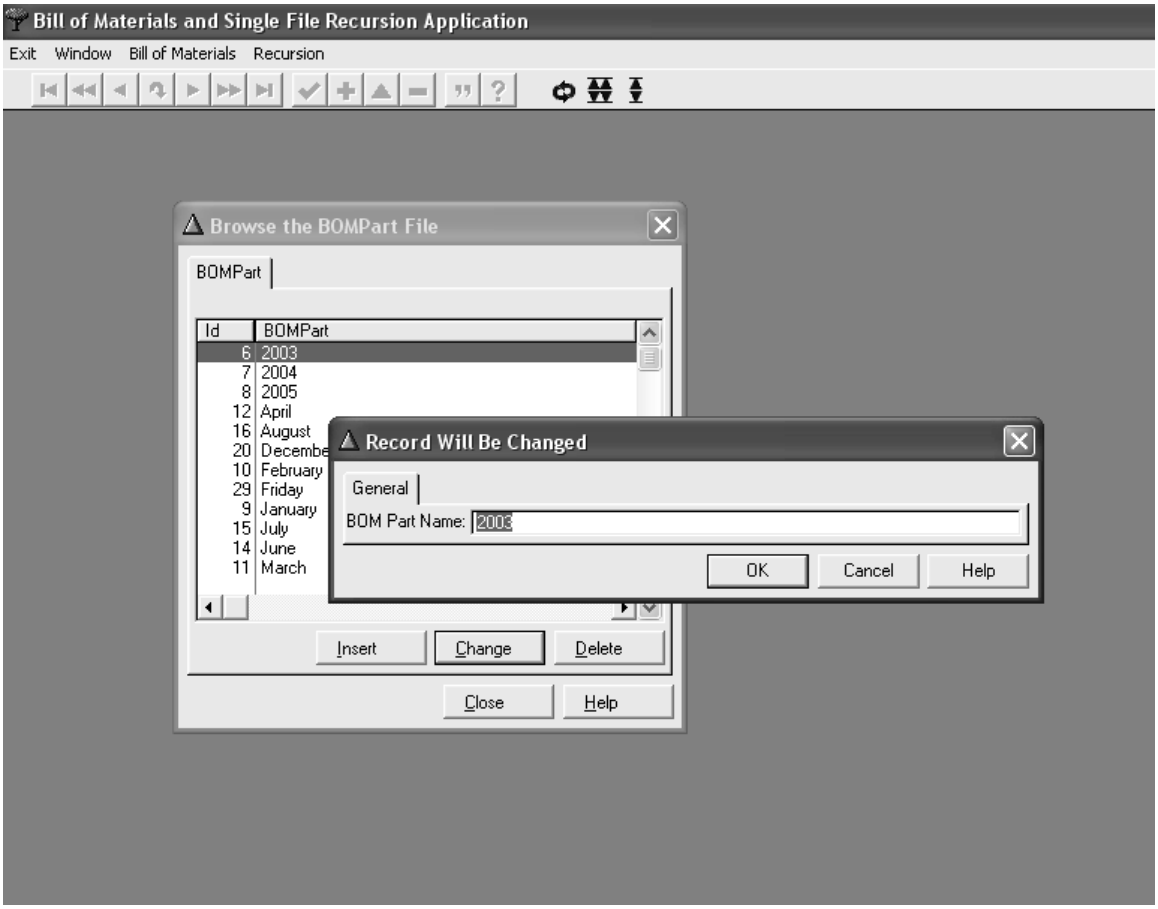

Figure 4. Adding or changing the part record, 2003.

Figure 5 presents the delete screen. If this part is not involved in any assembly the deletion is allowed. Otherwise it's prevented.

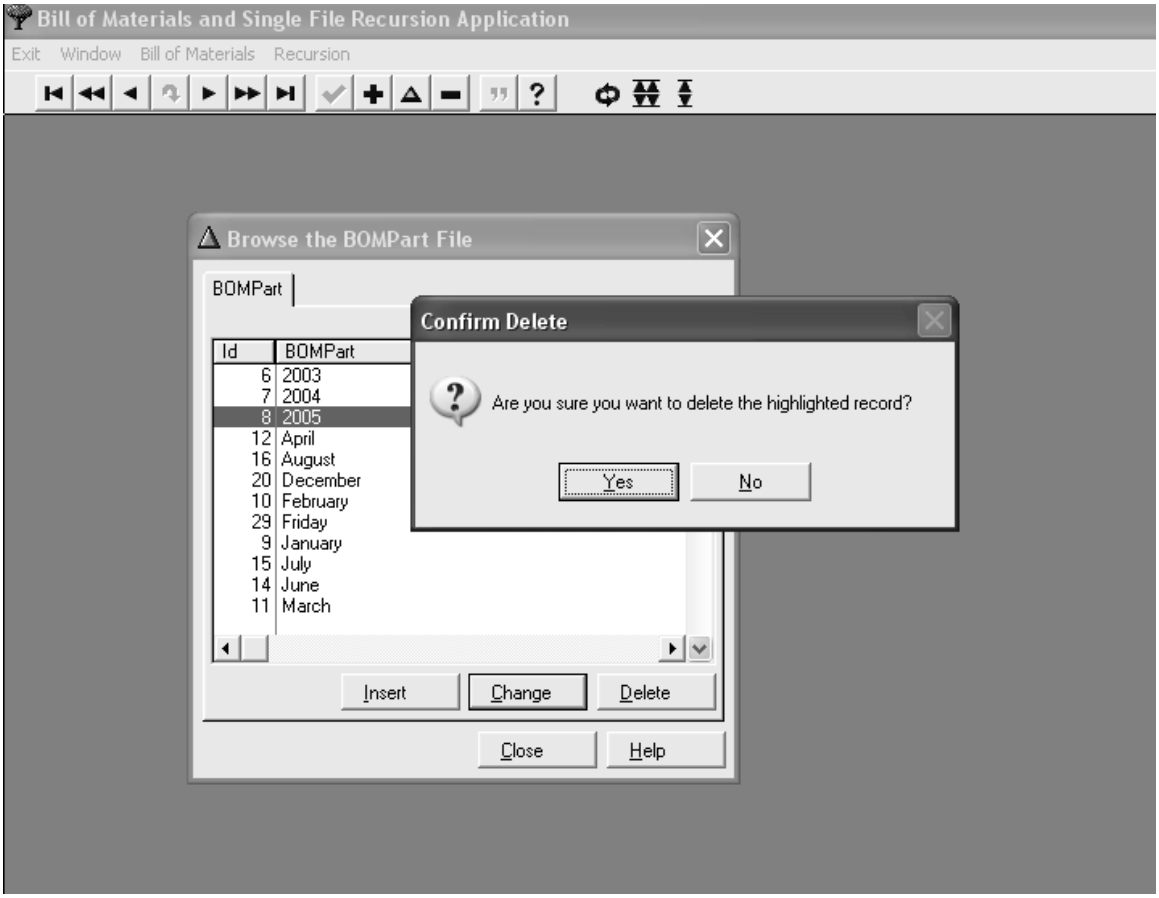

Figure 5. Part record delete screen.

6.1.2 Part Structure Processes

The part structure processes start with a list of the structures. A structure is the association of one part with a parent part. Structures are presented in a tree-like fashion. To present the structures, select the main Bill of Materials menu item and then the Part Structures menu item. A list like that in Figure 6 is presented.

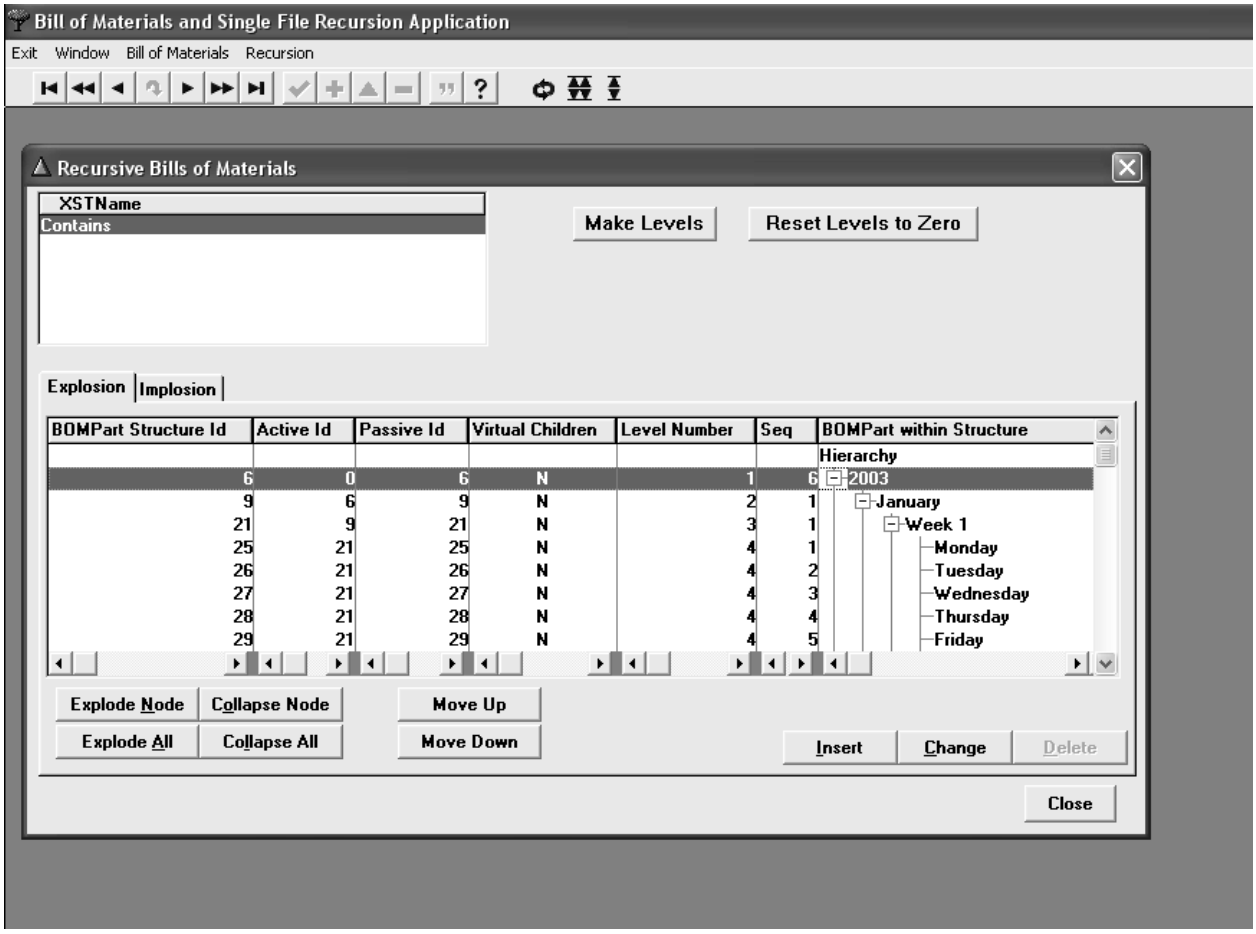

Figure 6. Listing of Part Structures.

This tree structure shows all the data associated with a structure. Normally in metabase structure windows the only data that is shown is the tree structure itself, that is, the calendar, and the sequence number. All the other columns of data, while present in the database are necessary only for the orderly processing of structure records. The reason why they are shown in this BOMSFR application is to illustrate how the tree structures are really engineered and operate. The first column is the primary key value of the structure record.

The second column is the Active Id value. For a given record, it represents the Part Id value of that record's parent. So, for 2003, which has no parent, the value of Active Id is zero.

The Passive Id value represents the Part Id of this specific record. The Virtual Children column is the indicator that all records for which this record is the parent are duplicated. This is specifically addressed in a section below.

The Level Number column is the level in the structure tree that this structure record actually represents when and only when the level record actually exists versus apparently exists. This too is specifically addressed in a section below.

The Seq column is the sequence number that this structure records is said to have with respect to its peers. Arbitrary Sequences are not always required in metabase Bills of Materials. Sometimes alphabetic sequences are the only ones necessary. Column sequences within a key, or module execution sequences within a procedure are however, examples of arbitrary but necessary sequences. In this particular example, sequence is necessary and can be maintained either by the sequence of inserting records into the structure or by selecting a records and "moving" it either up or down through the use of the "Move" buttons. Records can only be resequenced within their immediate level.

The last column, BOM Part within Structure, shows the parts and their relationship with each other within the structure. In this case, there is the year, 2003, then the month, January, then Week 1, then the days of the week. Notice that all the parts are unique. That is, that all the Passive Ids are different numbers. Hence each is uniquely represented. Parts that have already been entered into the structure are repeated, and so their Passive Id values would be the same. For example, in the BOMSFR application itself, rather than this user guide, look at the PassiveIds of the days of the week (Monday, ..., Sunday) within each week. Notice that they are the same part identifier values, meaning that Monday of Week 1 has the same identifier value as does Monday of Week 2. That is because there is one set of days of the week that are used over and over again for every week of every month of every year.

6.1.2.1 Inserting Structure Records

To insert a new structure record, press the insert button. When this is done, Figure 7 is presented. To insert a new structure, the following must be done:

- Select the parent, and in this case, it's January of 2003.
- Select the part to be inserted, and in this case it's a Week 5.

Now, if three distinct errors do not occur, then the "OK" message is presented and the structure can be created. If any of the three messages occur then the Select button is disabled.

The three possible errors are:

- Inserting the same part as was selected,
- Inserting a part that is already within that level of the structure

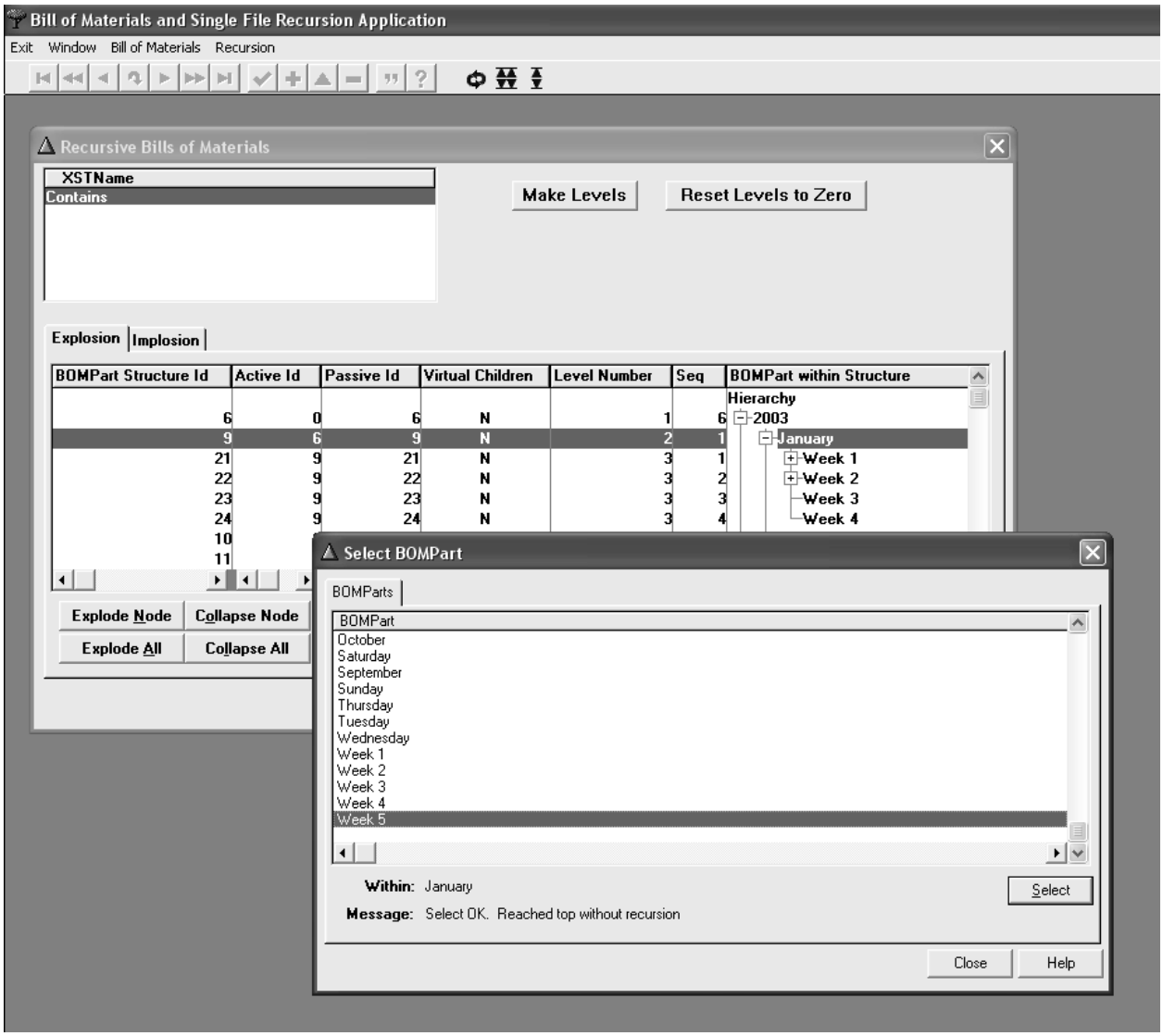

Figure 7. Acceptable BOM structure record insert.

- Inserting a part that then causes a recursion.

Figure 8 illustrates the "same part as was selected" error. In this particular case, Year 2003 was selected, and the structure to be created was also Year 2003. Hence, same as selected. Notice that the Select button is "greyed out."

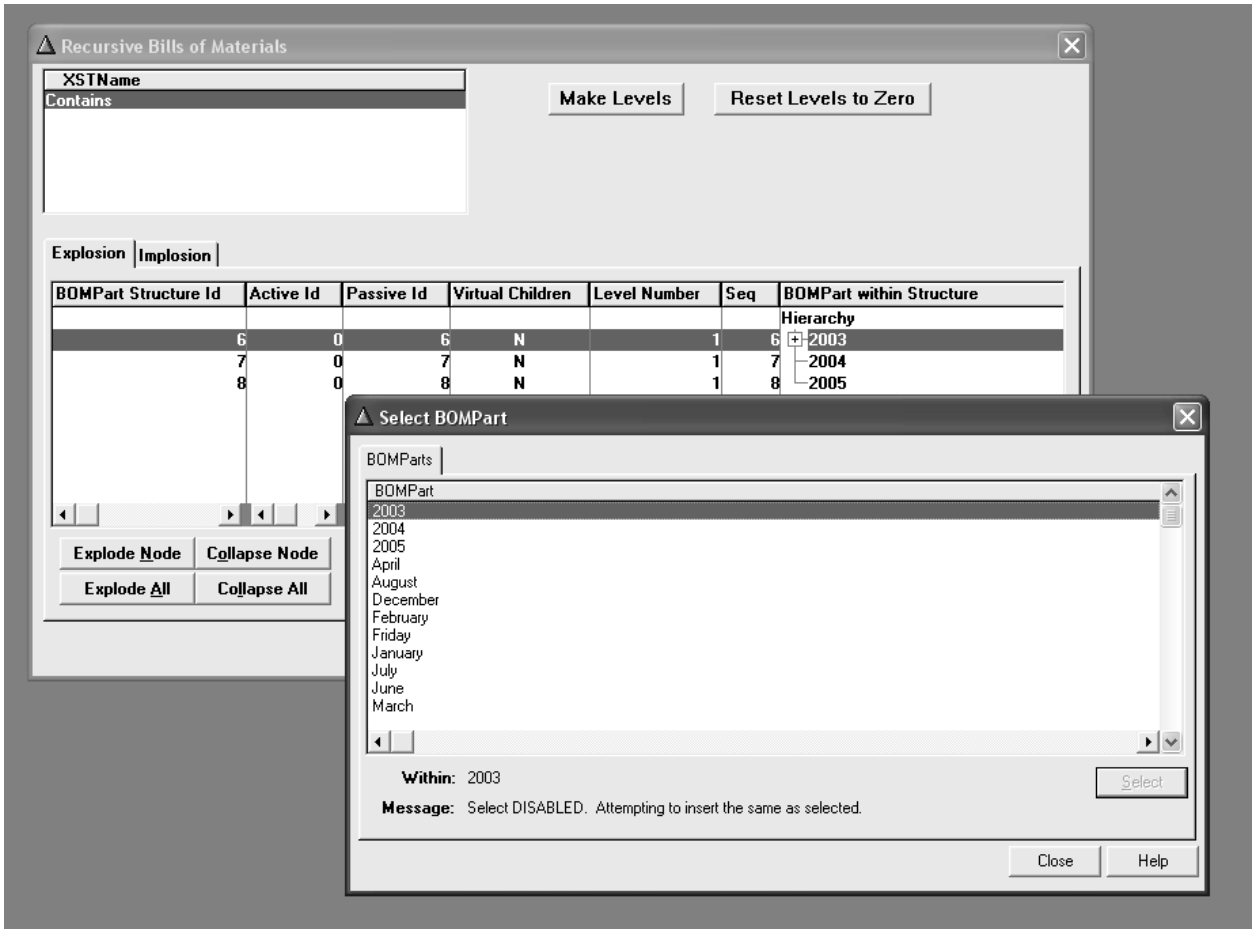

Figure 8. BOM insert error: Same part as was selected.

Figure 9 illustrates the "attempting to insert a twin" error. In this particular case, "Hierarchy" (the root level) was selected, and the structure to be created was Year 2003. Hence, it is a twin because at the root level, all the records are years, and 2003 is already present. Notice that the Select button is "greyed out."

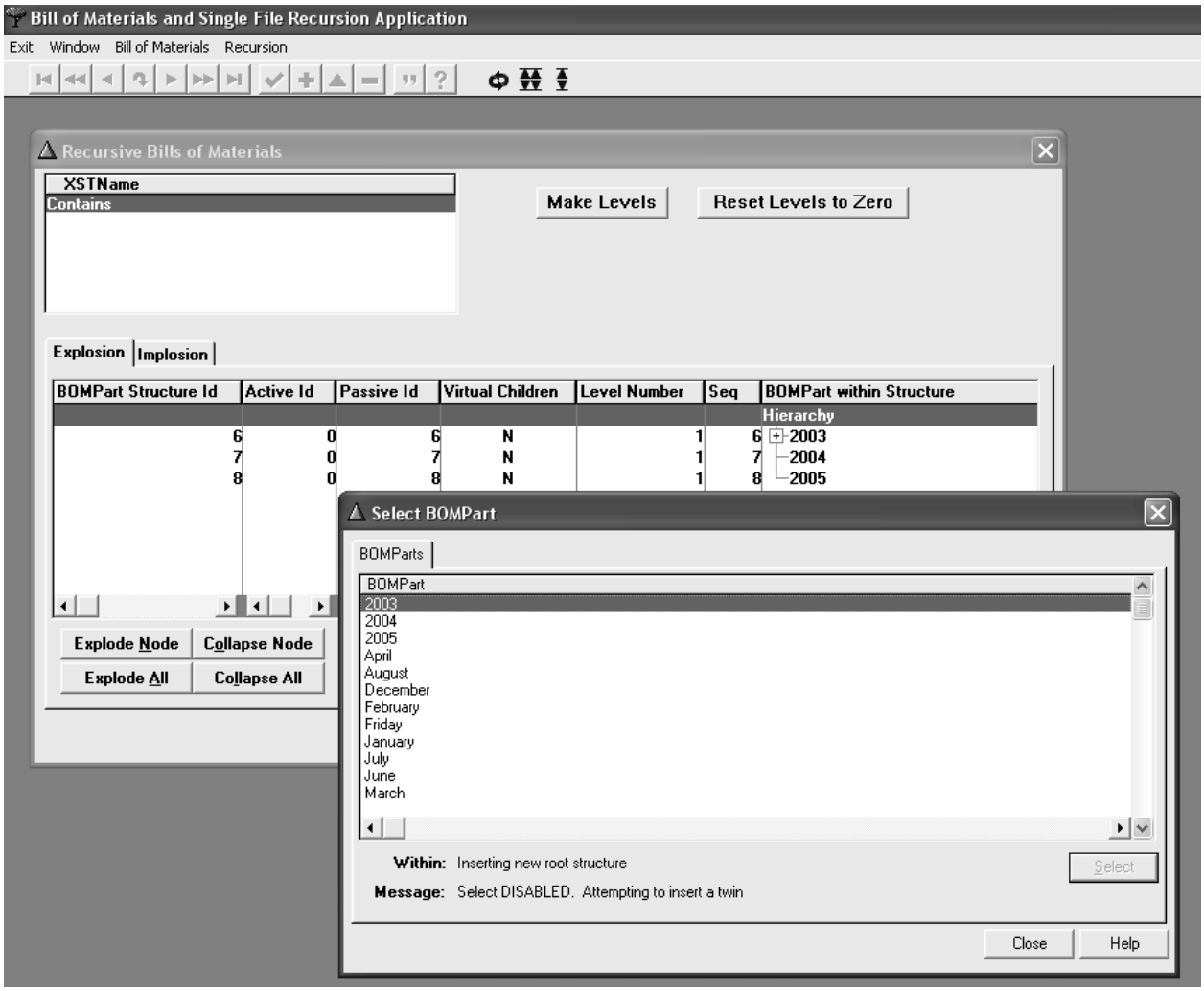

Figure 9. BOM insert error: Attempting to insert a twin.

Figure 10 illustrates the "Insert will result in recursion" error. In this particular case, January was selected, and the structure to be created was Year 2003. Hence a recursion. That is, Year 2003 would contain January, which in turn would contain 2003, which would in turn contain, January. Thus the infinite recursion. Notice that the Select button is "greyed out."

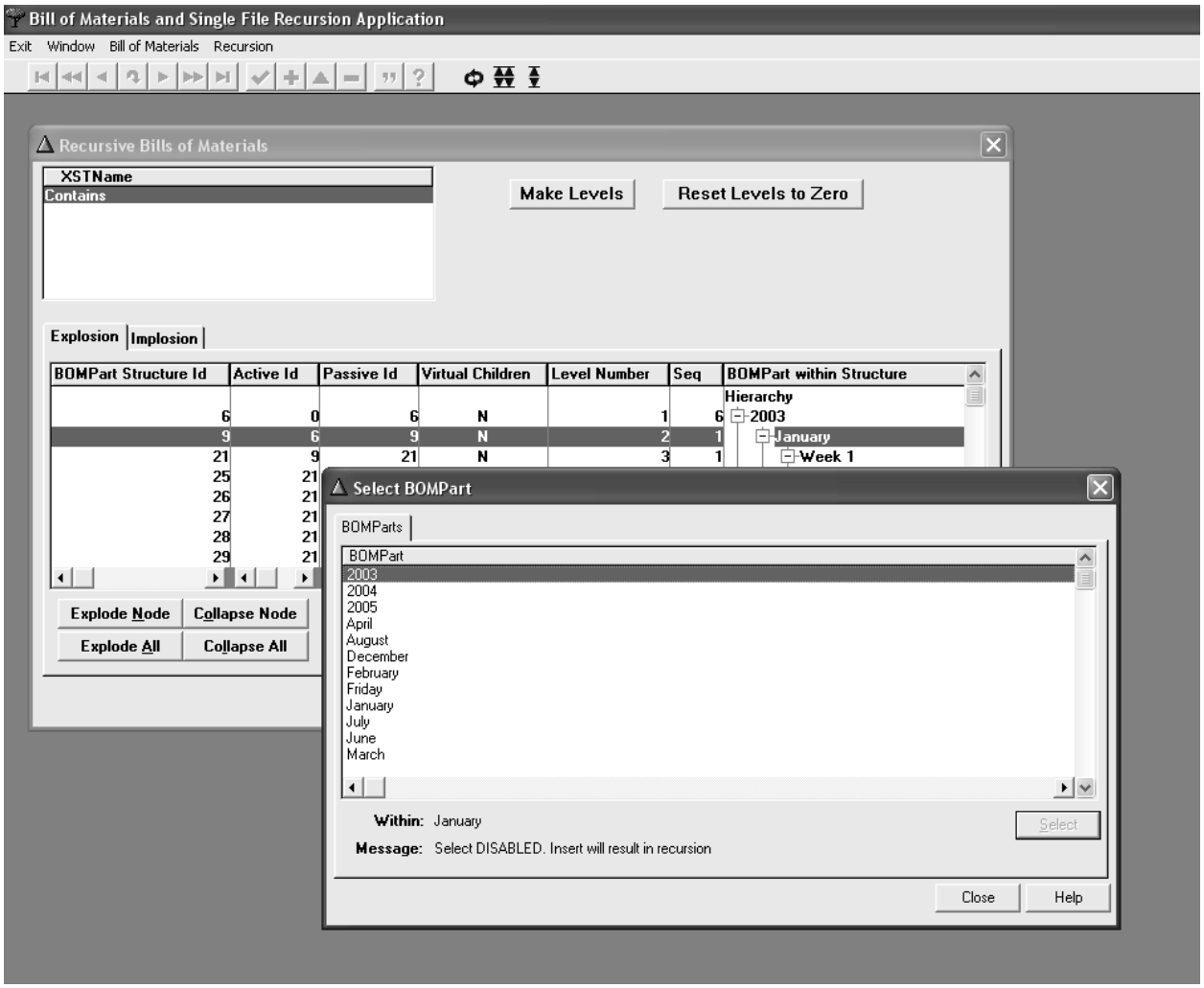

Figure 10. BOM insert error: Insert will result in recursion.

6.1.2.2 Structure Levels

As part records are associated with other part records within the Bill of Materials, they exist within levels. If the top level, that is 2003 is considered level 1, then the months, January through December are at level 2, and Weeks 1-4 are at level 3, and so on.. As part records are inserted into the structure their level is automatically calculated and then displayed. This is illustrated in Figure 11. There are two buttons on Figure 11 that can illustrate the process of creating and assigning levels. The Reset Levels to Zero, when pressed, resets all the record levels to zero. The "zero" levels are displayed after the finished message appears and the browse is toggled. That is, by re-selecting the XST Name.

6.1.2.3 Virtual Children

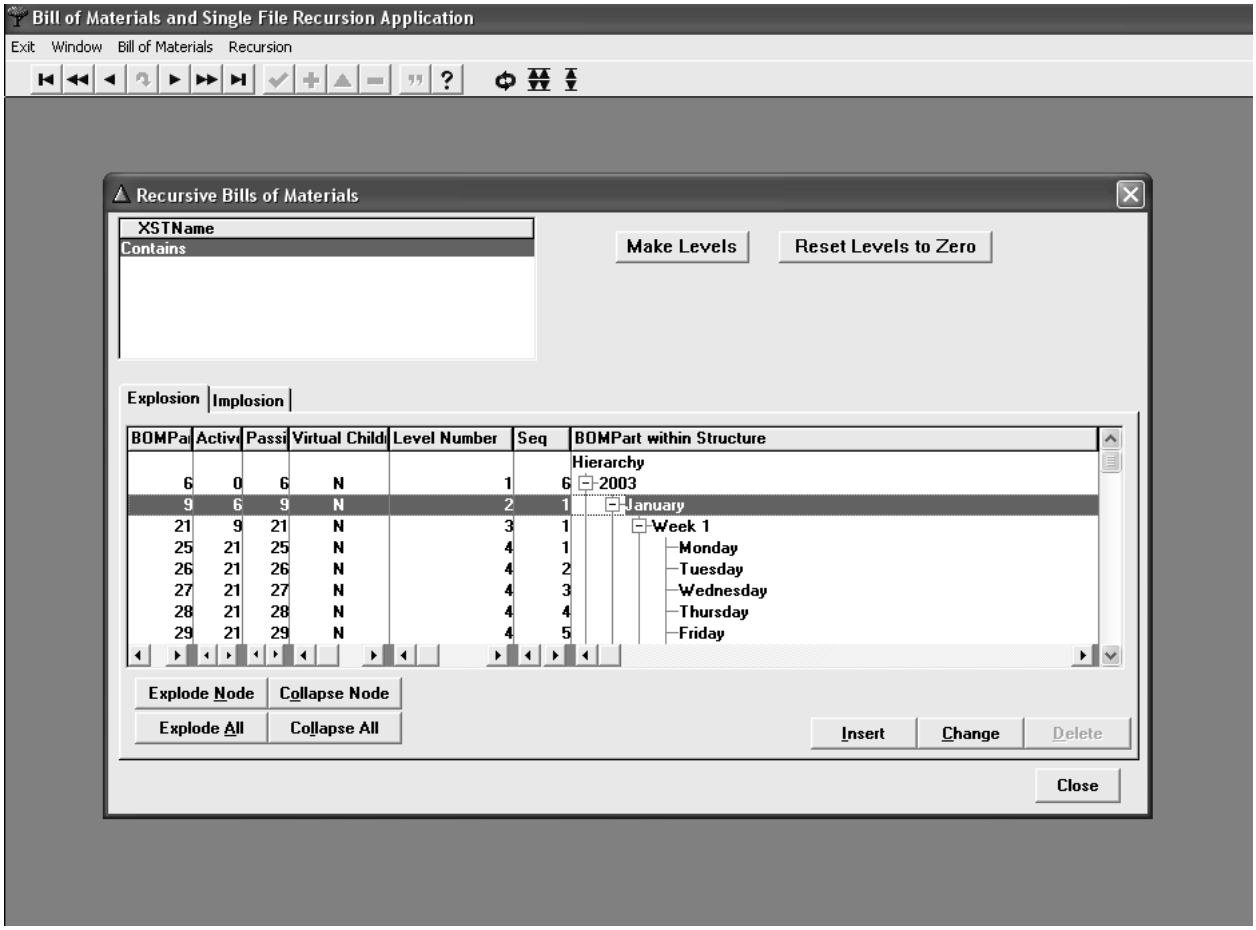

Figure 11. BOM structure levels.

A most interesting characteristic of a Bill of Materials is that when an existing structure that itself consists of multiple levels of structure records is related to another structure that does not already contain a set of structure records, the newly assigned set of structure records then appears to exist a second time. The quickest example of this feature is the fact that there are only four weeks in the calendar, Week 1, ..., Week 4. Further, days Monday through Sunday are associated with each of those weeks. When Weeks 1 through 4 are associated with January, they appear to be associated as the weeks within January. The same appears to be true when they are associated with February through December. Further, when January is associated with a new year, say, 2004, then "automagically" 2004 has a full January. That is, Weeks 1 through 4 and for each week, Monday through Sunday. Figure 12 illustrates this capability. The "parent" of the virtual children is shown a light gray.

Notice in Figure 12, the BOM Part Structure Ids for Monday through Sunday of Week 1 of January.. The numbers are 25 through 31. Now, notice the BOM Part Structure Ids for Week 1 of February. They are the same. That is, while the records appear in the window, they do not actually exist in the underlying database. If you assign Week 1 to March through December, the same result occurs. This is a fundamental characteristic of a Bill of Materials. Wherever virtual children exist, the parent of these children, Week 1 in February of 2003, and all the remaining weeks of February through December of 2003, and then every week of every month for 2004, etc. appears with a yellow highlight. It appears grey in this user guide. Starting in 2004, not only are the week records virtual, so too are the month records.

Now, if you delete Week 1 from January 2003, then it will automatically be deleted from every January for every Year. First, insert a Week 1 into 2004. Then, delete Week 1 from January 2003. Then see if Week 1 is in 2004. It's gone. When you do this, contract and then expand January. To restore Week 1 to all Januaries, just add it to 2003. It then appears in all years. But it now appears after week 4. To fix that, use the Move button to move it up to before Week 2. When you do the move, it moves in every January of every year.

Virtual children have an effect on the Level Numbers. If the records do not really exist, then the level numbers may similarly be affected. In the calendar example, the level numbers all appear to be acceptable only because we have maintained a symmetric calendar. If a virtual child, say Week 1 were to be placed directly below a year, say 2005, then the levels would appear broken. They are not however broken. This is shown in Figure 13. Week 1 was assigned to 2005. It's level with respect to 2005 is correct. But because Week 1 contains Monday through Sunday of another Week 1 that is already at level 3 for 2003 (Year then Month then Week), then the level numbers for the days are level 4. If however, Week 1 had been assigned to 2003, and then Months and then Weeks assigned to 2004 and 2005, the levels would have been based on 2003 because it is the first level 1 entry upon which the levels are based whenever a part is reused.

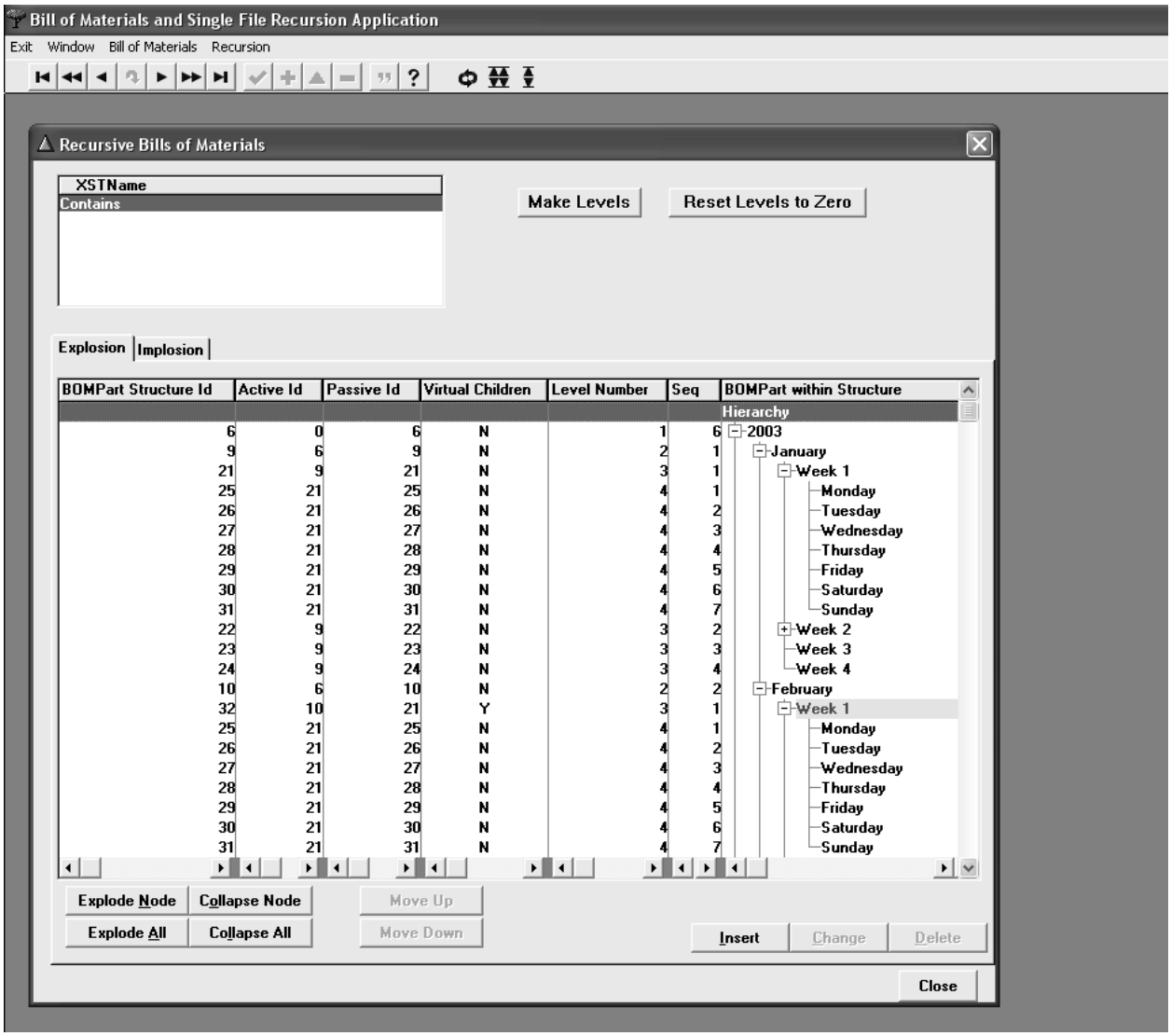

Figure 12. BOM and virtual structure records.

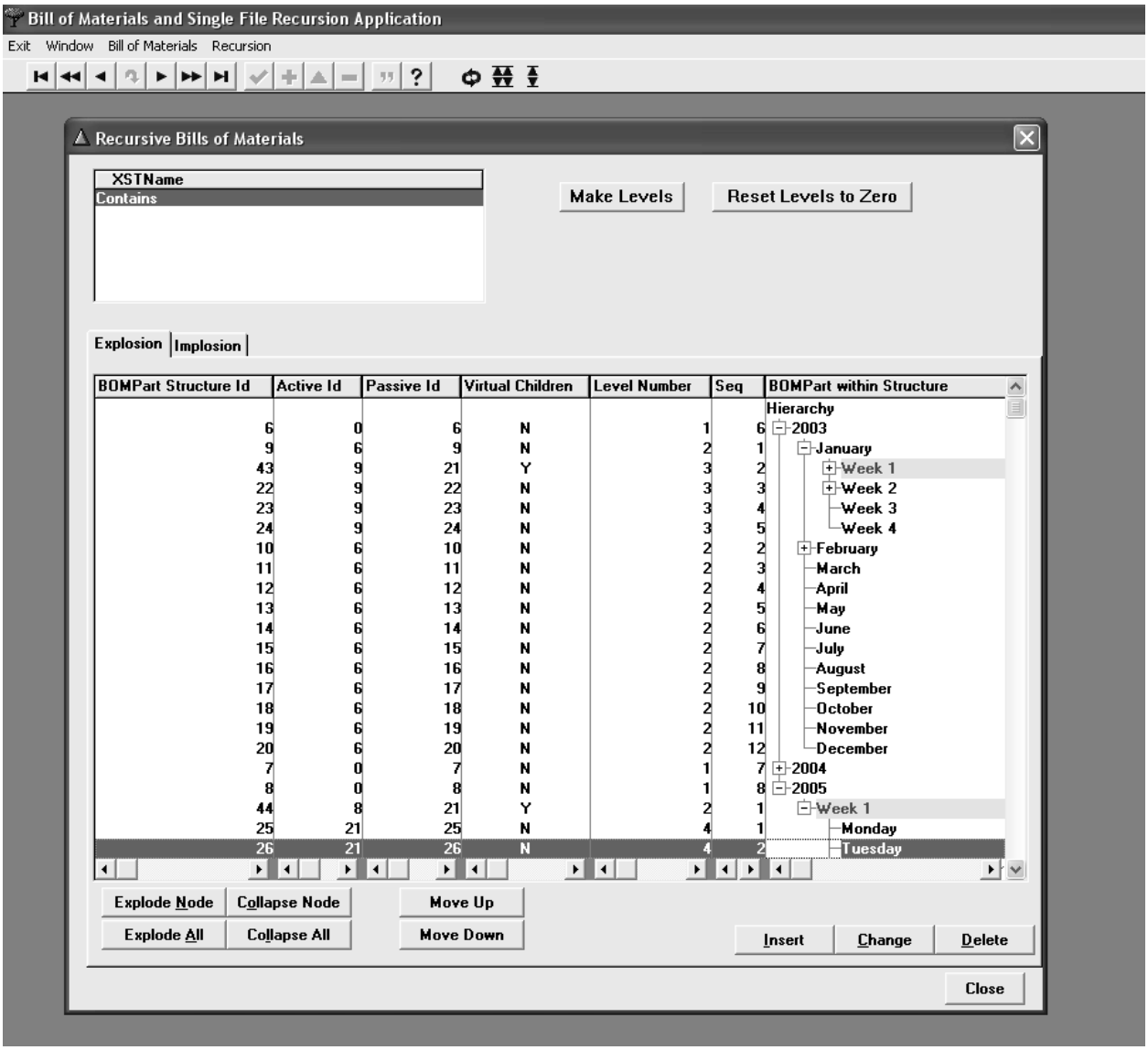

Figure 13. BOM virtual children and level numbers.

The final process for BOM structures is delete. Deletes are either allowed or not. Two rules govern deletes:

- If a structure has contained structures AND is not repeated elsewhere the delete is disallowed.
- If the structure has no contained structures or is repeated elsewhere the delete is allowed.

Figure 14 shows that the delete button is "greyed," thus the delete is now allowed. It is disallowed because January has contained structures (Week 1 through Week 4), and in this particular set of data there is no other year that contains a January.

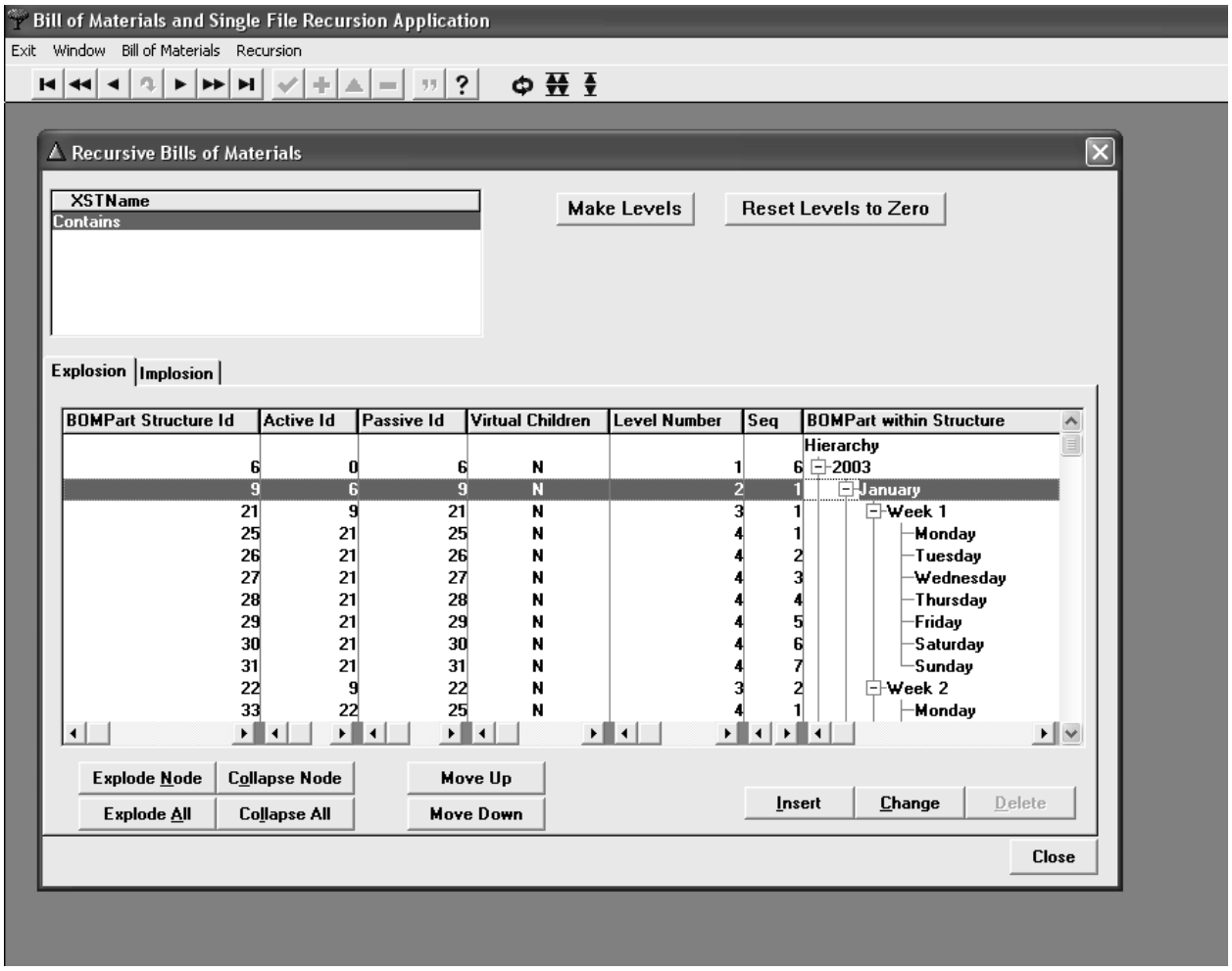

Figure 14. Disallowed BOM structure delete.

Figure 15 shows an allowed delete. The reason the delete is allowed, is because Week 1 is found in more than one month. What is then actually deleted is the single structure record between the January part and Week 1. Before the delete occurs, a warning messages is presented that requires confirmation.

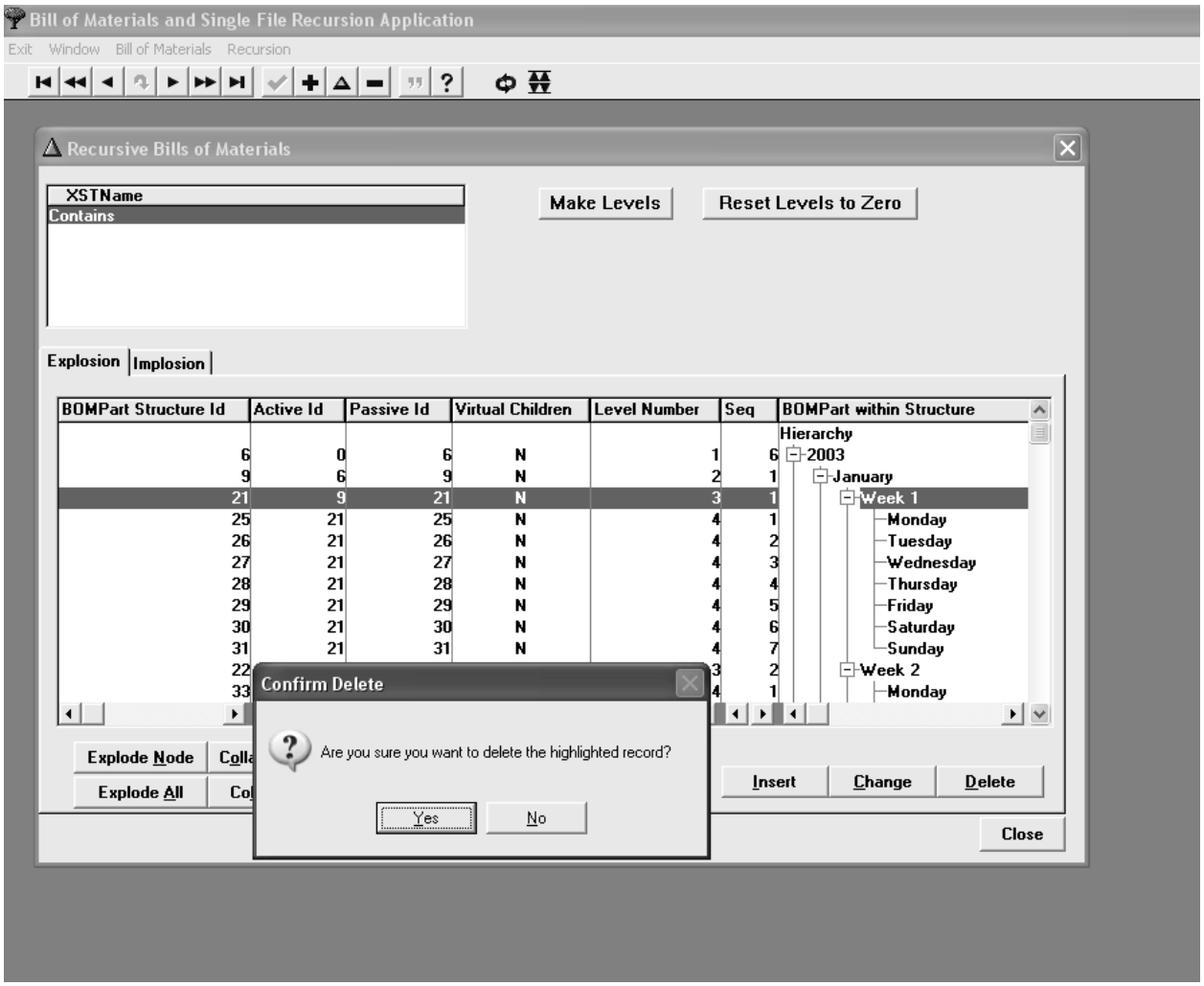

Figure 15. Allowed BOM structure delete.

6.1.3 Part Structure Type

Part structure types enable the isolation of collections of part structures. In this example, we only have containment. There could be others as well such as Controls, Directs, Manages, Influences, and the like. In the metabase proper, these part structure types all have two phrases for the concept: active and passive. So, for the once cited above, they would be:

Concept	Active	Passive
Containment	Contains	Is contained in
Control	Controls	Is controlled by
Direct	Directs	Is directed by
Influence	Influences	Is influenced by
Manage	Manages	Is managed by

Figure 16 presents the Part Structure Type screen. To add a new entry, press Insert. If an existing entry is deleted, it will be so only if there are no existing structures controlled by the part structure type. The Part Structure Type update screen is listed in Figure 17.

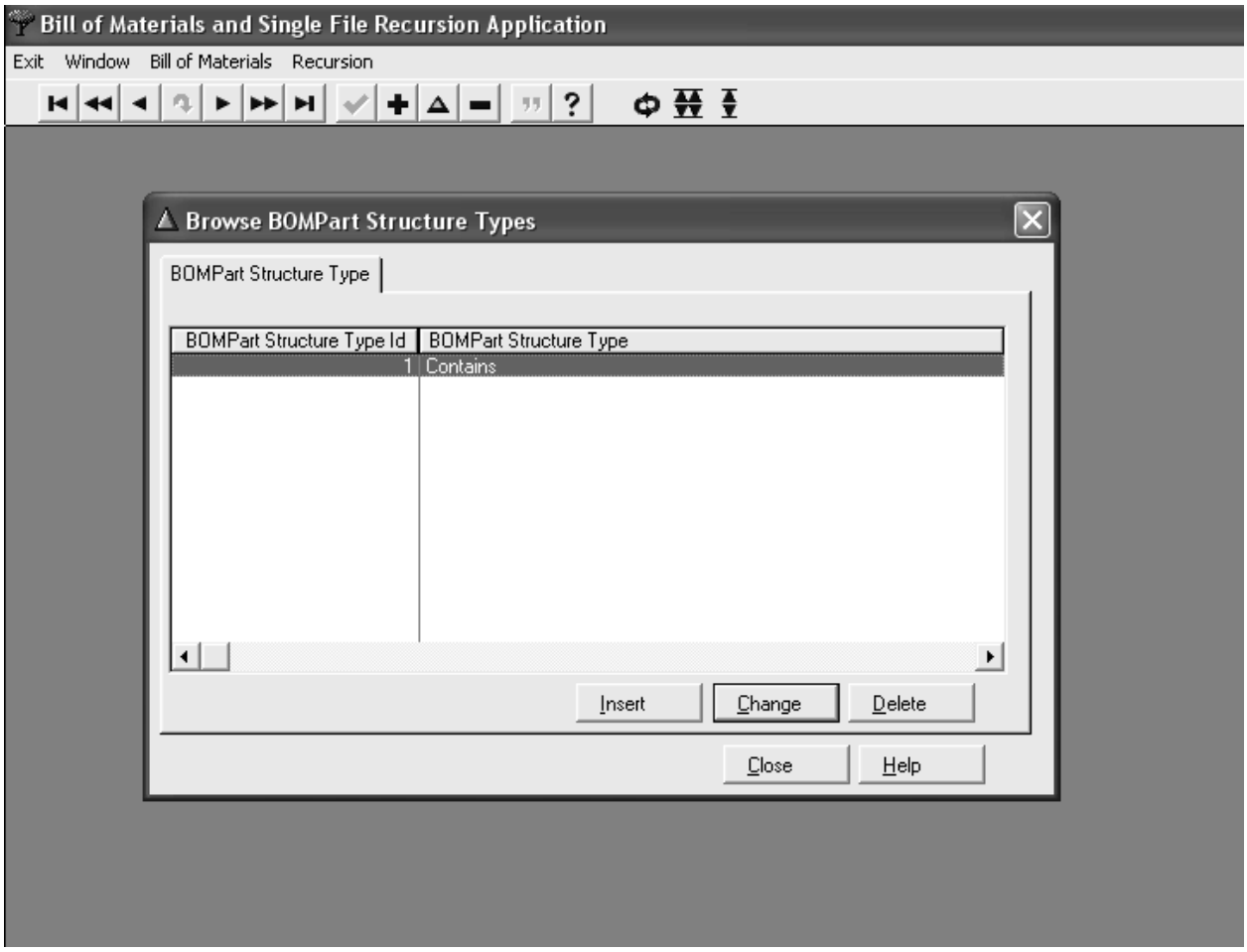

Figure 16. Listing of the BOM part structure types.

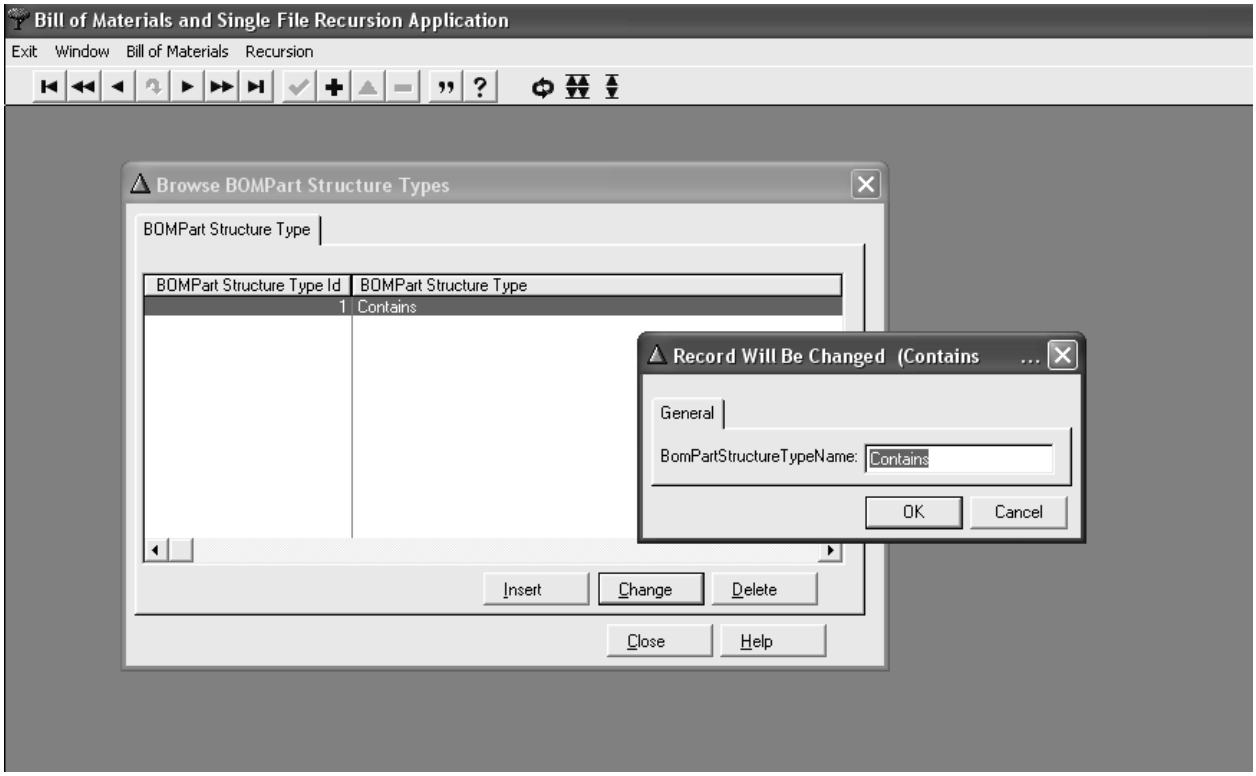

Figure 17. BOM Structure Part Type update.

6.2 Recursion

Recursive structures are different from Bill of Materials Structures in three ways:

- Recursive structures can only store hierarchies
- Recursive structures are based on only one table
- Recursive trees must have real parts at all levels, no virtual children.

Recursion process models consist of the recursion operations, levels, and reassignments. To start the recursion processes, select the menu item, Recursive Collection as displayed in Figure 18.

6.2.1 Recursion Process

Figure 18 presents the basic recursion window. The browse looks very much like the Calendar data that is in the Bill of Materials. The similarity is not accidental. There is, however several big differences. First, every part of the calendar is real. That is shown by the fact that every Id is a unique number. Note that with respect to the identifiers for Monday through Saturday in Weeks 1 and Week 2. Thus, in the actual file, Y, there are currently 14 records for days. This means that if you were in insert a new day into Week 1 for January 2003, for example. RestDay, then only Week 1 of January 2003 would have that new day. In contrast, in the Bill of Materials, if you added a new part, RestDay, and then associated it with Week 1, then every Week 1 for every month of every year would automatically have a RestDay.

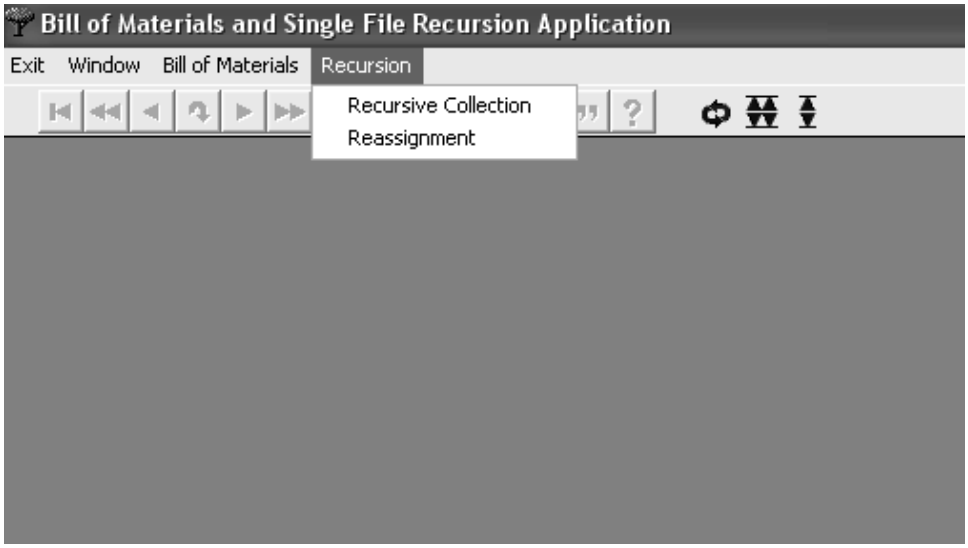

Figure 18. Recursion Menu.

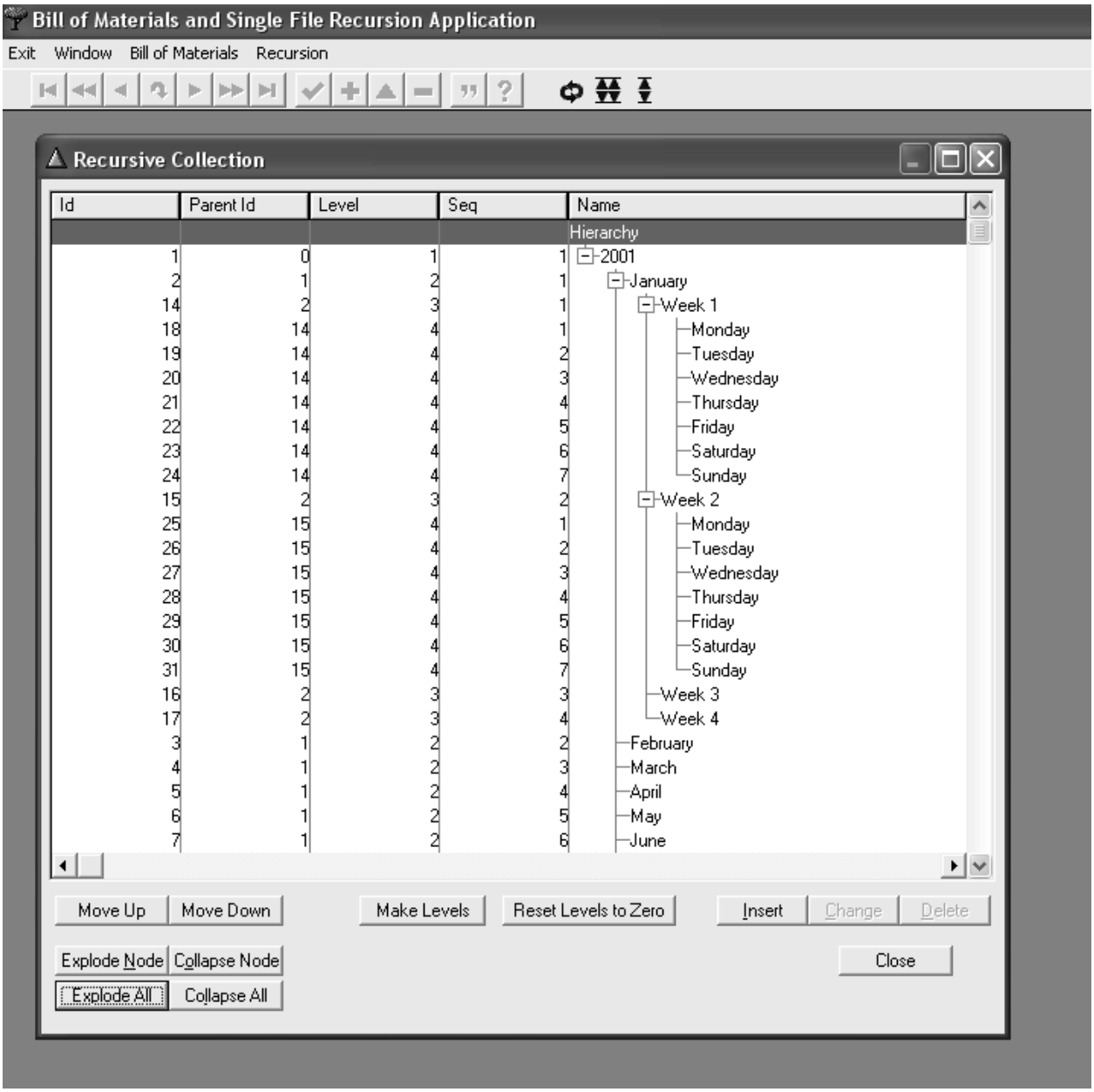

Figure 19. Single file recursion browse window.

Inserting, Updating is the same, and is instigated by selecting either in the Insert or Change button Figure 20 presents the update screen. To insert, select a parent part, press insert, enter the new part name, and upon closing the window the new part is inserted as a new child at the end of all existing children for that part. In this particular example, RestDay will be inserted after Sunday of Week 1. If a part is not properly sequenced, it can be moved up or down via the Move buttons.

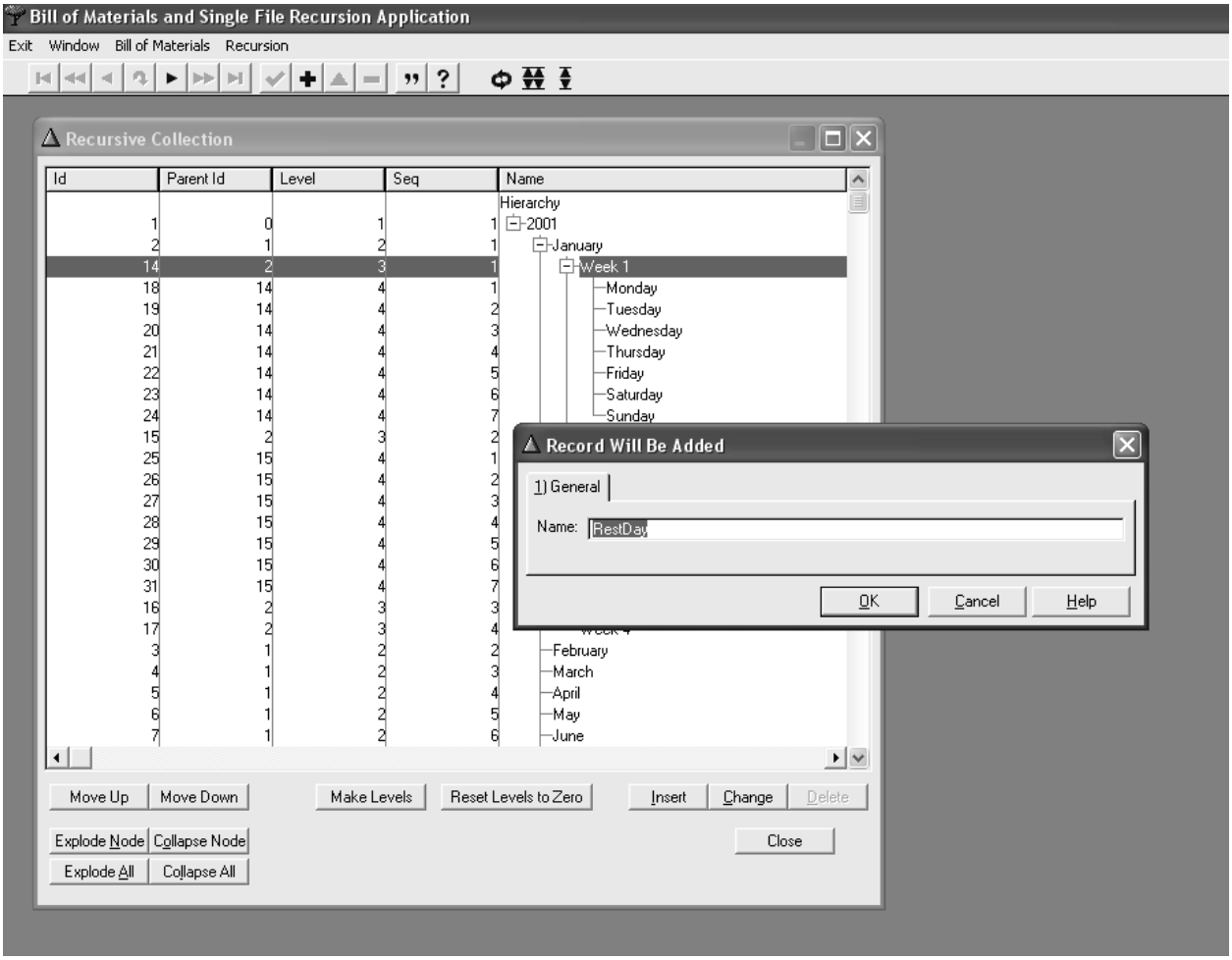

Figure 20. Inserting a new part in a single file recursion.

The result of inserting the RestDay is shown in Figure 21. Note that the RestDay is at the end of the week, and had the example had a Week 1 in February, then there would have been no RestDay.

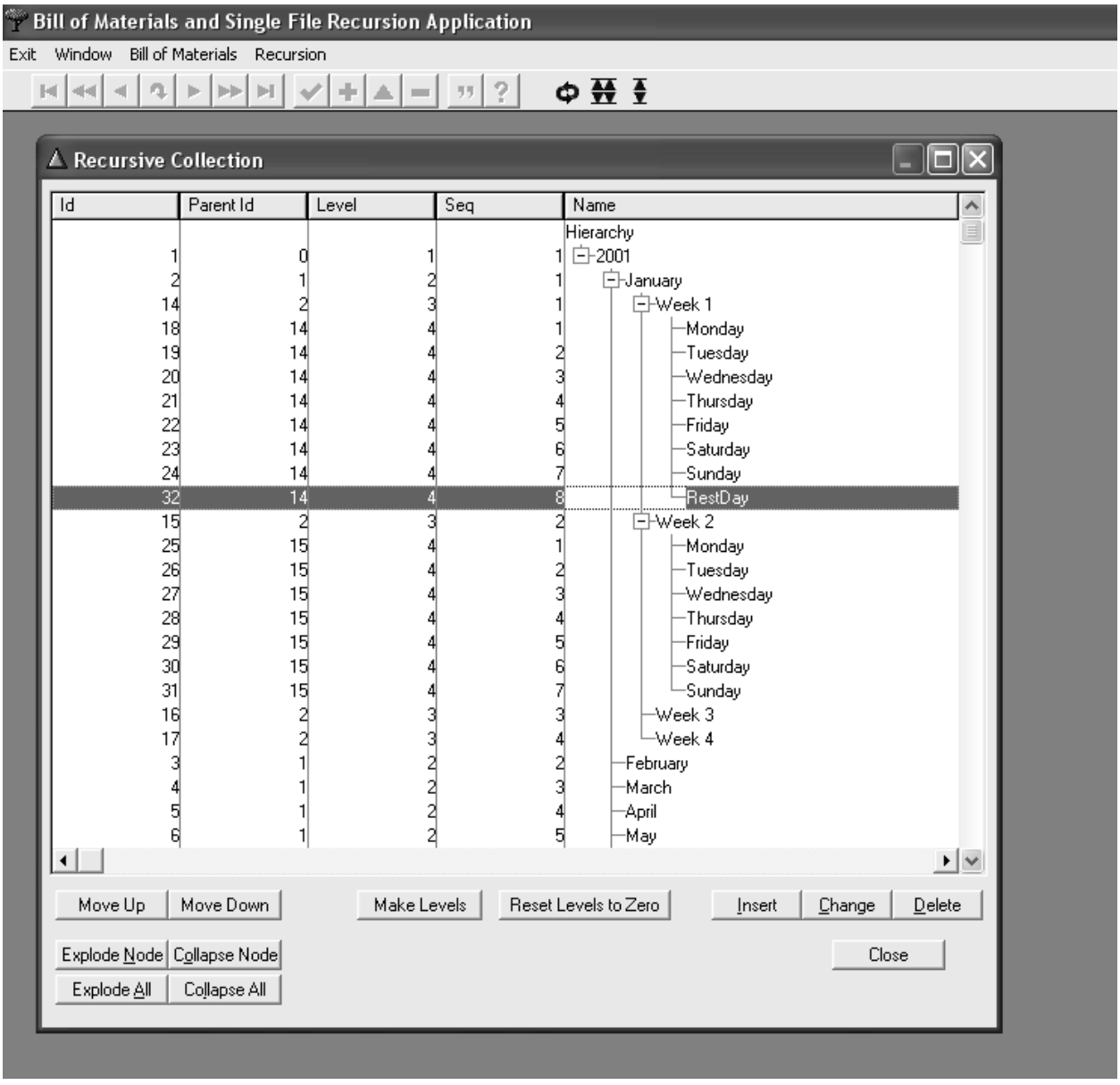

Figure 21. Result from inserting RestDay in the single file recursion.

There are the two buttons for Levels. In the case of Single File Recursions, where there are no virtual children, every part is given a level number either through the process of Make Levels, or through normal inserts or deletes.

Deleting is somewhat different. The delete button is automatically disabled if any part contains a subpart. The rule is that deletion occurs from the outer most part. This is merely to prevent whole trees from being deleted accidently. Figure 22 illustrated the "disabled" Delete button.

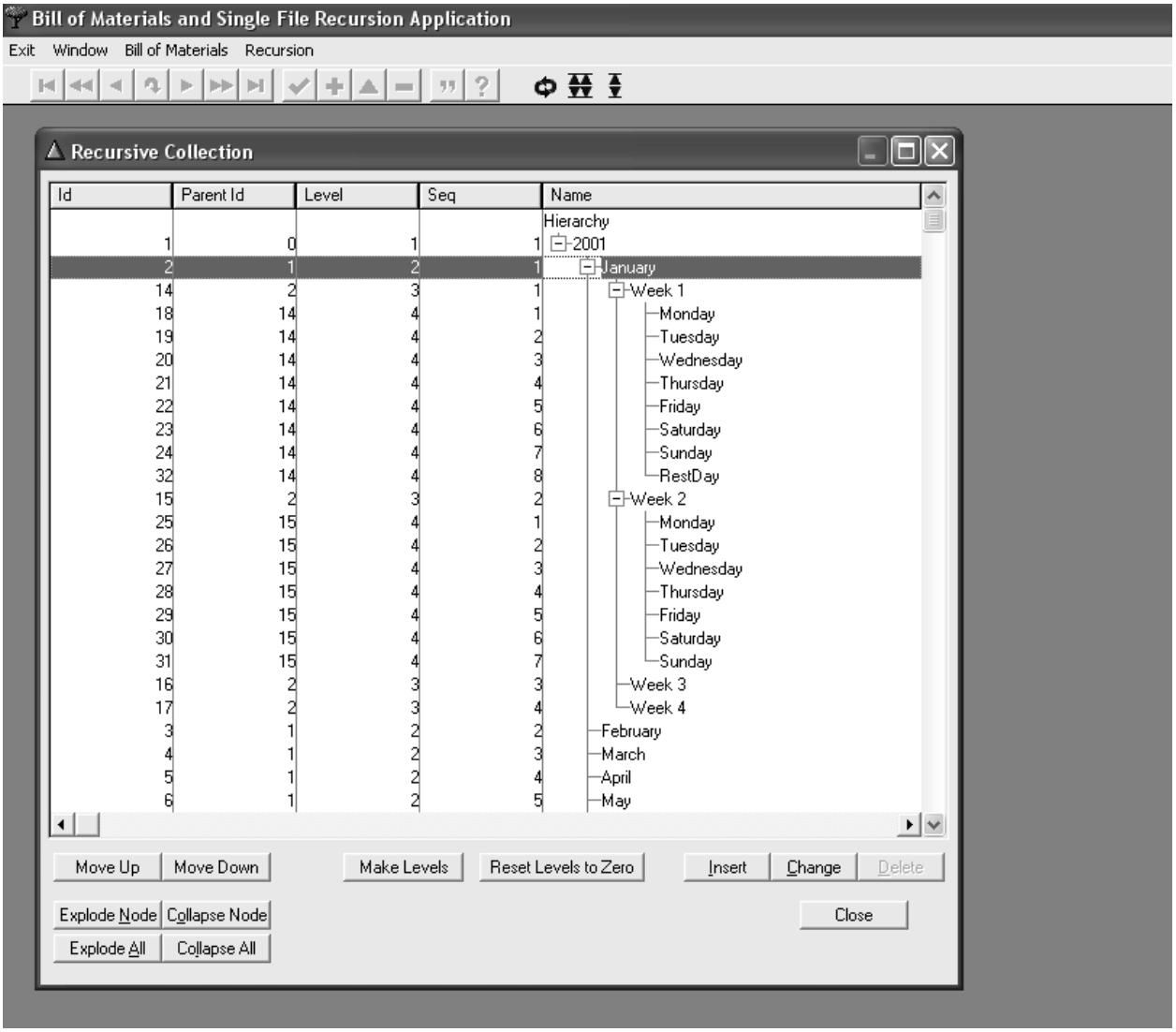

Figure 22. Disabled delete button on a single file recursion.

6.2.2 Recursive Reassignments

In the case of Single File Recursions, parts can be reallocated. This is accomplished though the process of reassignments as illustrated in Figure 23. In this window, Week 1 has days. Week 2 does not. Days Monday through Sunday are tagged (checked). Now, if on the right browse, Week 2 is tagged and then the "press" button is pressed, the days are MOVED from Week 1 to Week 2. Figure 2, Figure 24 results. The week days have been moved.

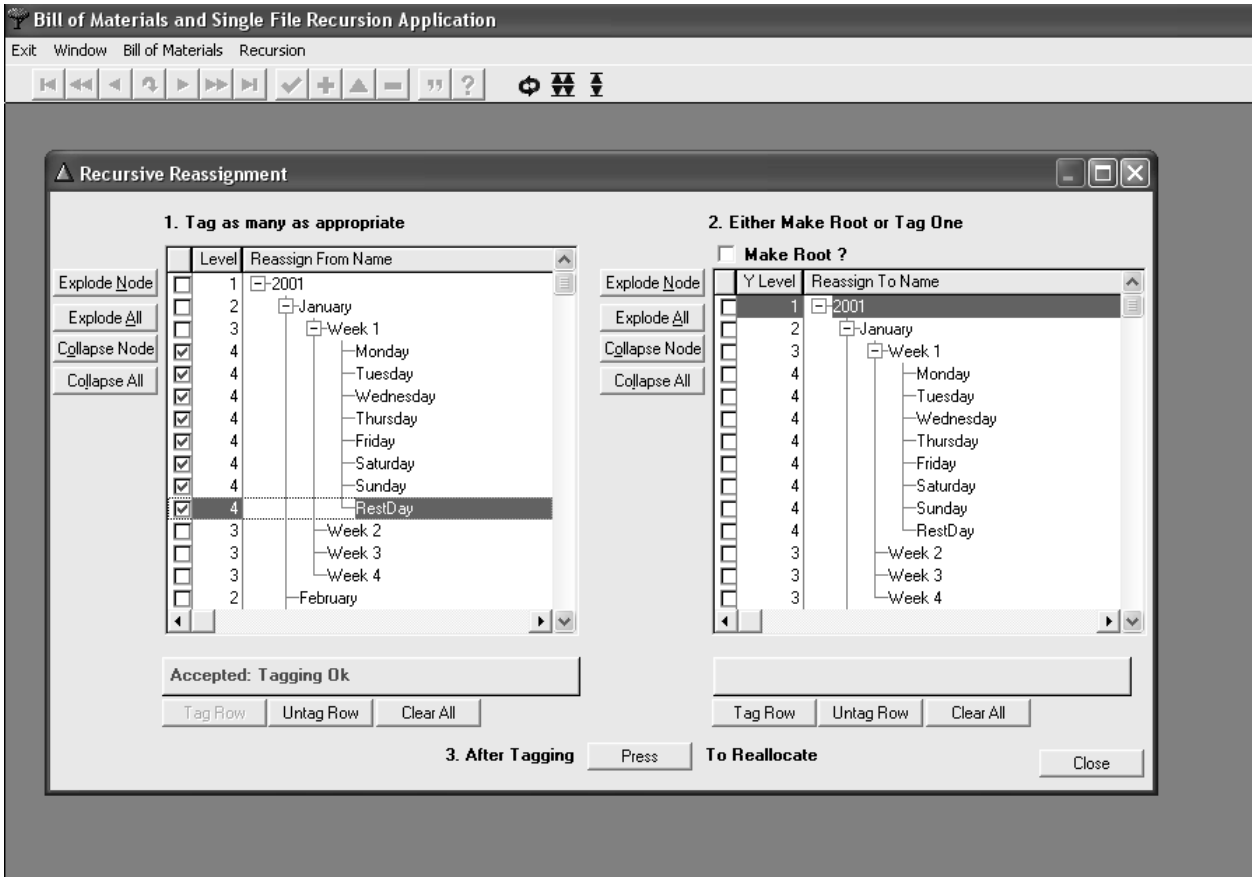

Figure 23. Single file recursion reassignment tagging.

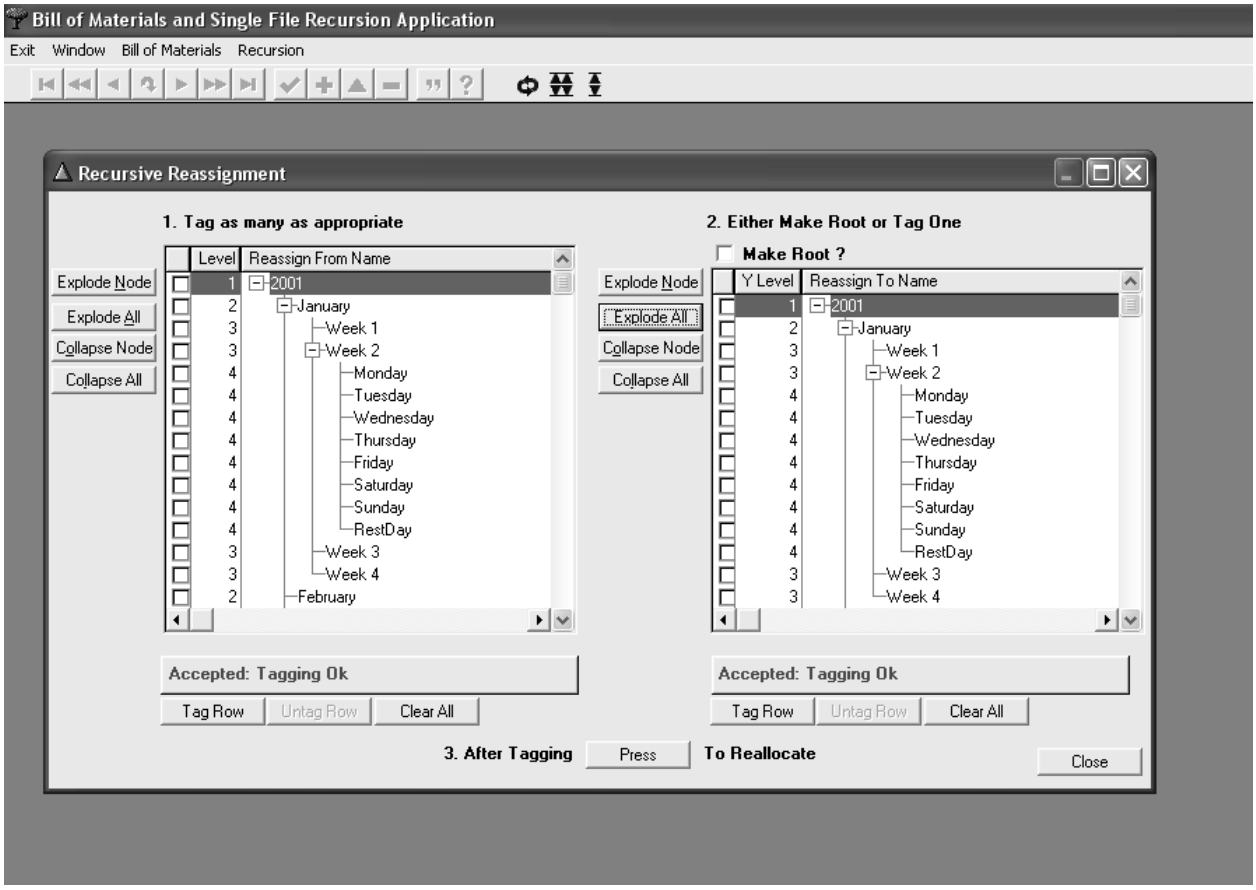

Figure 24. Single file recursion reassignment result.

7.0 Bill of Materials and Single File Recursion Summary

There are very significant differences between Bills of Material and Single File Recursion.

Bills of Materials are represented through three tables, part, part structure, and part structure type.
In contrast, Single File Recursions are represented through a single table.

Parts that are employed multiple times in a Bill of Materials almost always result in structures with "virtual children." Hence the part structure record count almost always exceeds the part record count. Further, Bill of Materials tree display record count may greatly exceed the part structure record count. And when the Bill of Materials display is closed the display records disappear altogether. In contrast, Single File Recursion display record counts always match one to one with the real record count. There are no "virtual children."

When structures with contained structures are added into a Bill of Materials the quantity of Bill of Materials tree display records will rise without an increase in part records. For each such structure addition only one new part structure record is created. In contrast, when a set of single file recursion records are reallocated to a different structure they are "moved." Actually on the Parent Id column value of the newly attached structure is modified.

Bills of Material structures may have levels, but those levels are "real" only if the part structure records are real. Thus the level numbers of virtual children are as originally created for the first instance (plus one) of the parent. In contrast, Single File Recursion levels are always "real" as they are always attached to real records.

Bill of Materials can store hierarchies as well as networks of related records. Single File Recursions can only store hierarchies.