# Upgrades with User Created Tables

By Thomas Fitch, Technical Support Engineer, 4D Inc.

Technical Note 02-08

## Abstract

This Technical Note prepares developers for creating a 4D application which allows end users to create their own tables. It includes two sample databases. One represents the current version of a database, just finished by the developer. The other represents the end user's version of the same database, which the developer would like to upgrade to the new version. Because the end user has added some user created tables to the database the upgrade process is more complicated than if he or she did not have that option.

## Introduction

There are three main parts of the Technical Note. First is a description of the process for developing and updating a database that allows end users access to the database structure. The second part is a short description of the sample database for use as the end user, similar to a user manual. Finally there is a scenario of what the end user would have to go through to update to the latest version of the database. The design of the sample database is explored in this third part.

Included with the Technical Note are two sample databases. Although these databases are very simple in what they allow end users to do with their user created tables, the ideas can be ported to more complex applications.

The first database, in the "End User" folder, represents a database that has been edited by the end user and contains tables created by the end user. This will be referred to as the **end user database** or **end user version** throughout this Technical Note.

The second database is in the "Developer" folder and has an updated version of the database with a table added by the developer. It is this database that the developer needs the end user to upgrade to. This database will be referred to as the **current database** or **current version** throughout this Technical Note.

In the Extras folder of both databases is one document: structure_log.xml. This file tracks changes the user makes to the structure.

## Planning to Implement User Created Tables

When developing a database and planning to offer the end user the ability to create tables and edit the database structure there are a few things the developer must decide. Here are some issues to consider:

- What options to offer the end user? Will they only be able to create tables? Will they be able to change the structure of the table, delete fields and change the data type stored in a field? Will they be able to delete tables? Will they be able to add indexes to fields or relations between fields and tables?
- How will end users enter data into user created tables?
- How will data be displayed for user created fields?

These first few points are outside of the context of this Technical Note. Each of these questions should be addressed before moving to the other design stages though.

Once those decisions have been made the developer can move onto the question of how to handle updating a database if they allow end users to create tables. Here are some questions to address:

- How to track user changes to the database structure?
- How to test the database to see if the structure needs to be updated to match user changes?
- How to update the latest version of the structure to include user changes?
- How to handle the data in user created tables when upgrading the structure?

## Tracking User Changes

Whenever a user makes a change to the structure of a database it must be tracked so that when the developer distributes a new version of the structure the end user's changes can be integrated into it. The complexity of the tracking system depends on how much structure information the end user can modify. If all they are allowed to do is create new tables then the system can be as simple as recording those user added tables to a file. As the utilities grow more complex, more information must be tracked. For example, if the user is allowed to create relations, each relation must be tracked as well, and if they are allowed to delete tables that must be implemented in the tracking system.

One way to track structure changes is with and external file. Any type of file that can be written to and read and parsed when necessary can work. Other possible options include running both databases simultaneously and querying and comparing the SQL System Tables or tracking the changes in the database in a table designed to store that information.

## Testing for User Changes

Once a tracking system is set up the data stored must be tested. It can be tested every time the database is opened to ensure that the structure of the database is correct. This testing can be done simply by parsing the tracking document and comparing it to the actual database structure.

Databases that offer end users more options would have to have a more complex testing system. A new feature in 4D v11 SQL is the access to SQL System Tables.

There are six system tables which used together can return the necessary data to test the tracking document against. The system tables can be accessed via SQL statements. The tables available are as follows:

- _USER_TABLES: This table contains data about the tables in a database, including table number and name.
- _USER_COLUMNS: This table contains data about the fields in a database, including what table it is from, its name, number, and data type.
- _USER_INDEXES: This table contains data about the indexes in a database, including a reference number for the index, what table it is in, and what type of index it is.
- _USER_IND_COLUMS: This table contains more data about the indexes in a database and includes the reference number, so it can be matched to data retrieved from the _USER_INDEXES table. It includes information about the indexed field.
- _USER_CONSTRAINTS: This table contains data about the relations in a database, including a reference number for the relation, the type, what table it is in and what table it relates to.
- _USER_CONS_COLUMNS: This table contains more data about the relations in a table and includes the reference number, so it can be matched to data retrieved from the _USER_CONSTRAINTS table. It includes information about the related fields.

For more information on SQL System Tables see the "SQL Reference" document, which can be downloaded from the 4D website:

http://www.4d.com/support/documentation.html

The information is under the "Principles for Integrating 4D and the 4D SQL Engine" heading.

## Importing and Exporting User Created Data

When opening a data file, if the structure file does not contain the same tables as the data file, the data in non-corresponding tables is lost. To prevent this, data from user created tables must be exported by the user before updating to the new structure. Then, once the new structure has been updated to include the user created tables, the data can be imported back into the data file and the database is ready to use again.

## Integrating User Changes

When the structure is tested and discrepancies are found the database must be updated to include user created tables. The changes that must be made can be found in the tracking document or tracking table depending on what tracking method was used.

The process of integrating the user's changes can be summarized as follows:

1. Create a back up of the end user database.
2. Export all user-created data from the end user database.
3. Upgrade the structure to the current version.
4. Integrate the structure changes/user created tables into the new structure.
5. Import the user-created data.

Note that the choice of tracking system, technique used for import/export, etc. has little bearing on this process. This process should be adhered to regardless of the design specifics.

This process should only trigger when a new version of the structure is distributed by the developer. In other words there is no need to perform these steps every time the database is launched.

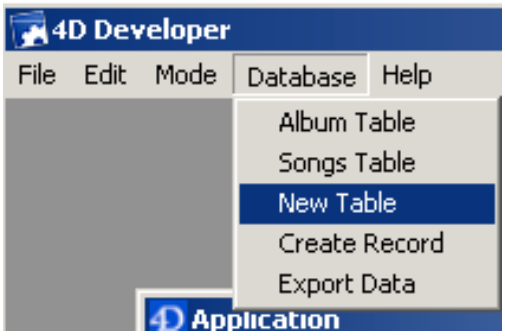
## The Sample Databases

In order to run through the update process described later in the document, it is a good idea to save a copy of this database as downloaded with the Technical Note. Open either the end user version of the database or the current version to explore. The only difference between the two versions is the database structure; some different tables and fields have been added to each version.

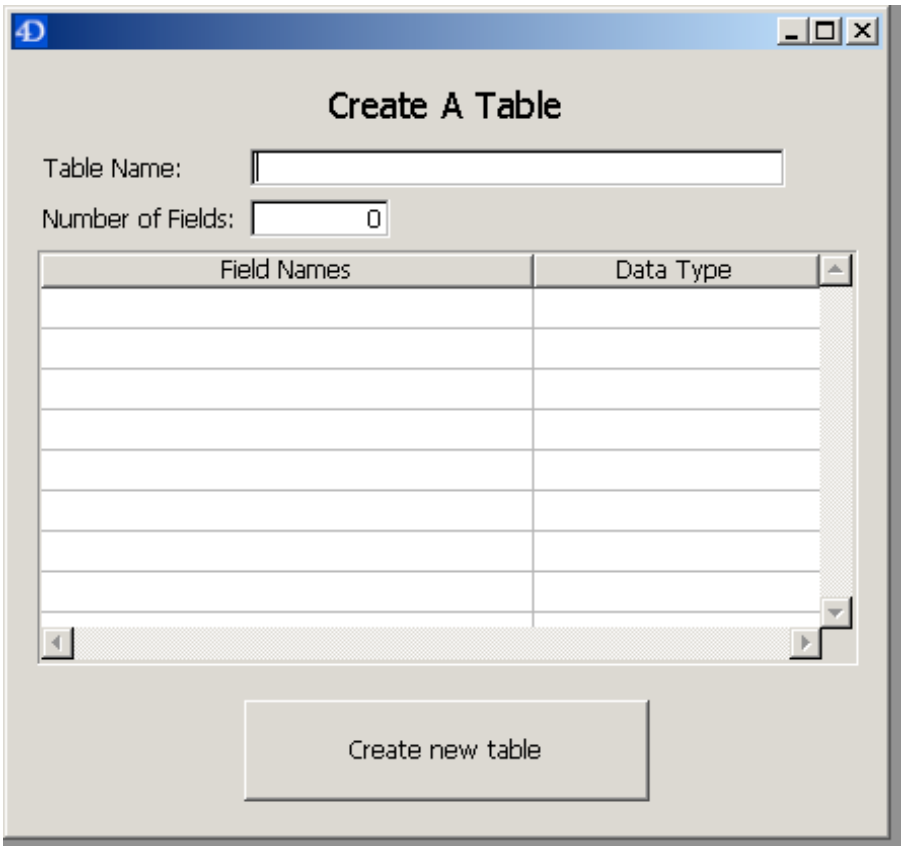
### User Created Tables

The option for end users to create tables is a 4D v11 SQL feature used by implementing SQL commands in 4D. This is done using the SQL command CREATE TABLE.

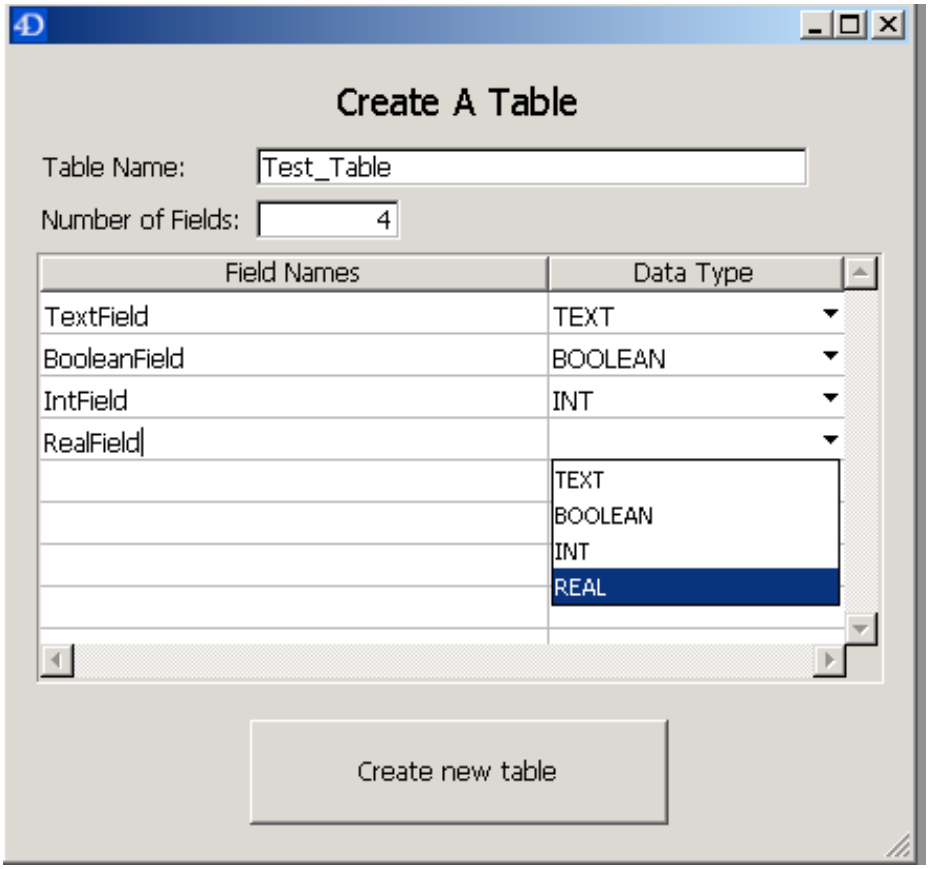
In the end user database choose the New Table menu item from the Database menu to create new tables, as shown below:




This opens the New Table form, as shown here:




In this form the end user can enter the table name and the number of fields they want in the table. Once those have been entered the list box for Field Names and Data Types is populated with blank values, allowing the user to enter the desired fields. This application supports Text, Int, Boolean, and Real fields, as shown in a drop down menu for each field the user would like to enter. An example is shown below:




Once the table and field data is entered the user can create the new table with the button at the bottom of the form. A warning is displayed giving the user the option to cancel and change their table and field data before continuing. This is important because all the information contained herein must be valid or else the code to create a new table will not work. The table name cannot contain certain characters (such as spaces) and the field names are limited as well (e.g. names such as "Text" are disallowed because Text is a reserved word).

## Creating a New Table

The Create new table button has two main functions.

As mentioned previously SQL commands are used to create a new table. The portion of the "Create new table" button object method regarding this is included below (variable names have been shortened to avoid breaking one line of code onto multiple lines in this document):

```
C_TEXT($crtbl)
$crtbl:="CREATE TABLE IF NOT EXISTS "+Table+" ("+Field{1}+" "+DataType{1}
For ($count;2;Fields)
    $crtbl:=$crtbl +","+Field{$count}+""+DataType{$count}
End for
$crtbl:=$crtbl+");"

ON ERR CALL("sql_err_handler")
```

```
Begin SQL
   EXECUTE IMMEDIATE :$crtbl;
End SQL
ON ERR CALL("")
```

Since the end user is allowed to name the table and fields, the SQL code to create the table becomes a dynamic statement. EXECUTE IMMEDIATE is used to run this statement, with different table names inserted based on the variables specified in the $crtbl text variable. This allows the developer to input 4D variables into the SQL code, which is not possible between the BEGIN SQL and END SQL commands otherwise.

Because of this a text variable is set up to store the SQL statement to create the new table. It is made using the table, fields, and data types as defined by the user on the form. This is used with the SQL command CREATE TABLE to create the new table. The SQL clause "IF NOT EXISTS" is used to the user from creating multiple tables with the same name.

Once the text variable ($crtbl) is set up it can be passed into the SQL command EXECUTE IMMEDIATE to create the table. The ON ERR CALL command is used to trap errors and warn the user if they have used an illegal table name, field name, or data type.

## Logging the New Table

As a developer it is important to track new tables created by the end user so that when a new version of the database is distributed the user created tables can be added to that new version as well. In the sample databases the tracking is done in an XML document where each new table's structure is stored (the structure_log.xml file in the database's Extras folder). The code can be found in the databases under the Update_XML method. This is only one way to track the changes to a database's structure, and thus only the general logic of it will be discussed in this Technical Note. There are numerous other means of tracking user added tables, and the decision of how to do it is up to the developer.

In this case each table is stored as a separate XML element and each field in a table is a child element of the table element. The field data types are stored as the value of each field element.

The sample databases only support users creating new tables, and thus that is the only function which this logging code implements. In other situations it would also be necessary as a developer to log table deletions or changes to table structures if these functions are available to the end user.

***Note*** *If you want to add the ability to delete tables you must also delete the elements created in structure_log.xml. If you do not every time the database is opened it will check that document as part of the On startup code and recreate those tables.*

More information on the DOM XML commands used in the Update_XML method can be found in the 4D online documentation at:

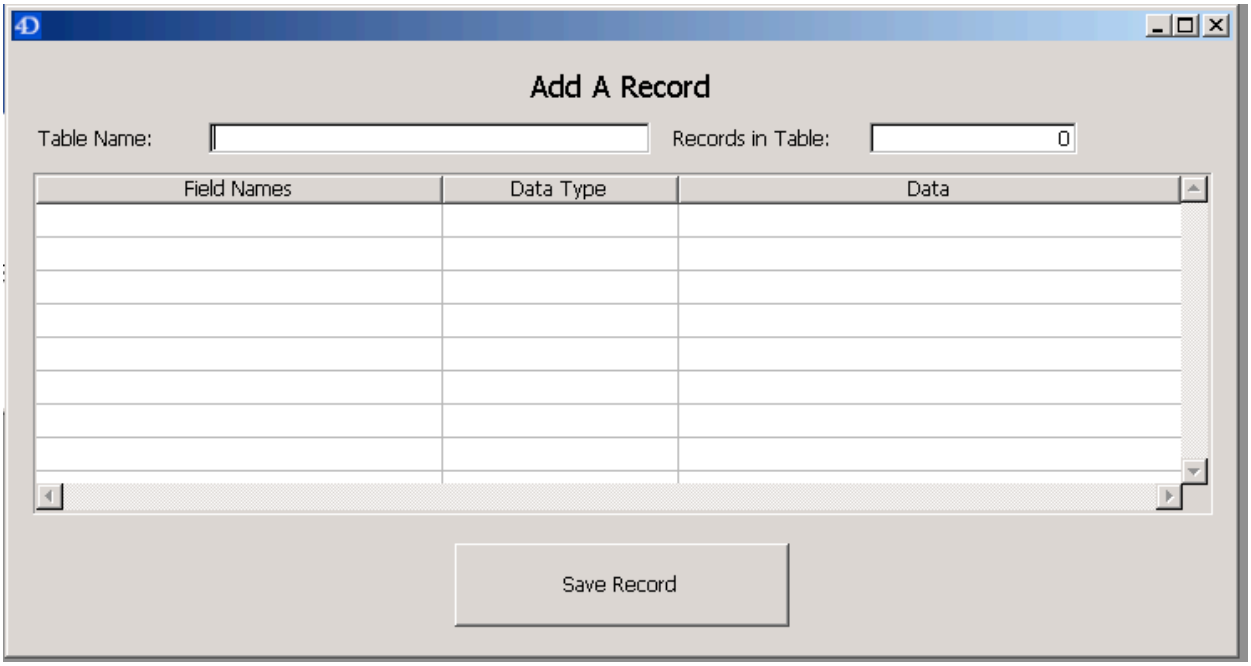
http://www.4d.com/docs/V6U/V6U00066.HTM

## Entering and Displaying Data

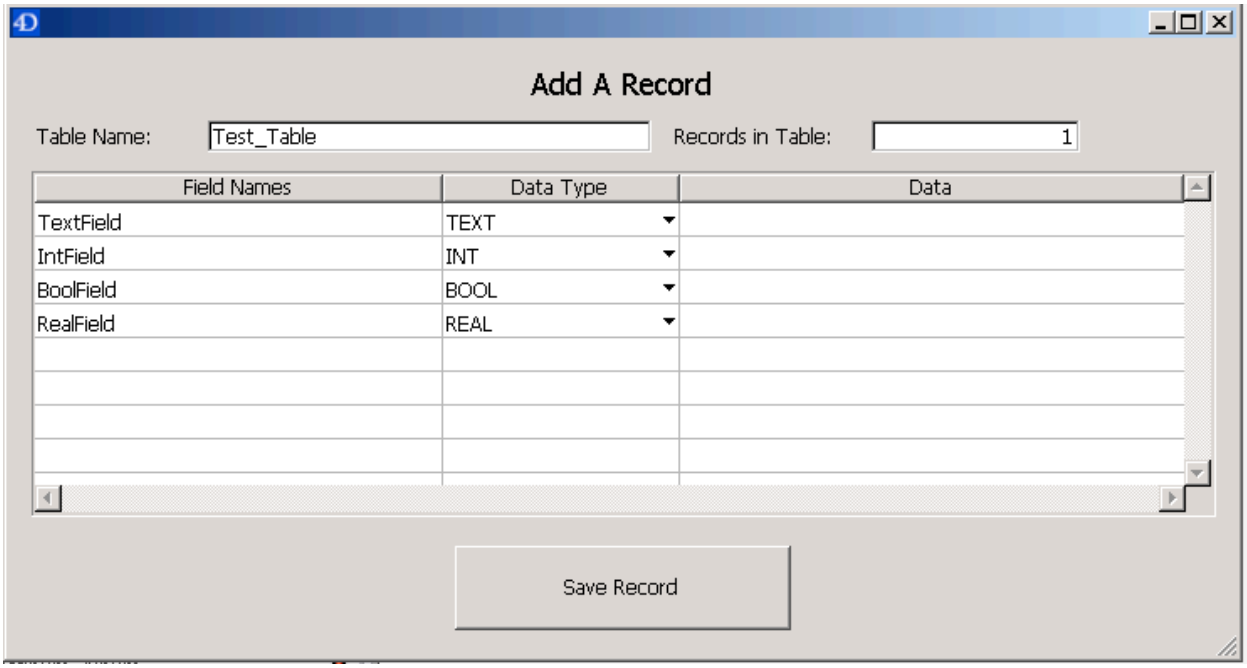
User created tables bring up another issue for developers in how to display and enter data into those tables. In the sample databases only the entering of data into these tables is handled, but it can be used as a blueprint for how to display it as well. This is a very simple data entry form and the more complicated options that are offered to end users, the more complicated the entry and display of data would be.

To enter data into a user created table the end user selects Create Record from the Database menu, as shown below:


This brings up the Add a Record form, as shown here:



To add a record the user must input the table name in the first field. This triggers the On data change form event in that object's method. First the text is checked against the table list to ensure that it is a valid table for the database. If so, then the Records in Table field is updated for the total number of records in that table and the list box is updated to show each field name and data type for that table. At this point the user can input the data. Carry on from the previous example, the form would appear as follows when they tab out of the Table Name field:




Once the data is entered the Save Record button can create a new record. Since the Data column of the list box stores a text array True or False must be entered for Boolean values and then parsed. Similarly any integers or real number values must be changed from text to the desired type before saving the record. The object method for the Save Record button does all of this. It also resets the list box and Records in Table process variables so another new record can be entered.

A similar list box form could be used as an output form for user created tables, as list boxes can be created dynamically in code to match the needs of each table the user creates.

## Database Update Scenario

There are three main parts of updating the database. Part one is to export data from user created tables. This must be done before the user opens the new structure. Part two is to read the structure information from wherever the developer chose to store it, in this case structure_log.xml, and use that to create new tables in the updated structure. Part three is to import the data from these user created tables back into the database.

## Sample Update

Here is a sample update to run before going through the three parts of updating a database in detail. These instructions are similar to those a developer would pass to the end user.

1. Create a backup of the end user database.
2. Open the end user version of the database with its current data file. Once in the database go to the Database menu and choose Export. This will export the data.
3. Quit out of the End User version of the database.
4. Go to the "Developer" folder and copy the .4db and .4Dindy files. Paste these into the End User database folder. This will overwrite the user structure files.
5. Open the updated database. Choose the corresponding data file when prompted to by 4D, if necessary. The structure will be updated with user created tables and the data from those tables will be imported automatically.

That five step process is all that is necessary to update to a new version of the database.

***Note:*** *In this example the database is "upgraded" by simply copying the new structure file over the old one. This is a somewhat contrived example of how to perform an upgrade. Certainly a more advanced/fail-safe process could be used (i.e. an installer) but the concept is unchanged by these details.*

To test that the upgrade was performed correctly:

- The tables created were "Test_Table_1" and "Test_Table_2". These may be used in the "Add a Record" form to ensure they were created
- The correct number of records, 2 for Test_Table_1, and 3 for Test_Table_2 should also exist.

## Exporting Data

How data import and export is done is ultimately the developer's decision. There are many options available. Here is a partial list:

- XML files
- Text files
- 4D format
- SEND RECORD and RECEIVE RECORD commands
- SOAP commands

In the sample database the SEND RECORD and RECEIVE RECORD commands are used. The code can be found in the sample database in the "Export" method.

The most important issue to note is that the developer must make their end user export data before updating their database structure to the newest version. When

opening the End User version of the data file with the updated structure all data from user created tables will be lost. This means that if the user did not export their data and does not have a backup file all of that data will be permanently lost.

On a similar note this is why it is important that end users follow the first step and backup the database before updating to a new version from the developer.

## Updating the Structure

This process is similar to that of creating new tables when the end user inputs the data, but instead the structure is read from the tracking system the developer implemented in his or her database. In the sample database the structure_log.xml file is used.

All the code for this can be found in the "On startup" method for the sample database and the methods it calls as sub-routines. Every time the database is started a Boolean interprocess variable, <>upgrade_flag, is set to false. Then the "Check_XML" method is called. This method compares the structure of the database to the structure in the XML tracking file and updates it as needed. We will go through some of the code in that method here.

To start, an array of table names and numbers is created. This code is repeated often throughout the database, as it is commonly useful to check to see if a table name read in from an external source or input by the end user is valid or already exists in the database. Here is that code:

```
For ($thetable;1;Get last table number)
   If (Is table number valid($thetable))
          APPEND TO ARRAY($table_array;Table name($thetable))
   Else
          APPEND TO ARRAY($table_array;"")
   End if
End for
```

This code is simply gets the table name for each table number and stores table names sequentially in an array. This is so that later in the process the array can be searched for a specific table name, and the name's position in the array corresponds with that table's number in the database.

The next part of the code deals with reading the XML file to find table names and table structures. This part will be different depending on how the developer chose to track user created tables. It is important that, as each table is read in, it is tested against the array of table names that was created. The code that checks this is run every time the database starts, so all the table creation code hinges on whether or not the table name was found in the array. If the table described in the tracking mechanism is not already in the database, then a new table is created.

The table creation code is as follows (once again variable names have been shortened to keep single lines of code from breaking into multiple lines in this document):

```
$crtbl:="CREATE TABLE IF NOT EXISTS "+$table+" ("+$field{1}+" "+$data{1}
For ($count;2;Size of array($field))
    $crtbl:= $crtbl+", "+$field{$count}+" "+$data{$count}
End for
$crtbl:= $crtbl+");"
Begin SQL
    EXECUTE IMMEDIATE :$crtbl;
End SQL
<>upgrade_flag:=True
CLEAR VARIABLE($field)
CLEAR VARIABLE($data)
```

The SQL code here is fully described in the previous section of this Technical Note, "Creating New Table". The main difference is that the arrays that the field names and data types are stored in are created as they are parsed from the structure_log.xml file. The table creation code is run as the XML file is parsed, after each table's structure information has been stored in the arrays, so each array is cleared after the table is created. This way the method can continue parsing the XML file and add another table if there are multiple user created tables.

Also note that the "<>upgrade_flag" variable is set to True if a new table is created. Setting this flag triggers the next part of the code from the On startup database method; if <>upgrade_flag was set to true, then the "Import" method is called to import data.

### Importing Data

Exported data is automatically imported at startup in the example database.

The intricacies of importing and exporting data are not within the scope of this document. How to import and export data is a decision to be made by the developer.

In the sample database SEND RECORD and RECEIVE RECORD are used and the import procedure can be found in the Import method.

## Conclusion

This Technical Note went through a process a developer could use to upgrade a database that allows end users to create tables. How the upgrade process is designed will be dependent on the development of the database, and thus it must be planned from the start. It is important to take into account many different ramifications when allowing users this kind of freedom, but since all the user utility must be implemented by the developer each developer will know what issues must be handled for a particular application.