Data Synchronization

Presented by: **Dani Beaubien**

INTRODUCTION

Data synchronization means different things to different people. For our purposes, data synchronization is the process of one or more independent data sources sharing changes so that each data source is an exact copy of the other sources. This can be done in real-time or near-real-time. Data should flow between the various data sources and be flexible enough to handle unexpected anomalies in the network topology as well as infrequent and sporadic synchronization.

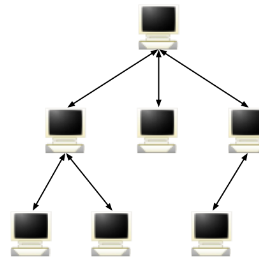
Data synchronization is not appropriate for all applications, but in the correct situation it can be extremely valuable. Situations like where your clients have their users spread out across a WAN or are connected sporadically. Another extremely valuable application is to use data synchronization to distribute users across a number of 4D Servers while keeping all the data accurate and up to date. As well, an additional “backup” server can be kept ready incase of catastrophic failure.

This article goes through the basics of how I have implemented data synchronization in previous projects. This is a real technology that is in place and has been used continuously for the last 4 years by hundreds of users. I have gone through a lot of refinements that have made vast improvements to how the data synchronization process works. I have supplied some code that you can use as a starting point for your own solutions. Feel free to utilize anything that you find in this article. All I ask for in return is a bit of credit and perhaps some feedback on how this has worked for you. You can contact me directly at dbeaubien@myballteam.ca

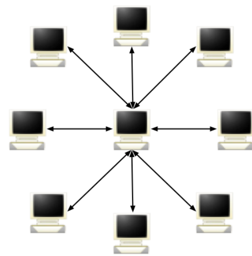
NETWORK TOPOLOGIES

The topology of the data synchronization network that you want to create will affect how you implement the solutions. Networks can be broken down into a number of classifications:

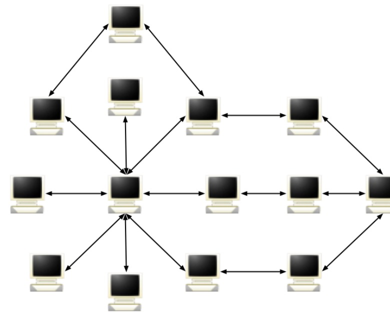
Line



Traditional Hierarchical



Star



Network

In fact, there are only two. The line is a special case of the hierarchical where each parent has, at the most, one child node. The star is also a special case of the hierarchical where there is only one parent and that parent has all the other nodes as a child. The "Network" topology is essentially a hierarchical network with one or more loops. The advantage of the Network topology is that is handles failures quite well. In the Network topology diagram above, of all the machines shown, only 4 are vulnerable to a single failure, the others require, at least, two failures to be isolated.

POTENTIAL ADVANTAGES & DISADVANTAGES

What ever the reasons are for wanting to implementing data synchronization, there are a number of advantages and disadvantages that we should be aware of. Each of the

Advantages

Remote copies – Traveling or remote users can have access to the all the information and data that they are used to accessing when they are in the office.

Scalability – Provides a way of scaling for very large sites (i.e. 200+ users across 4 servers).

Reduce Hardware Costs – Spread the load amongst many smaller/cheaper servers or standalones.

Improved Application Speed – Standalone 4D applications respond faster than 4D Client/Server applications.

Enhance Data Security by Only Sharing Minimal Data – Build the application so that only the data that the user has access to is actually on their machine. This can be a very powerful advantage. Data files are smaller.

Easier Installs – Can install an application with an empty data file and it gets all it's data from the rest of the "system". Extra care needs to be taken to prevent unauthorized access to the rest of the data. This is a very powerful feature; the implication is that if a user's computer has a failure then it is much easier to rebuild their application and all the data contained therein.

Improved Stability / Failure Resistant – A catastrophic failure doesn't take out the entire system since data is replicated throughout the entire network. If the topology is designed with this in mind, then the disruption can be minimized.

Disadvantages

Reduce Physical Security of Data – Less data security since it physically exists on many machines. Depending on the application, this might or might not be a consideration. Extra care can be taken to reduce this risk by encrypting parts of the data.

More Difficult Updates – Potentially more complicated to deploy updates since the application exists on many machines. An auto-update feature helps to reduce the impact of updates.

Additional Time to Perform Synchronizations – No matter what, synchronizing takes time, the more data that must be moved, the longer the process takes.

More Complication Configuration – Configuration is more involved and potentially requires more planning.

Version Control Issues – Version control becomes more important. All users in the entire system must be on the same version. Once again, an auto-update feature helps to reduce the impact of this issue.

Time Zones – If your users travel, them time zones need to be considered. An accurate clock is very important.

Unique Record IDs – Since the data is distributed, that means that record creation happens on many different machines. Ensuring that records IDs are truly unique must be given some additional consideration.

<u>Unique Database IDs</u> – Each database must be uniquely identified. The system needs to have a sense where data is coming from. There could be issues if data files are copies and shared, each DB needs a unique identifier.

OVERVIEW OF THE SOLUTION

How I have implemented Data Synchronization is through triggers. Each time a record is touched (either through saving or deleting), the trigger fires and updates a record in the `[_Sync_RecordTracker]` table. There will be exactly one of these records for every record in every table in the database that will be sync'd with other data sources.

When a client synchronizes, then it searches the `[_Sync_RecordTracker]` table for any records that the `Event_DTS` is on or after the `[_Sync_ServerTracker]Last_UpdateDTS` that is recorded for that server and sync type. Then it sends to the server, from oldest to newest, all the changes that it has on record. When it is time for the client to receive changes from the server, it passes on the `[_Sync_ServerTracker]Last_UpdateDTS` that is recorded for that server & sync type. There are four sync types:

- Get Deletes from sync Server
- Send Deletes to sync Server
- Send Changes to sync Server
- Get Changes from sync Server

Normally, there will be four `[_Sync_RecordTracker]` records per server, one for each sync type. Each record will have it's own Last_UpdateDTS which represents the DTS of the start of that particular phase of the sync cycle. This is very important since the server will have other active synchronization sessions going on while the client is syncing. This way, no changes get missed. They might not go during the active session, but they will on the next.

A typical sync session will go through the 4 sync types in that order. The delete all happen first so that those get out of the way as quickly as possible and the server might want to send a change back that the user has deleted. We want those deletes to happen first. The changes are sent to the server first so that the server can do some rudimentary duplicate record detection (from two different users adding the same record at the same time and then syncing it into the server). If a duplicate is detected, the server can send back to the client a request to change the record ID to one that it already has. Finally any changes are received from the sync server.

To cover clock discrepancies, there are a few defensive measures that I have implemented. One is that part of the initial application authentication that happens between the client and the server is that the server verifies that the current date and time is within 3 minutes of the server. If it is not, then the server rejects the connection. The other measure that I have implemented is that I always backdate the DTS that I record in the `[_Sync_ServerTracker]` table 6 minutes, just to make sure that when I am sending a DTS to the server to ask for deletes/changes, that it will cover the +- 3 minutes that the other clients might be from me.

ADDITIONAL TABLES TO TRACK INFORMATION

To implement Data Synchronization, there are a few changes that are required to your structure. A few tables need to be added, a few fields need to be added to each table that you want to synchronize as well as triggers.

_Sync_RecordTracker	
Rec_GUID	A30
TableNumber	L
Rec_DTS	A14
Event_DTS	A14
Event_Code	I
Student_GUID	A30
fromDB_GUID	A30
RecordNumber	L

_Sync_ServerTracker	
Last_IP_Addr	A20
Last_UpdateDTS	A14
Server_DB_GUID	A30
User_Key	A30
Sync_Type	L

The two tables that need to be added are `[_Sync_RecordTracker]` and `[_Sync_ServerTracker]`. The record tracker table is responsible for holding the last time each record, in the entire database, was modified and from which database it came from. The server tracker table keeps track of the synchronizations that have already happened.

All the DTS (date-time-stamp) fields use "yyyymmddhhmmss" to record the date and time. I have some special routines that I use to quickly convert the current time to that format. The one advantage is that the table can be sorted on that field to get the chronological order of the DTS values. As an aside, there is some code that exists that does the same thing but in a longint. I would suggest that you use a longint rather than the A14. My experience is that longint indexes do not corrupt as often as an alpha index.

Due to the demands for a globally unique record ID (GUID) I am using an A30 field to hold the ID. Check out the supplied code in the appendix for the code for the GUID_CreateNew method that I use to produce the GUID. It has worked very well and doesn't require a plug-ins to retrieve the machine's MAC address.

The Student_GUID field is a special purpose field so that the server can discriminate what data to send the remote clients based on the currently logged in user. This application that I am using as a basis for this article is utilized in the school systems. Users are assigned access to particular students and only have access if they have been assigned. This is part of the privacy. The server can use that information to reduce the amount of data that need to go to the remote client to just the records that belong to students that they have access to AND any data that doesn't belong to any student. There was a huge benefit in implementing this since most of the remote users are only seeing a small subset of the total data. Data security is enhance since the user only has the data that they have access to and the synchronization process is much faster since there is less data to move around.

In the [_Sync_RecordTracker] table there are two very similar fields, Rec_DTS and Event_DTS. Rec_DTS is the DTS value from the record and Event_DTS is the DTS of when this record tracker record was updated. The Event_Code field is either a 1 (Save) or a 2 (Delete). On the sync client, the Rec_DTS and Event_DTS will be the same or very close, but the differences will be on the sync server. When the server receives a change from a remote client the DTS that the server records the change will vary greatly from the DTS that is recorded in the record. The client might have sync'd hours after the record change happened. The Event_DTS is used to determine when that copy of the software received a change.

The fromDB_GUID field records the unique identifier of the data source that this change came from. The is used to prevent an "echo" happening with sharing changes. An example will illustrate the issue: sync client (with a dbGUID of "1") opens a connection with a sync server. The client asks for all the deletes from the server. 4 are received. The sync client proceeds to delete those 4 records and updated the appropriate [_Sync_RecordTracker]Event_DTS fields. Now the client searches for what deletes to send since the last time it sent deletes. The 4 deletes that is just received will get picked up in this search since their event DTS is recent. The only way to detect this is to record, in the [_Sync_RecordTracker]records, which dbGUID caused the change to happen.

You will notice that in the record tracker field I am tracking the record number and the record GUID (along with the table number). I do this for a huge speed improvement when I am searching for the record that I want to update or send to the other side. I have a nice routine that will perform the search either using the record number or by query depending on what is most valid. That code is in the Appendix as well.

ADDITIONAL FIELDS IN EACH TABLE

There are three fields that need to be added to each table that will be sync'd. These fields are:

- _DTS_Modified (Alpha 14)
- _checksum (Longint)
- _skipDTS (Boolean)

A big hurdle that I had to overcome came when I had the realization that most of my users, when closing a record just hit the save button regardless if they have made a change or not. That meant that there were considerably more "changes" that they sync system want to share than there really were. My solution to this was the _checksum field that holds a CRC checksum. The old function would only work for fixed fields which meant the only way to detect a change in the variable length fields (text, pictures, blobs) is by using a checksum. Then when a record is "saved", I can scan all the fixed length fields seeing if the old version of that field is different than the current version of the field; I build a checksum and compare to what is stored. If it is the same, then the record has not been changed. Simple, clean and works very well. Check out the Table_RecordHasBeenModified method that I use in the Appendix.

If the record has been modified, then update the _DTS_Modified field to hold the current DTS as well as update that record's [_Sync_RecordTracker] record. If the _skipDTS field was set, then do not update any of the DTS fields, just update the checksum. There are two situations where you don't want the record tracker record update, one is when you are doing data maintenance that is part of a software update data conversion, the other is when the application is receiving a change through the sync system. The sync system is in the process of updating the record tracker record and the trigger could potentially mess things up. It is easier to have a way to signal the trigger to not record this one since you already have it covered.

TRIGGERS

Now we are getting to the place that can make or break the solution. Extreme care must be taken to minimize the impact that any triggers have on the normal moment to moment operation of your application. The triggers have one job; the first is to determine if the record has changed/deleted. The second is to update the `[_Sync_RecordTracker]` record appropriately. If all the applications are standalones, then triggers have less impact. If you are using 4D Servers that need to server many users, then more care needs to be taken as to how the triggers are written. You might want to consider adding an extra field called `_recordNumber` that is the record number of the `[_Sync_RecordTracker]` record. Then in the trigger you can use that field to "try" to quickly locate the appropriate tracker record to maintain. Obviously the record numbers can change so that has to be managed as well. The `Table_LoadByRecNoOrID` method that I have supplied below can be used for that. The `Table_RecordHasBeenModified` method should also be modified to ignore that field.

For each table that I want to be synchronized, I add the following line into the trigger: `Sync_TriggerMethod (->[Students];->[Students]Student_ID)`. This line is from a `[Student]` table. I like to use a common method like that contains all the code. This makes it much easier to make modifications and improvements to all the triggers at one time. Obviously only the sync trigger related code is there. The code for this method and some of the methods that this uses are included in the Appendix.

MERGING THE A CHANGES BACK INTO THE DATABASE

So now, we are tracking all the changes that the users are making. All the deletes are being tracker. There are only two items left: the actual sync client/server communications and how to "merge" any received changes into the database.

The biggest trick with merging the changes into the database is to do what you want without causing any received change to be recorded as a new change, it must be recorded as an old change that has been recently received.

The following code is an "example" of the code that I use to manage the merge of an already created (but not saved) record and determine if there is an older copy in the database.

```
If ($tablePtr=(->[Students]))
	$keep_GUID:=[Students]Student_ID
	$importingModDateDTS:=[Students]_DTS_Modified
	$saveImportedRec:=True

	PUSH RECORD([Students])
	Table_LoadByRecNoOrID (->[Students];$recordNumber;-
>[Students]Student_ID;$keep_GUID)
	If (Records in selection([Students])>0)
		If ($importingModDateDTS>[Students]_DTS_Modified)
			READ WRITE([Students])
			LOAD RECORD([Students])
			[Students]Student_ID:="DEL."+String(Table($tablePtr))   `
need to change ID so sync is okay
			SAVE RECOR($tablePtr)
			DELETE RECORD([Students])
		Else
			$saveImportedRec:=False
		End if
	End if
	POP RECORD([Students])

	If ($saveImportedRec)
		[Students]_skipSync:=True
		SAVE RECORD($tablePtr)
	End if
End if
```

PUTTING IT ALL TOGETHER BY DO THE SYNCHRONIZATION

I am not going to cover the Synchronization Client/Server communication cycle in this article. That will be covered that in the session at the summit. I will illustrate the finite state machine that the Synchronization Server is based on

and the structure of the XML that I use to communicate. Come to the session for the 2nd half of Data Synchronization.

APPENDIX – DANI'S BIO – A BIT OF SHAMELESS SELF PROMOTION ;-)

Dani is a lead developer and development supervisor for Jonoke's development department as well as providing continued expert consulting/contracting to other 4D development organizations. He has over 15 years experience building specialized commercial database applications for vertical markets using 4D and 4D on the web. Specializing in enhancing products through improved usability, Dani fully understanding how the users view software and what their priorities are. He is always focused on the "big picture" and delivers ever improving products. He has developed many innovative solutions such as custom XML SAX parser, 4D based web shell, auto updating of 4D applications, data synchronization using TCP/XML, on-line help using SOAP, drug/allergy interaction checker using SOAP, integrating independent EHR using TCP/HTML/HTTP/XML/HL7 and many others.

His broad technical foundation started with two Bachelor of Science degrees from the University of Alberta, one in Mathematics and the other in Computing Science. He has worked every aspect of the software development process -- from requirements gathering, design, development, mentoring, training, direct customer support, to project and staff management. He prides himself on his creativity and ability to diagnose and problem solve. Through out his entire career, he has worked closely with the end user and is able to communicate clearly in a way that is atypical of software developers.

APPENDIX – MISC CODE

`TIME_Get_Hours`, `TIME_Get_Minutes` and `TIME_Get_Seconds` are public domain methods that are available by search the NUG. The `Date2String` method is a method that I use to convert a date to a specified format, if you want it, email me and I will send it. What follows is some of the routines that I use.

```
   ` ----------------------------------------------------
   ` METHOD: GUID_CreateNew
   `
   ` DESCRIPTION:
   `   This method returns a globaly unique number based on the
   `   current date and time, an internal counter and the user id of the
   `   current user. This should be a unique numbe through out the system.
   `
   ` ASSUMPTIONS:
   `   none
   ` ----------------------------------------------------
   ` PARAMETERS:
   `   none
   `
   ` RETURNS:
C_STRING(30;$0;$guid)
   ` ----------------------------------------------------
   ` HISTORY:
   `   Created by Dani Beaubien
   ` ----------------------------------------------------

C_LONGINT(<>guid_LastNumberGiven)

   ` build our guid, get the date and time stamp
$guid:=Date2String (Current date;"YYYYMMDD")+":"
$guid:=$guid+String(TIME_Get_Hours (Current time);"00")  `hour
$guid:=$guid+String(TIME_Get_Minutes (Current time);"00")  `minute
$guid:=$guid+String(TIME_Get_Seconds (Current time);"00")  `second

   ` make sure that it is returning a unique number.
   ` If called one after another in a compiled database, it might
   ` return the same #

   ` now add the rest
```

```
$guid:=$guid+":"+String(<>guid_LastNumberGiven;"000")+":"
$guid:=Substring($guid+CurrentUser ;1;30)

  ` store this for the next time we are called
<>guid_LastNumberGiven:=<>guid_LastNumberGiven+1
If (<>guid_LastNumberGiven>999)
      <>guid_LastNumberGiven:=1
End if

  ` return it
$0:=str_StripInvisibles ($guid)



  ` ----------------------------------------------------
  ` METHOD: Table_LoadByRecNoOrID
  `
  ` DESCRIPTION:
  `   This method loads a particular record by record number, if it can
  `   other wise by the ID passed it.
  `
  ` ASSUMPTIONS:
  `   none
  ` ----------------------------------------------------
  ` PARAMETERS:
C_POINTER($1;$tablePtr)
C_LONGINT($2;$recordNumber)
C_POINTER($3;$uniqueFieldPtr)
C_TEXT($4;$guid)
  `
  ` RETURNS:
  `   none
  ` ----------------------------------------------------
  ` HISTORY:
  `   Created by Dani Beaubien
  ` ----------------------------------------------------

$tablePtr:=$1
$recordNumber:=$2
$uniqueFieldPtr:=$3
$guid:=$4

REDUCE SELECTION($tablePtr->;0)

  ` Try direct, if we can
If ($recordNumber>0) & ($recordNumber<18000000)
      GOTO RECORD($tablePtr->;$recordNumber)
End if

  ` Try just by the index, if we can
If ($uniqueFieldPtr->#$guid)
      $recordNumber:=Find index key($uniqueFieldPtr->;$guid)
      If ($recordNumber#-1)  ` If this name has already been entered
            GOTO RECORD($tablePtr->;$recordNumber)
      End if
End if

  ` All else fails, do it the old way
If ($uniqueFieldPtr->#"") & ($uniqueFieldPtr->#$guid)
```

```
      Log_AddDatedEntry ("*** BAD INDEX *** - Find index key failed for
["+Table name($tablePtr)+"]"+Field name($uniqueFieldPtr)+" with value
'"+$guid+"', '"+($uniqueFieldPtr->)+"' was found instead.")
      QUERY($tablePtr->;$uniqueFieldPtr->=$guid)
End if

  ` if nothing found, the make sure nothing is found.
If ($uniqueFieldPtr->#$guid)
      REDUCE SELECTION($tablePtr->;0)
End if



  ` ----------------------------------------------------
  ` METHOD: Table_RecordHasBeenModified
  `
  ` DESCRIPTION:
  `   Goes through all the fields in the DB and determines if the
  `   record has been modified or not.
  `
  `   NOTE: This method will also update the checksum of the record
  `   if it has been changed.
  `
  ` ----------------------------------------------------
  ` PARAMETERS:
C_POINTER($1;$tablePtr)
C_POINTER($2;$checksumPtr)
  `
  ` RETURNS:
C_BOOLEAN($0;$hasBeenModified)
  ` ----------------------------------------------------
  ` HISTORY:
  `   Created by Dani Beaubien
  ` ----------------------------------------------------

$hasBeenModified:=False

If (ASSERT_PARMCOUNT (Current method name;2;Count parameters))
      $tablePtr:=$1
      $checksumPtr:=$2

      C_BLOB($dataBLOB)  ` holds the data to do the checksum on
      SET BLOB SIZE($dataBLOB;0)

      C_LONGINT($tableNo)
      $tableNo:=Table($tablePtr)

      C_LONGINT($fieldNo)
      $fieldNo:=1

        ` figure out which field positions are these three fields.
      C_LONGINT($skip_FieldNo;$modDTS_FieldNo;$checksum_FieldNo)
      $skip_FieldNo:=<>_syncFieldPos_skipDTS{$tableNo}
      $modDTS_FieldNo:=<>_syncFieldPos_DTS_modified{$tableNo}
      $checksum_FieldNo:=<>_syncFieldPos_checksum{$tableNo}

      While ($fieldNo<=Count fields($tablePtr))
            $fieldPtr:=Field($tableNo;$fieldNo)
            $fieldType:=Type($fieldPtr->)

              ` check for changes
```

```
            Case of
                  : 
($fieldNo=$skip_FieldNo) | ($fieldNo=$modDTS_FieldNo) | ($fieldNo=$checksum_Fi
eldNo)
                          ` ignore these fields.

                  : ($fieldType=Is Picture )
                        C_PICTURE($aPicture)
                        $aPicture:=$fieldPtr->
                        VARIABLE TO BLOB($aPicture;$dataBLOB;*)

                  : ($fieldType=Is BLOB )
                        COPY BLOB($fieldPtr->;$dataBLOB;0;BLOB
size($dataBLOB);BLOB size($fieldPtr->))

                  : ($fieldType=Is Text )
                        TEXT TO BLOB($fieldPtr->;$dataBLOB;Text with length
;*)

                  : ($hasBeenModified)
                          ` need to still loop through all fields to get an
acurate checksum below

                  Else
                        If (Old($fieldPtr->)#($fieldPtr->))
                              $hasBeenModified:=True
                        End if
            End case

            $fieldNo:=$fieldNo+1
      End while   ` (Modified record($tablePtr->))

        ` get a checksum on the dataBLOB
      C_LONGINT($checksum)
      $checksum:=0
      If (BLOB size($dataBLOB)>0)
            $checksum:=CRC_Calculate_onBLOB ($dataBLOB)
      End if

        ` check to see if the table's checksum is different that what we have
      If (Not($hasBeenModified))
            If ((Field($tableNo;<>_syncFieldPos_checksum{$tableNo})-
>)#$checksum)
                  $hasBeenModified:=True
            End if
      End if

      $checksumPtr->:=$checksum
      SET BLOB SIZE($dataBLOB;0)

End if   ` ASSERT

$0:=$hasBeenModified
```

APPENDIX – TRIGGER CODE

This is the method that is called by each trigger.

```
  ` ----------------------------------------------------
  ` METHOD: Sync_TriggerMethod
  `
  ` DESCRIPTION:
  `   This is a generic method that is called by every table's trigger
  `   that wants to be included in the synchronization system.
  `
  ` ----------------------------------------------------
  ` PARAMETERS:
C_POINTER($1;$tablePtr)
C_POINTER($2;$guidFieldPtr)
  `
  ` RETURNS:
  `   none
  ` ----------------------------------------------------
  ` HISTORY:
  `   Created by Dani Beaubien
  ` ----------------------------------------------------

$tablePtr:=$1
$guidFieldPtr:=$2

  ` Need to mark if we are in sync or not.
C_BOOLEAN(◊inSyncTrigger;$current_inSyncTrigger)
$current_inSyncTrigger:=◊inSyncTrigger  ` just in case we are cascading
changes
◊inSyncTrigger:=True

If ($guidFieldPtr->#"DEL.@")  ` Bypass the sync stuff entirely if the ID
starts with "DEL."
      Case of
            : (Database event=On Saving New Record Event ) | (Database
event=On Saving Existing Record Event )
                  Sync_SaveRecordEvent ($tablePtr;$guidFieldPtr)


            : (Database event=On Deleting Record Event )
                  If (Not(◊disableSyncTrigger))
                        Sync_TrackDeleteRecEvent ($tablePtr;$guidFieldPtr->)
                  End if
      End case
End if

◊inSyncTrigger:=$current_inSyncTrigger
```

This method contains references to a sync log table. I use it to track some statistics during the sync process. So if the client or the server are actively syncing, then there will be a sync log record loaded and in read-write mode.

```
  ` ----------------------------------------------------
  ` METHOD: Sync_TrackDeleteRecEvent
  `
  ` DESCRIPTION:
  `   This method updates a "Delete Event" in the Sync record
  `   tracker table
  `
  ` ----------------------------------------------------
```

```
  ` PARAMETERS:
C_POINTER($1;$tablePtr)
C_STRING(40;$2;$recGUID)
  `
  ` RETURNS:
  `   none
  ` ----------------------------------------------------
  ` CALLED BY:
  `   Sync_TriggerMethod
  ` ----------------------------------------------------
  ` HISTORY:
  `   Created by Dani Beaubien
  ` ----------------------------------------------------

$tablePtr:=$1
$recGUID:=$2

C_LONGINT($tableNo)
$tableNo:=Table($tablePtr)

READ WRITE([_Sync_RecordTracker])

  ` find the matching sync tracking record (if there is one)
Table_LoadByRecNoOrID (->[_Sync_RecordTracker];0;-
>[_Sync_RecordTracker]Rec_GUID;$recGUID)
If ([_Sync_RecordTracker]TableNumber#$tableNo) & (Records in
selection([_Sync_RecordTracker])#0)
      QUERY([_Sync_RecordTracker];[_Sync_RecordTracker]Rec_GUID=$recGUID)
End if
If (Records in selection([_Sync_RecordTracker])>1)
      QUERY
SELECTION([_Sync_RecordTracker];[_Sync_RecordTracker]TableNumber=$tableNo)
End if

  ` if no record, then create one
If (Records in selection([_Sync_RecordTracker])=0)
      CREATE RECORD([_Sync_RecordTracker])
      [_Sync_RecordTracker]Rec_GUID:=$recGUID
      [_Sync_RecordTracker]TableNumber:=Table($tablePtr)
End if

  ` if the record DTS has changed, then update the tracker record
If ([_Sync_RecordTracker]Event_Code#2)  ` not already deleted
      [_Sync_RecordTracker]Event_DTS:=str_GenerateDateTimeStamp
      [_Sync_RecordTracker]Event_Code:=2  ` saved action

        ` take advantage of a trigger trick
        ` if this record is loaded then sync is active and is saving this
record
        ` record that it came from the other machine
        ` This record would only be loaded if it by the sync process
      [_Sync_RecordTracker]fromDB_GUID:=""
      If (Records in selection([_Sync_Log])>0)
            If (Is record loaded([_Sync_Log]))
                  [_Sync_RecordTracker]fromDB_GUID:=[_Sync_Log]otherDB_GUID
            End if
      End if
End if

[_Sync_RecordTracker]RecordNumber:=0  ` clear this since the record is deleted
```

```
If (◊LogLevel>0)
	If ([_Sync_RecordTracker]fromDB_GUID="")
		Log_AddEntry ("SyncDeleteRecEvent:: ["+Table name($tableNo)+"]
deleted with GUID:"+$recGUID+", RecDTS:"+[_Sync_RecordTracker]Rec_DTS+",
EventDTS:"+[_Sync_RecordTracker]Event_DTS)
	Else
		Log_AddEntry ("SyncDeleteRecEvent:: otherDB_GUID:
"+[_Sync_RecordTracker]fromDB_GUID+", ["+Table name($tableNo)+"] deleted with
GUID:"+$recGUID+", RecDTS:"+[_Sync_RecordTracker]Rec_DTS+",
EventDTS:"+[_Sync_RecordTracker]Event_DTS)
	End if
End if

  ` track some statistics
If (Is record loaded([_Sync_Log])) & ([_Sync_RecordTracker]Rec_GUID#"DEL.@")
	[_Sync_Log]my_deletedCount:=[_Sync_Log]my_deletedCount+1
	SAVE RECORD([_Sync_Log])
End if


SAVE RECORD([_Sync_RecordTracker])
UNLOAD RECORD([_Sync_RecordTracker])
READ ONLY([_Sync_RecordTracker])



  ` ----------------------------------------------------
  ` METHOD: Sync_SaveRecordEvent
  `
  ` DESCRIPTION:
  `   This method might be called from a trigger or might be called from
  `   the "SaveRecord_TrapError" method.
  `
  `   if called, not from a trigger, then the check sum will be stored as a
  `   negative number. This is a signal to the trigger to not bother with
  `   the rec mod calc cod
  `   Once the trigger has run, it will correct the negative number though.
  `
  `   This is done to minimize the time spent in the trigger.
  `   But it will never be missed. This code MUST be run everytime a
  `   record is saved.
  `
  ` ----------------------------------------------------
  ` PARAMETERS:
C_POINTER($1;$tablePtr)
C_POINTER($2;$guidFieldPtr)
  `
  ` RETURNS:
  `   none
  ` ----------------------------------------------------
  ` HISTORY:
  `   Created by Dani Beaubien
  ` ----------------------------------------------------

$tablePtr:=$1
$guidFieldPtr:=$2

C_BOOLEAN($hasBeenModified)
C_LONGINT($tableNo)
$tableNo:=Table($tablePtr)
```

```
C_POINTER($skipDTS_fieldPtr;$checksum_FieldPtr;$modDTS_fieldPtr)
$skipDTS_fieldPtr:=Field($tableNo;◊_syncFieldPos_skipDTS{$tableNo})
$checksum_FieldPtr:=Field($tableNo;◊_syncFieldPos_checksum{$tableNo})
$modDTS_fieldPtr:=Field($tableNo;◊_syncFieldPos_DTS_modified{$tableNo})

If (Not(◊disableSyncTrigger)) & ($guidFieldPtr->#"DEL.@")

        ` check to see if the record has been modified
        ` make sure our check sum field is accurate
      $hasBeenModified:=Table_RecordHasBeenModified
($tablePtr;$checksum_FieldPtr)

        ` only mark a new DTS if modified and the skip DTS field is false
      If ($hasBeenModified) & (($skipDTS_fieldPtr->)=False)
            $modDTS_fieldPtr->:=str_GenerateDateTimeStamp
      End if

      Sync_TrackSaveRecEvent ($tablePtr;$guidFieldPtr->)

End if   `  (Not(◊disableSyncTrigger))

  ` clear our skip DTS field
$skipDTS_fieldPtr->:=False



  ` ----------------------------------------------------
  ` METHOD: Sync_TrackSaveRecEvent
  `
  ` DESCRIPTION:
  `   This method updates a "Save Event" in the Sync record tracker table
  `
  ` ASSUMPTIONS:
  `   none
  ` ----------------------------------------------------
  ` PARAMETERS:
C_POINTER($1;$tablePtr)
C_STRING(40;$2;$recGUID)
  `
  ` RETURNS:
  `   none
  ` ----------------------------------------------------
  ` HISTORY:
  `   Created by Dani Beaubien
  ` ----------------------------------------------------

Profiling_enteringMethod (Current method name)

$tablePtr:=$1
$recGUID:=$2

C_LONGINT($tableNo)
$tableNo:=Table($tablePtr)

C_LONGINT($fieldNo)

READ WRITE([_Sync_RecordTracker])

  ` find the matching sync tracking record (if there is one)
Table_LoadByRecNoOrID (->[_Sync_RecordTracker];0;-
>[_Sync_RecordTracker]Rec_GUID;$recGUID)
```

```
If ([_Sync_RecordTracker]TableNumber#$tableNo) & (Records in
selection([_Sync_RecordTracker])#0)
      QUERY([_Sync_RecordTracker];[_Sync_RecordTracker]Rec_GUID=$recGUID)
End if
If (Records in selection([_Sync_RecordTracker])>1)
      QUERY
SELECTION([_Sync_RecordTracker];[_Sync_RecordTracker]TableNumber=$tableNo)
End if

  ` if no record, then create one
If (Records in
selection([_Sync_RecordTracker])=0) | ([_Sync_RecordTracker]TableNumber#$table
No)
      CREATE RECORD([_Sync_RecordTracker])
      [_Sync_RecordTracker]Rec_GUID:=$recGUID
      [_Sync_RecordTracker]TableNumber:=$tableNo
      [_Sync_RecordTracker]Event_Code:=1  ` saved action
End if

  ` force to be a saved action
If ([_Sync_RecordTracker]Event_Code#1)  ` this might happen if deleted and
then imported!
      [_Sync_RecordTracker]Event_Code:=1  ` saved action
      [_Sync_RecordTracker]Event_DTS:=""  ` force it to be in the next sync
End if

  ` if the record DTS has changed, then update the tracker record
If
([_Sync_RecordTracker]Event_DTS="") | ([_Sync_RecordTracker]Rec_DTS#Field($tab
leNo;◊_syncFieldPos_DTS_modified{$tableNo})->)
      [_Sync_RecordTracker]Rec_DTS:=(Field($tableNo;◊_syncFieldPos_DTS_modifie
d{$tableNo})->)
      [_Sync_RecordTracker]Event_DTS:=str_GenerateDateTimeStamp

        ` take advantage of a trigger trick
        ` if this record is loaded then sync is active and is saving this
record
        ` record that it came from the other machine
      [_Sync_RecordTracker]fromDB_GUID:=""
      If (Is record loaded([_Sync_Log]))
            [_Sync_RecordTracker]fromDB_GUID:=[_Sync_Log]otherDB_GUID
      End if

      If (◊LogLevel>0)
            If ([_Sync_RecordTracker]fromDB_GUID="")
                  Log_AddEntry ("SyncSaveRecEvent:: ["+Table name($tableNo)+"]
saved with GUID:"+$recGUID+", RecDTS:"+[_Sync_RecordTracker]Rec_DTS+",
EventDTS:"+[_Sync_RecordTracker]Event_DTS)
            Else
                  Log_AddEntry ("SyncSaveRecEvent:: otherDB_GUID:
"+[_Sync_RecordTracker]fromDB_GUID+", ["+Table name($tableNo)+"] saved with
GUID:"+$recGUID+", RecDTS:"+[_Sync_RecordTracker]Rec_DTS+",
EventDTS:"+[_Sync_RecordTracker]Event_DTS)
            End if
      End if
End if

  ` make sure that this is not blank, defaults to a zero value
If ([_Sync_RecordTracker]Rec_DTS="")
      [_Sync_RecordTracker]Rec_DTS:="00000000000000"
End if
```

```
  ` make sure that the student id is accurate
If ($tablePtr=(->[Students]))
      [_Sync_RecordTracker]Student_GUID:=[Students]Student_ID
Else
      $fieldNo:=Table_Relations_FindStudKey ($tablePtr)
      If ($fieldNo>0)
            [_Sync_RecordTracker]Student_GUID:=Field($tableNo;$fieldNo)->
      End if
End if

  ` track some statistics
If (Is record loaded([_Sync_Log])) & ([_Sync_RecordTracker]Rec_GUID#"DEL.@")
      Case of
            : (Is new record([_Sync_RecordTracker]))
                  [_Sync_Log]my_addedCount:=[_Sync_Log]my_addedCount+1
                  SAVE RECORD([_Sync_Log])

            : ([_Sync_RecordTracker]Rec_DTS#Old([_Sync_RecordTracker]Rec_DTS))
                  [_Sync_Log]my_updatedCount:=[_Sync_Log]my_updatedCount+1
                  SAVE RECORD([_Sync_Log])
      End case
End if

If (Record number($tablePtr->)>0)
      [_Sync_RecordTracker]RecordNumber:=Record number($tablePtr->)
End if

SAVE RECORD([_Sync_RecordTracker])
REDUCE SELECTION([_Sync_RecordTracker];0)
READ ONLY([_Sync_RecordTracker])

Profiling_leavingMethod (Current method name)
```