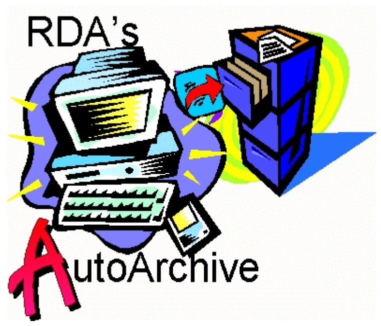

AutoArchive 1.1

by

Resource Development Associates

Installation and Use

RDA's AutoArchive is an automated facility for creating record archives and audit trails and deploying that functionality in your applications. It is designed to be easy to use and requires very little processing overhead. It also comes with a bonus custom toolbar you can add to your applications. It contains the following components:

The AutoArchive Library
The Archive File Cleanup Form

Installing AutoArchive

Installation consists of placing the library (autoarc.ldl) in the appropriate folder and setting up your application to use it. The details of connecting your application to this library and calling it's methods are described below. To use the Archive Cleanup functions you also need to place the Archive File Cleanup Form (fileslct.fdl) in the backup alias folder.

The AutoArchive Library

The AutoArchive library is the heart of the system. This library contains several custom methods which you can use in your applications. The library functions that AutoArchive contains are:

1. RDARecordBackup(stActiveName string, stSourceTable string, stBackupAlias string)

This method is the one you call from your applications to copy the current record to an archive file with a time stamp. This method uses a file named with the same name and structure as your table but prefixed with the letters "arc" and with a time stamp field added to the record. You must designate a backup alias which can either be one alias for all users of your application or individual aliases for each user. If the table is not in the alias designated it will be created the first time this method is called.

stActiveName is the identifier of the active object at the time the method is called. stSourceTable is the source table name for that object. Ways to set these two values while calling this method are described in the **Adding AutoArchive to your Applications** section of this document. stBackupAlias is an alias (without the colons) that identifies where the archive record table is located.

The method checks to make sure the stBackupAlias alias is valid. It then attempts to copy the current record, locate the table in the stBackupAlias, place the record in it and timestamp it. If the archive table does not exist it retrieves the structure of the current table, adds a timestamp field to it and creates the table. It then adds the record to the newly created table.

Error checking is included to inform the user of an error at any step in the process.

You would generally call this method such as:

lbAutoArc.RDARecordBackup(stActiveName, stSourceTable, stBackupAlias)

This method will function from any user interface object that maintains a link to a table (field, tableframe or multirecord object) but you should generally refrain from placing a field on a form without a record container (MRO or tableframe) as it causes other difficulties.

2. RDARecordBackupWithName(stActiveName, stSourceTable, stBackupAlias, stName)

This functions the same as the previous method but adds the user name supplied in stName to an additional "User" field added to the archive table in addition to the time stamp. Archive tables created with this method are named the same as your table name but are prefixed "arcwn" to distinguish them from the archive tables created without user names.

3. RDARecordBackupTc(var tcFromMethod tcursor, stBackupAlias string)

This method is the one you call from your applications when using a tcursor to update or add records rather than editing through a user interface object (UIO). This method, properly inserted in your tcursor editing code will copy the current record to an archive file with a time stamp. This method uses a file named with the same name and structure as your table but prefixed with the letters "arc" and with a time stamp field added to the record. You should designate a backup alias which can either be one alias for all users of your application or individual aliases for each user. If the table is not in the alias designated it will be created the first time this method is called.

TcFromMethod is the identifier of the tcursor which is updating the record at the time the method is called. Ways to set this value while calling this method are described in the **Adding AutoArchive to your Applications** section of this document. stBackupAlias is an alias (without the colons) that identifies where the archive record table is located.

The method checks to make sure the stBackupAlias alias is valid. It then attempts to copy the current record from the named tcursor, locate the table in the stBackupAlias, place the record in it and time stamp it. If the archive table does not exist it retrieves the structure of the current table, adds a time stamp field to it and creates the table. It then adds the record to the newly created table.

Error checking is included to inform the user of an error at any step in the process.

You would generally call this method such as:

lbAutoArc.RDARecordBackupWithNameTc(tcOwner, stBackupAlias, stName)

This method will function from any code with an active tcursor which has been attached to a table and has a record active. While this method could also be called in code where a tcursor has been attached to a UIO the first 2 methods would be preferable in that instance.

4. RDARecordBackupWithNameTc(var tcFromMethod tcursor, stBackupAlias string, stName string)

This functions the same as the previous method but adds the user name supplied in stName to an additional "User" field added to the archive table in addition to the time stamp. Archive tables created with this method are named the same as your table name but are prefixed "arcwn" to distinguish them from the archive tables created without user names.

5. RDACleanUpArchive(stBackupAlias string)

This function allows you to either delete selected archive files or delete records prior to a certain date from selected archive files. When this function is called it checks to make sure that stBackupAlias is a valid alias. It then locates all archive files (both those prefixed with "arc" and with "arcwn") in that directory including any .mb files associated with those files. It then asks if you want to delete whole archives or just records from archives.

If you select the whole archives option then you are presented a dialog which lists all the archive files located in that directory with a check box next to each file name. If you check mark the field next to a file name it will be selected and, when you close the form, it will be deleted if possible. If it can not be deleted, because it is in use for example, you will be informed which file was not able to be deleted and the process will continue. You can delete any or all archive files in a backup alias in one process this way.

If you select the records from archives option then you are presented a dialog which lists all the archive files located in that directory with a check box next to each file name and a date field on the form which is set to the current date. You can change the date if you wish. The records will be deleted for dates prior to the date input. If you check mark the field next to a file name it will be selected and, when you close the form, records date time stamped earlier than the date specified will be deleted if possible. If records can not be deleted, because the table is locked for example, you will be informed which file was not able to have records deleted and the process will continue. You can delete records from any or all archive files in a backup alias in one process this way.

To both delete archives and delete records from archives you will have to run 2 separate processes.

6. RDAtoolbar()

This method is included as a bonus and allows you to deploy the same toolbar that shows up with the Administration form. It includes cut, copy, paste, undo, locate, locate next, vcr controls, edit toggle, post record, filter field, multi-field filter, help and close form buttons. You should keep the undo.bmp with the library or in the current working directory to enable the undo button.

See the next section of this document for more information on how to implement these methods in your applications.

Adding AutoArchive to Your Applications

To add AutoArchive to your applications you must deploy the AutoArchive Library, AutoArc.ldl file with your application. Make sure the library is in an alias that your application knows about (a public in the idapi.cfg file which your application uses or project alias that you add at runtime). You then need to add code to your application to call the AutoArchive functions. Basics are described below.

In your Uses method (note the Objectpal keyword on the Uses line):

```
Uses Objectpal
RDARecordBackup(stActiveName string, stSourceTable string, stBackupAlias string)
RDARecordBackupWithName(stActiveName string, stSourceTable string, stBackupAlias string, stName string)
RDARecordBackupTc(var tcFromMethod tcursor, stBackupAlias string)
RDARecordBackupWithNameTc(var tcFromMethod tcursor, stBackupAlias string, stName string)
RDACleanUpArchive(stBackupAlias string)
RDAToolbar()
endUses
```

You can use any or all of the library methods here depending on your needs.

In your Var method:

```
Var
lbAutoArc library
rdatb toolbar           ;this line is only needed of you are using the toolbar
endVar
```

You can use any variable you wish for the library variable here but this helps keep library calls clear.

In your open or arrive method:

```
lbAutoArc.open(":YourAlias:autoarc.ldl")
```

Where YourAlias is the alias in which the ldl file is located.

Then you need calling code which triggers these methods when you want to archive a record. You can increase or decrease the granularity of your audit trail or archive depending on where and how you call these methods. For example, at the highest level of granularity you could call these methods for every time a field level change value is called. In that case you would literally have a record in your archive which showed every field change. The overhead for this level of granularity would, of course be higher than the lower end of the spectrum which would be calling the methods only when a record actually posts. Most developers and systems will only need archives at the low end of granularity. You can also pick a level of granularity in between. For example, suppose you want to capture an archive record each time a post is attempted, whether it succeeds or not in each of the tables on your form. You could do so this way.

In the form level Action method:

```
method action(var eventInfo ActionEvent)
var
        siId  smallInt
endvar
   if eventInfo.isPreFilter() then
                ;// This code executes for each object on the form
   else
                ;// This code executes only for the form

        siId = eventinfo.id()
        if siId = DataPostRecord or
           (siId = DataUnlockRecord and
            active.RecordStatus("Modified")
           )
        ;We must have posted or tried to post a record

        then
           disableDefault                    ;suspend posting ...
           stActiveName = active.Name   ;find which object triggered the action
           ; get the associated table name
           stSourceTable = active.GetPropertyasString("tableName")
           stBackupAlias = "AutoBackup" ;assign the alias for your backups
           stName = GetNetUserName()        ;get the current username

           try
           ;call the method
           lbAutoArc.RDARecordBackupWithName(stActiveName, stSourceTable,
stBackupAlias, stName)
           dodefault  ;complete the post
           onfail
           msgstop("Library Error", "Can not open the library")
           errorshow("Lib Error")
```

```
                doDefault  ;complete the post
                endtry
              endif
dodefault
          endIf
```

endMethod

To call the toolbar you need calling code such as:

```
if not rdatb.attach("RDA Toolbar") then
 lbAutoArc.RDAToolbar()
 rdatb.attach("RDA Toolbar")
endif
```

With the tcursor versions of the methods you might instead place code such as the following in a pushbutton. This example shows granularity for only successful posts. Attempted but unsuccessful posts would not be archived.

method pushButton(var eventInfo Event)

```
stBackupAlias = "AutoBackup"
stName = getnetusername()

tcInputData.open(":FacilityCredData:Provider.db")
tcOwner.open(":FacilityCredData:owner.db")
tcOwner.edit()
num = 1

;set the owner names for Providers with id's lower than 301
for num from 1 to 300


    ;if we match a provider id get the id and the name
    if  tcInputData.Locate("ProviderID", num) then
         liPid = tcInputData.ProviderID
         stVName = tcInputData.Vendor

         ;put a blank record in the owner table and add the id and name
         tcOwner.insertRecord()
         tcOwner.OwnerID = liPid
         tcOwner.Name = stVName

             ;if we can post it then it is not a key violation
             if tcOwner.postrecord() then
```

```
                    ;now run the archive process
                    try
                    lbAutoArc.RDARecordBackupWithNameTc(tcOwner, stBackupAlias, stName)
                    dodefault
                    onfail
                    msgstop("Library Error", "Can not open the library")
                    errorshow("Lib Error")
                    doDefault  ;complete the post ...
                    endtry
              endif
    endif

message(strval(num))
endfor
tcOwner.close()
tcInputData.close()
endMethod
```

The cleanup method would normally be called from a button with just the code:

```
method pushButton(var eventInfo Event)
stBackupAlias = "AutoBackup"
lbAutoArc.RDACleanUpArchive(stBackupAlias)
endMethod
```

Release Notes

There are a few cautions you should consider when using AutoArchive.

1. You can not have a BLOB field as the first field in a table you wish to have AutoArchive process. In general this should be no problem as it would be unusual to have a BLOB be the first field as it would not make an effective key.

2. AutoArchive tables are not keyed as if they were they could not back up multiple changes. While you can key an AutoArchive table for use in restoring backups, speeding searches, etc. You should make a copy of the table prior to doing so as it will no longer function as an AutoArchive table once it is keyed and will most likely cause numerous processing errors if you attempt to use it after keying.

3. Auto increment fields are converted to long integer fields in AutoArchive. This allows the preservation of the actual value. Nevertheless, Paradox's auto increment field type present potential problems when used as keys, and should not be used for that purpose because of these problems. These include the loss of data links or incorrect links created when auto increment fields are resequenced in a table rebuild and the replacement of these fields' values in table adds. Auto increment records also behave differently when posting new records since the integer value

does not exist until the record is actually posted. This can greatly complicate your ability to trap for posting and unlocking of records with AI fields as well as other actions involving those records. You might consider RDA's AutoKey product as a substitute for using auto increment fields. Otherwise you should learn how to manage and create incrementing integer fields in your applications rather than use auto increment fields.

4. File names of tables being archived with AutoArchive can not exceed 42 characters (excluding ".db") if you are using the date, time and name stamping and can not exceed 44 characters if you are using the date and time stamping only. Again this should not be a serious drawback as if you are using names in that size range you may find yourself exceeding path and filename limits in certain operations anyway.

5. When using the Archive Cleanup function and deleting files, if you wish to delete a file with an MB extension be sure you also delete the DB extension version of the file. Otherwise you will most likely cause table corruption. When deleting a file with a DB extension it is a good idea to check the list on the form and be sure there is no MB file associated with it and, if there is, delete them both. (MB files are created when your tables have BLOB fields such as memos.)

6. You should use Aliases when placing tables on your forms or attaching or opening tcursors to them. While AutoArchive 1.1 can, in most cases, correctly parse the tablename to a table which is called without an alias it is bad practice not to use aliases.

Licensing

RDA believes in clear and simple licensing. Each Full license for RDA's AutoArchive allows the developer to use AutoArchive at their development site and one production site if that is different from the development site.

A site is defined as all users having access to a server where the data for an application which uses the AutoArchive library is installed (LAN, WAN or stand alone PC). There are no user limits at a site and RDA's AutoArchive library may be used for all applications housed on that server.

Distribution licenses must be purchased for additional sites. (Distribution licenses are only $25/site.) For example, if you build an application which includes RDA's AutoArchive library and install it at 2 different clients, an additional distribution license is needed, in addition to your full license, regardless of the user count. If you install that application at a 3rd site you would need another distribution license.

On the other hand, if you build 2 applications which include RDA's AutoArchive library and install them both at the same site, only your full license is required. For each additional site you install either or both applications in you would need an additional distribution license.

Systems requiring different versions of AutoArchive (for different versions of Paradox running at the same site) must purchase one license per site for each version of AutoArchive. The full end user licensing agreement is included with the product or available by e-mail request.