AutoKey 3.0

by

Resource Development Associates

New in Version 3.0

This version of AutoKey includes 1 new library method and a modification of the control table structure. The new methods allow you to use AutoKey by creating key values padded to any length with a padding character such as leading or trailing * or zeroes. These values are returned from the library without posting them immediately which lets you use AutoKey with tables with required fields, abandon records prior to posting and avoid some cleanup you might otherwise have to perform for abandoned records. The documentation has also been improved by adding additional info on ways to use the methods that returns the key field name and the key value in an array.

Using this new method or the previously included method that returns a field name and a unique integer value (AKSetByUIOReturnFieldAndKey) would allow you to concatenate strings with the unique integer. This would allow you to use these method to create non key identifiers as well as keys. For example, you might want to have sequential invoice numbers that are not key but are unique and contain an alpha string plus a sequence number. They could contain a specific pattern (such as the first n letters of the customer name followed by a unique integer like RDA1982). This could also be useful for creating customer numbers and many other things. If you wanted to to generate such things as non key unique customer numbers or non key invoice numbers which always have the same number of characters as well as a specific pattern you can use the new method or combine it with the concatenation technique just described (e.g. creating values such as 0000085471, *******567, RDA00001982, PAR00045127, etc.) Of course such values, from either method, could also be used as keys if you wanted.

There is also a revised demonstration application available including open source code to demonstrate implementation of all the methods of AutoKey. This demonstration application is available with AutoKey in the new AutoKey Deluxe pack or separately. The separate version will not run without a copy of AutoKey but the source code may prove valuable. More detailed info on this revised demonstration application is available for download for free from the www.RDAWorldWide.com product page.

Installation and Use

AutoKey is an automated facility for creating and deploying automatically incrementing integer keys for Paradox tables. It is designed to be easy to use and trouble free. It also comes with a bonus custom toolbar you can add to your applications. It contains the following components:

The Autokey Startup Dialog
The AutoKey Administration Form
The AutoKey Library
The Internal AutoKey ID Table
The Deployment ID Table

AutoKey has two modes: Administration and Deployment. Administration helps you set up the automated key function for your applications. You can use it or not as you prefer. It is highly recommended that you use it as it will greatly simplify your setup and use of AutoKey. The deployment mode consists of the library and calls to methods in the library as well as the table which manages the automatic key incrementation. These components and how to use them are described in detail below.

To actually use AutoKey all you have to do is create a deployable key control table and deploy this table and the AutoKey library with your application. Then Autokey is invoked by calling library methods from your application. AutoKey can help you create and manage deployable key control tables. Specifics of how to add AutoKey to your application are included in the instructions below.

Installing AutoKey

To use the Administration mode you can install AutoKey in an alias of its own or in an existing alias. Installation is as simple as copying the self extracting zip file to a directory and double clicking it. If you have placed it in a directory with an existing alias, installation is complete.

If you have installed it into a directory without an existing alias, or if you want to have a specific alias for AutoKey, then open the Alias Manager in Paradox, create a new alias and map it to the directory in which you placed the AutoKey files.

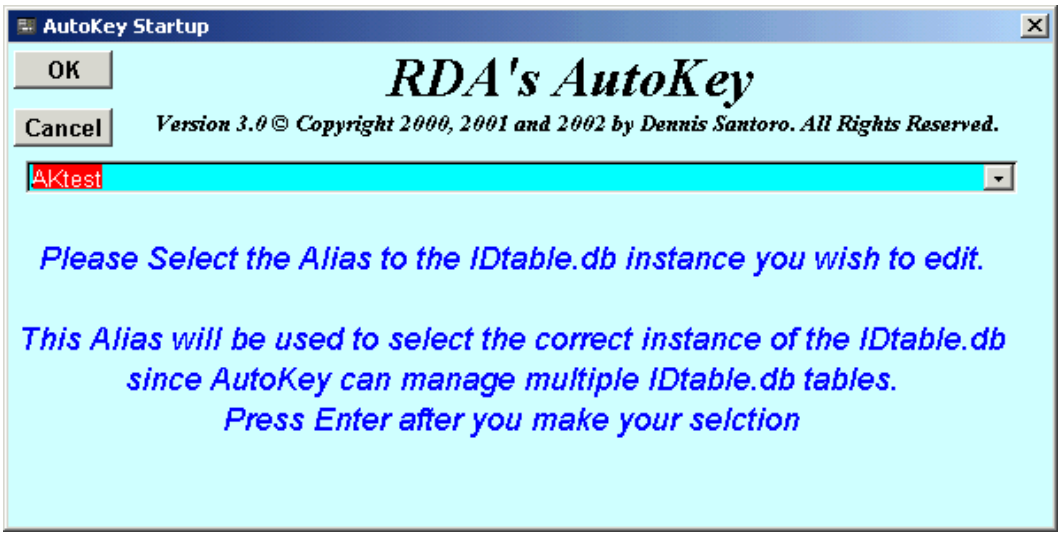
In addition, if you wish to place an icon for starting AutoKey in administration mode on your desktop or in your start menu, create an icon which starts Paradox and after the paradox exe part of the icon target line add

Drive:YourPathToAutokey\Aliasdlg.fdl -wDrive:YourPathToWorkingDirectory
-pDrive:PathToASeperatePrivateDirectory

where you substitute your drive and path names in the -w and -p statements and before the Aliasdlg.fdl call. This launch the AutoKey start up alias dialog and take you directly into the administration form. You can have AuotKey in your working directory but it is not needed nor recommended. The folder must have an alias, however. Note that to deploy AutoKey in your applications you do not need to do the above. Deployment is described below.

The Autokey Startup Dialog

You can launch AutoKey in administration mode from a startup icon or directly by opening the



AutoKey Startup dialog form Aliasdlg.fdl. The startup dialog must be in the same directory as the Administration form and a copy of the library. This does not have to be the working directory. It is best if the internal AutoKey table, IDtabID.db, is there as well but it can be in another alias. We recommend that you place these all in one separate directory and give it an Alias of it's own. Do not use AK or Akloc which are project Aliases used internal to AutoKey.

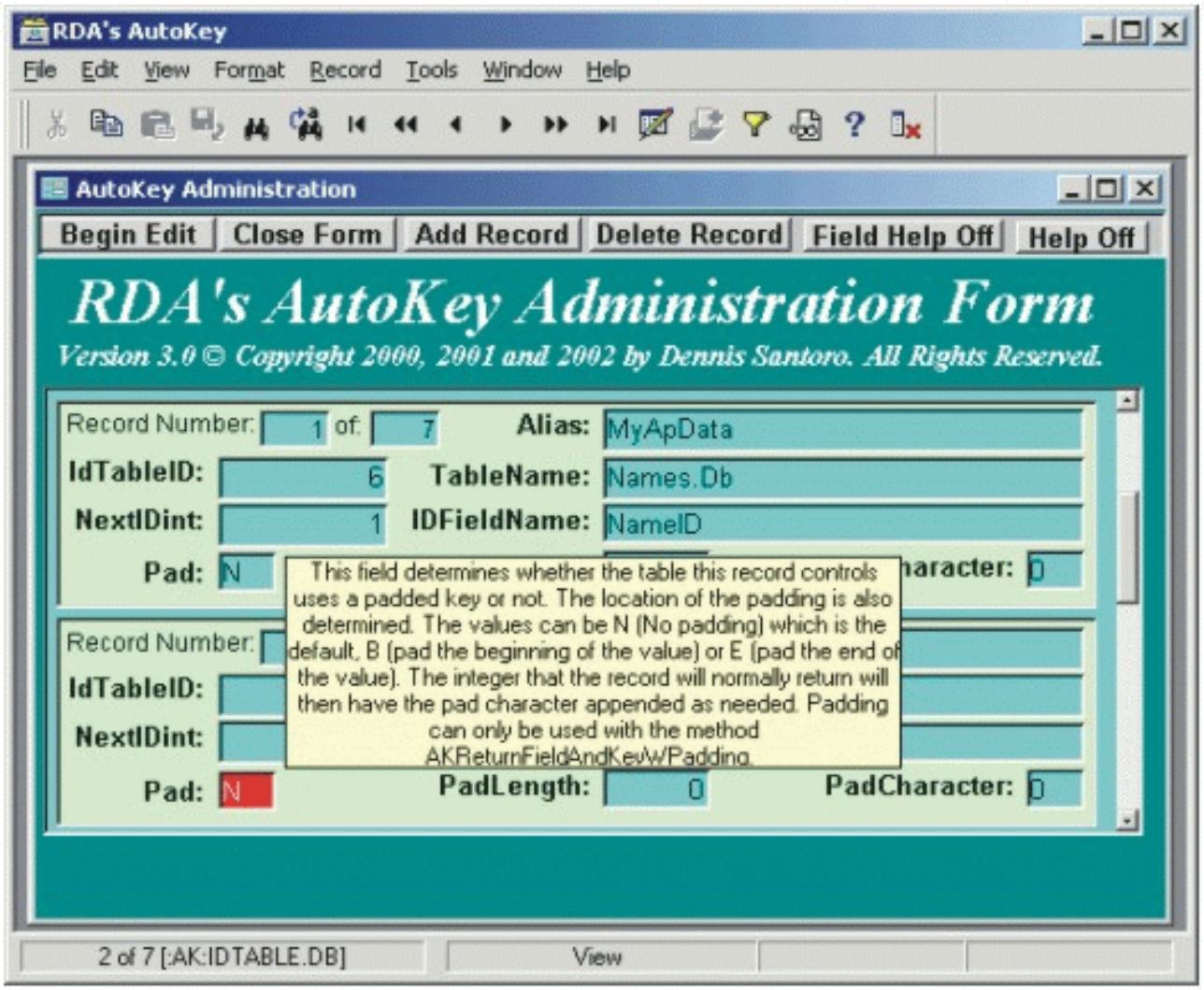
On opening, the dialog checks for the existence of the internal AutoKey table (IDtabID.db). If it is in the current working directory it asks for the Alias for the location of the Deployment table, IDTable.db, that you want to use. There can be any number of deployment tables as described below. The Deployment table must be navigated to by Alias but you can use the :Work: Alias although this is not recommended. You must not change the name of this table although you can have multiple copies of it.

If the internal AutoKey table is not in the current working directory the Dialog will present a warning describing the situation. It will then ask you for the alias in which the internal AutoKey table is located. It will then ask for the location of the Deployment table as described above. Be sure you understand the difference between the AutoKey internal table (IdtabID.db) and the deployment table, or tables, IDTable.db. (They are described in detail later in this document.) The startup then creates project aliases for the session and opens the administration form with the Deployment table that you specified.

The AutoKey Administration Form

The AutoKey Administration form allows you to specify all the tables that will have their keys managed by this deployment table. The Deployment table will be described more fully below.

The Administration form allows you to create records for the Deployment table, delete them, edit them and allows you to manage as many deployment tables as you wish. The Administration form has RDA's BubbleHelp built in (this version of BubbleHelp is not deployable but you can buy a fully deployable version from our web site). The BubbleHelp provides both short and long BubbleHelp messages for the buttons and fields on the form You can toggle between the long and short messages. You can also toggle the BubbleHelp for fields on or off.



The AutoKey Library

The AutoKey library is the heart of the system. This library contains several custom methods which you can use in your applications. Besides the library functions that AutoKey uses in it's administration it contains:

1. AKKeyValSetWithReturn(Kint longint, IDTableAlias string) longint

This method is the one you can call from your applications to set a long integer key for a table and return the key value to your calling method. This is useful since this AutoKey method sets the key in your table with a Tcursor and the return of the key value allows you to locate the record that was just placed.

Kint is the identifier of the record in the Deployment table (its key) that controls the key for the table you are adding a record to on this call. IDTableAlias is an alias (without the colons) that identifies where the Deployment table is located. The deployment table does not have to be located with your data tables. Generally it is suggested that you keep your deployment table with

the application data or in a separate folder for administrative types of tables. All users using the tables for which AutoKey increments the IDs must have access to this table as well as your applications' tables.

The method checks to make sure the IDTableAlias alias is valid. It then attempts to open the Deployment table in that alias, place the Tcursor in edit, locate the specified record by ID specified in Kint, lock it, get the next key value, increment it by 1 and put it back.

The method then attempts to open a Tcursor to the table specified in the record from the Deployment table, place the Tcursor in edit, insert a new record, set the key value, and post it.

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:

YourUIOname.Locate("YourIdfield",LibraryVariable.AKKeyValSetWithReturn(1,"YourIDTable Alias") which would add a record to the table attached to the UIO and return the key value of the new record and then locate it so the user was on the record at the end of the process. If the process fails the method returns 0. You can test for this value in your code if you wish.

You could also call it as:
liKeyVal = LibraryVariable.AKKeyValSetWithReturn(1,"YourIDTableAlias")

```
if liKeyVal = 0 then
;your error message here
else
 YourUIOname.Locate("YourIdfield",liKeyVal)
endif
```

As with all the AutoKey methods there are many other possible ways to call and use the method. See the AutoKey demo application for a variety of implementations of calls to the various AutoKey library methods. The code is open for your review. (The AutoKey demo application is available in the AutoKey Deluxe pack or can be purchased separately.)

2. AKKeyValSetNoReturn(Kint longint, IDTableAlias string)

This functions the same as the previous method but does not return a value.

You would generally call this method such as:

LibraryVariable.AKKeyValSetNoReturn(1,"YourIDTableAlias")

3. AKKeyValSetByUIOwithReturn(Kint longint, IDTableAlias string, stAKUIOName string) Longint

This method is the one you can call from your applications to set a long integer key for a table and

return the key value to your calling method. This version is useful since this AutoKey method sets the key in your table through a User Interface Object (UIO) on your form and the return of the key value allows you to locate the record that was just placed. The UIO based methods are particularly useful to take advantage of Paradox's built in behavior of placing the link value of the parent in a child record automatically in a form where the tables are linked.

Kint is the identifier of the record in the Deployment table that controls the key for the table you are adding a record to on this call. IDTableAlias is an alias (without the colons) that identifies where the Deployment table is located. StAKUIOName is the string which identifies the name of the UIO to use to place the record.

The method checks to make sure the IDTableAlias alias is valid. It then attempts to open the Deployment table in that alias, place the Tcursor in edit, locate the specified record by ID, lock it, get the next key value, increment it by 1 and put it back. If the process fails the method returns 0.

The method then attempts to attach to the UIO specified in the method call. It then checks to see if there is a new record already placed in the UIO. If not, it attempts to place one, returning an error if it can not. If there already is a new record, the method locks it so we can edit it. The method then sets the key value and attempts to post it. If the post fails an error message is triggered. You will be able to continue editing from there.

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:

```
liKeyVal = AutoKey.AKKeyValSetbyUIOwithReturn(2,"YourIDTableAlias", "YourUIOName")
active.container.container.locate("YourFieldName",likeyval)
```

4. method AKKeyValSetByUIO(Kint longint, IDTableAlias string, stAKUIOName string)

This functions the same as the previous method but does not return a value.

You would generally call this method such as:

```
AutoKey.AKKeyValSetbyUIO(2,"YourIDTableAlias", "YourUIOName")
```

5. AKSetByUIOReturnFieldAndKey(Kint longint, IDTableAlias string) dynAnyType

This method is the most flexible but a bit more complex to implement. It requires the use of a custom type so that a field name and a key value can be returned. You can use this so that you can add additional values and manipulate other fields or so that the return values are placed in the record, but the record is not posted. This latter use allows you to use AutoKey with tables using required fields. It would also allow you to concatenate the key integer with a string or manipulate it in other ways should you wish. For example, you could return the key value, add a prefix or suffix based on a constant or extracted from such things as the first 4 characters of the Customer

name. You could then cast the returned integer as a string, add the suffix or prefix and place that as the record key for the new record. More information on this is detailed later in this document.

Kint is the identifier of the record in the Deployment table that controls the key for the table you are adding a record to on this call. IDTableAlias is an alias (without the colons) that identifies where the Deployment table is located. DynAnyType is a custom type cast as a dynArray Anytype variable. Using custom types like this allows you to pass complex data types which would otherwise not be able to be passed between a library method and a form. See below in the implementation section on Type for additional details on this or check out Lance Leonard's excellent discussion of this on his www.techtricks.com web site.

The method checks to make sure the IDTableAlias alias is valid. It then attempts to open the Deployment table in that alias, place the Tcursor in edit, locate the specified record by ID, lock it, get the next key value, increment it by 1 and put it back. If the process fails the method returns an empty dynArray. If it succeeds it will set the dynArray index stField to the field name of the key field in the table you are incrementing the key for and the index liKey as the value of the new integer key for the record you wish to add.

You would generally call this method such as:

```
dynFieldAndKey = AutoKey.AKSetByUIOReturnFieldAndKey(2,"YourIDTableAlias")
;// an empty array is returned if there was an error
if dynFieldAndKey.size() = 0 then
msgInfo("Error", "Returned no values. This means there was an error")
return
else

    ;//since we are trapping for the insert action we should be on
    ;//a blank record. But we will need to lock it

    if active.lockrecord() then
    ;//set the returned field name with the returned field value
    ;//This method can be used to add any other values to the record before posting
    ;//or you can simply leave the record unposted after placing the key value.
    ;//so that the user can complete the record.

    ;//the returned dynArray has 2 values. The stField is the string value of the  field
    ;// name of the key field as designated in the record in the Idtable.db record we designated

    ;//the liKey is the long integer value of the returned key.

        ;//the active.container.(dynFieldAndKey["stField"]) indicates the field name
        ;//as indicated in the dynArray.
        ;//the dynFieldAndKey["liKey"] is the returned key value
         active.container.(dynFieldAndKey["stField"]) = dynFieldAndKey["liKey"]
```

```
                if active.postrecord() then
                        if not active.unlockrecord() then
                        errorshow("Can't Unlock record")
                        else
                        active.locate(dynFieldAndKey["stField"],dynFieldAndKey["liKey"])
                        endif
                else
        ;// since this table has a required field we will move to it and
        ;//show the user a message

        msginfo("Required field", "The field named \"Required\" is a required field. Please enter a
value.")
        required.moveto()
          errorshow("Could not post the value "+dynFieldAndKey["liKey"]+ " to field
"+dynFieldAndKey["stField"])
                endif
          else
          errorshow("Can not lock the record here in "+active.name)

          endif
endif
```

If we wanted to use a concatenated value which, for example, consisted of the first 2 characters of the customer name plus the returned key integer we could do the following.

```
    ;//since we are trapping for the insert action we should be on
    ;//a blank record. But we will need to lock it

    if active.lockrecord() then
    ;//set the returned field name with the returned field value after adjusting it
    ;//This method can be used to add any other values to the record before posting
    ;//or you can simply leave the record unposted after placing the key value.
    ;//so that the user can complete the record.

    ;//the returned dynArray has 2 values. The stField is the string value of the  field
    ;// name of the key field as designated in the record in the Idtable.db record we designated

    ;//the liKey is the long integer value of the returned key.

        ;//the active.container.(dynFieldAndKey["stField"]) indicates the field name
        ;//as indicated in the dynArray.
        ;//the dynFieldAndKey["liKey"] is the returned key value

        ;//get the first 2 characters of the customer Name field and concatenate with
        ;//with the key, casting the value as a string. Obviously the key field would
```

```
        ;//need to be an alpha field long enough to accept the result rather than a long int
if Customer.CustName.isspace() then
;//no customer name available
MsgStop(“Error”, “There is no customer name entered yet. We can not complete the process.”)
;//end the process
Return
endif

        ;//this would give us a concatenated string such as RE87265 assuming the
        ;//customer name was Resource Development Associates and the returned
        ;//long integer key was 87265
        active.container.(dynFieldAndKey["stField"]) = Customer.CustName.substr(1,2)+
String(dynFieldAndKey["liKey"])
                if active.postrecord() then
                        if not active.unlockrecord() then
                        errorshow("Can't Unlock record")
                        else
                        active.locate(dynFieldAndKey["stField"],dynFieldAndKey["liKey"])
                        endif
                else
      ;// since this table has a required field we will move to it and
      ;//show the user a message

      msginfo("Required field", "The field named \"Required\" is a required field. Please enter a
value.")
      required.moveto()
        errorshow("Could not post the value "+Customer.CustName.substr(1,2)+
string(dynFieldAndKey["liKey"])+ " to field "+dynFieldAndKey["stField"])
                endif
        else
        errorshow("Can not lock the record here in "+active.name)

        endif
endif
```

Using this method of concatenation would allow you to use this method to create non key identifiers as well. For example, you might want to have sequential invoice numbers that are not key but are unique and contain an alpha string plus a sequence number. They could contain a specific pattern (such as the first n letters of the customer name followed by a unique integer like RDA1982). This could also be useful for creating customer numbers and many other things.

If you wanted to to generate such things as non key unique customer numbers or non key invoice numbers which always have the same number of characters as well as a specific pattern you can use the method in the next segment or combine it with the concatenation technique just described (e.g. creating values such as RDA00001982, PAR00045127, etc.) Of course such values, from

either method, could also be used as keys if you wanted.

6. AKReturnFieldAndKeyWPadding(Kint longint, IDTableAlias string) dynAnyType

This method is also very flexible but a bit more complex to implement similarly to the method AKSetByUIOReturnFieldAndKey. It also requires the use of a custom type so that a field name and a key value can be returned. The returned key value is, however, already a string, rather than a long integer. It is also padded to a specific length.

To use this method you must first modify at least 2 of the three fields on the IDtable.db record for the control record that will be used. The Idtable.db has 3 fields for each control record that normally are left in their default state. These fields are:

Pad - This field determines whether the table this record controls uses a padded key or not. The location of the padding is also determined. The values can be N (No padding) which is the default, B (pad the beginning of the value) or E (pad the end of the value). The integer that the record will normally return will then have the pad character appended as needed. Padding can only be used with the method AKReturnFieldAndKeyWPadding.

PadLength - This field determines the total size of the alpha value returned for a padded key. The padding needed is determined by the size of the returned key integer subtracted from the value in this PadLength field. The value can not exceed the size of the key field controlled by this record. It is best to set it to the full size of the alpha field of the controlled key. If you are concatenating the padded key with additional characters such as the beginning characters of a customer name you must shorten this PadLength value by the number of characters you intend to add sothat the Padlength plus the number of additional concatenated characters does not exceed the length of the target field. Padding can only be used with the method AKReturnFieldAndKeyWPadding.

PadCharacter - This field determines the alpha character to add to a padded key. The value is added to the integer value returned for a key field controlled by this record. This value defaults to 0 but extreme care should be used if padding the end of the value rather than the front. If you use 0 as the end pad character in a 5 character field, for example, 1 will become 10000, and so will 10, 100, 1000 and 10000. Obviously this would cause key violations. This would not be the case with a nonnumeric character. X, for example, would produce 1XXXX, 10XXX, 100XX, etc. Padding can only be used with the method AKReturnFieldAndKeyWPadding.

You can use this so that you have consistent size key values and so you can manipulate other fields or so that the return values are placed in the record, but the record is not posted. This latter use allows you to use AutoKey with tables using required fields. It would also allow you to concatenate the key integer with another string or manipulate it in other ways should you wish as you can with AKSetByUIOReturnFieldAndKey. For example, you could return the key value, add a prefix or suffix based on a constant or extracted from such things as the first 4 characters of the Customer name. You could add the suffix or prefix to the returned value and place that as the record key for the new record. If you do this, however, you must make the PadLength as much shorter as what you intend to concatenate so that the result will fit in the target alpha key field.

More information on this is detailed in this document in the above section on AKSetByUIOReturnFieldAndKey. You would not need to cast the return strKey, instead of liKey as a string, however, as it already is returned as a string.

Kint is the identifier of the record in the Deployment table that controls the key for the table you are adding a record to on this call. IDTableAlias is an alias (without the colons) that identifies where the Deployment table is located. DynAnyType is a custom type cast as a dynArray Anytype variable. Using custom types like this allows you to pass complex data types which would otherwise not be able to be passed between a library method and a form. See below in the implementation section on Type for additional details on this or check out Lance Leonard's excellent discussion of this on his www.techtricks.com web site.

The method checks to make sure the IDTableAlias alias is valid. It then attempts to open the Deployment table in that alias, place the Tcursor in edit, locate the specified record by ID, lock it, get the next key value, increment it by 1 and put it back. If the process fails the method returns an empty dynArray.

If it succeeds it checks to see if you are using the version 3.0 type deployment table which contains the Pad field. If not, an error is shown and the process terminates. If the Pad field is found it sets the pad type to no, beginning or end. If the type is No an error is shown and the process terminates. Otherwise it checks padLength.

If Padlength is 0 an error is shown and the process terminates. Otherwise the PadCharacter is set. The length of the returned key value is then evaluated against the Padlength. If the returned key value length is longer than the Padlength an error is shown and the process terminates since the key value would not fit when the post attempt was made.

If all the above succeeds it will set the dynArray index stField to the field name of the key field in the table you are incrementing the key for and the index stKey as the string value of the new integer key for the record you wish to add.

You would generally call this method such as:

```
dynFieldAndKey = AutoKey.AKReturnFieldAndKeyWPadding(2,"YourIDTableAlias")
;// an empty array is returned if there was an error
if dynFieldAndKey.size() = 0 then
msgInfo("Error", "Returned no values. This means there was an error")
return
else

    ;//since we are trapping for the insert action we should be on
    ;//a blank record. But we will need to lock it

    if active.lockrecord() then
    ;//set the returned field name with the returned field value
```

```
;//This method can be used to add any other values to the record before posting
;//or you can simply leave the record unposted after placing the key value.
;//so that the user can complete the record.

;//the returned dynArray has 2 values. The stField is the string value of the  field
;// name of the key field as designated in the record in the Idtable.db record we designated

;//the stKey is the string value of the returned key.

    ;//the active.container.(dynFieldAndKey["stField"]) indicates the field name
    ;//as indicated in the dynArray.
    ;//the dynFieldAndKey["stKey"] is the returned key value
     active.container.(dynFieldAndKey["stField"]) = dynFieldAndKey["stKey"]
            if active.postrecord() then
                    if not active.unlockrecord() then
                    errorshow("Can't Unlock record")
                    else
                    active.locate(dynFieldAndKey["stField"],dynFieldAndKey["stKey"])
                    endif
            else
    ;// since this table has a required field we will move to it and
    ;//show the user a message

    msginfo("Required field", "The field named \"Required\" is a required field. Please enter a
value.")
    required.moveto()
      errorshow("Could not post the value "+dynFieldAndKey["stKey"]+ " to field
"+dynFieldAndKey["stField"])
            endif
      else
      errorshow("Can not lock the record here in "+active.name)

      endif
endif
```

7. 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 must keep the undo.bmp with the library to enable the undo button.

The Internal AutoKey ID Table (IDTabID.db)

This table is used to set the key field of the deployment tables as you create them. It starts at 1

and increments as you add records to deployment tables. It continues to count regardless of which deployment table (IDtable.db) you are using. This would allow you to add your deployment tables together in the future should you ever want to. Both the Internal AutoKey ID table and the Deployment table use long integers as keys themselves. That allows 2,147,483,647 values (since they start at 1 and do not use negative values) IDtabID.db is the Internal ID table that controls the Deployment tables IDtable.db. (See the next section for details on the IDtable.db) There can be as many IDtable.db's as desired. IDtable.db's each have records which control 1 other table in your application. By using one IDtabID.db to control all IDtable.db's the combined total records of all IDtable.db records can be 2,147,483,647 records. Each record in an IDtable.db controls another table that can have 2,147,483,647 records. This allows the IDtable.db's to be added together should you wish. Do not rename the table, however, as AutoKey uses your alias and the name IDtable.db. Generally we recommend that you use one copy of IDTable.db for each application (set of tables) you wish to control. This facilitates distribution of your applications later. See the license agreement for details on licensing for distribution purposes. You can not change the passwords on the IDtabID.db.

The Deployment ID Table (IDTable.db)

This table manages the keys in your applications. This table, named IDTable.db (how creative is that!?!?) is what the AutoKey methods look for. This table has a master password, provided to you when you receive your copy of AutoKey and an I&D password used by the methods and also provided to you. You can change the master password of this table if you want. Do not change the I&D password as the AutoKey functions use it. Password protection is provided so that users can not modify the deployment table casually and inappropriately.

The Idtable.db contains the fields in the table below. You can modify the size of the fields or add fields if you choose. Do not modify the field names or types of the existing field. If you add fields you will have to access those fields in table mode or through a form of your own. We suggest that you copy this table blank to your application before starting to work with it.

While some of the alpha fields here are set to 35 characters each, you can modify this. Consider whether you really need aliases, table names or field names longer than 35 characters if you do though. The longer these are the more you increase your risk of coming up against internal limits of Paradox and the BDE. Also, use of spaces, dashes, underscores and other special characters is discouraged in your naming conventions as these can create problems within OPAL, especially for queries. You should not modify the length of the Pad and PadCharacter fields. The methods will have problems if these fields are not 1 character each.

The old version 1.0 and 2.0 AutoKey IDTable Types can still be used with the new AutoKey 3.0 Library but you can not use the new AutoKey.AKReturnFieldAndKeyWPadding method with the old version IDTable.db

Field	Type	Size	Description
IDTableID	I		The Key for this table. This integer is automatically incremented by AutoKey when you use the Administration form. Alternatively you can create your own form and use AutoKey itself to increment this field. You can not edit this field in the Administration form. This is the integer you pass to the AutoKey methods to locate the record which controls one of your application's table's key integer.
Alias	A	35	The Alias where the table that this record controls can be found (no colons). The methods use this Alias to find the table to add a record to and to place the key in. This can be blank but that is not recommended. Without aliases you will be required to have the Idtable and the library in the current working directory. Aliases are strongly recommended.
TableName	A	35	The name of the table this record controls including the .db. This and the alias are used to attach a Tcursor to the table that AutoKey will add the record to.
IDFieldName	A	35	The field name of the integer key field in the specified table. This will be the field that is set to the new integer value.
NextIDint	I		The next Id to use for this table. It defaults to 1 which is where you normally wish to start. It can be edited so that you can set it back to one after testing and before deployment. It is not necessary to reset the integer and it should be avoided when working with the table controlling an application which is in active use. This field allows the tables it controls to contain 2,147,483,647 records. If you need more values than that you could set it to a negative number to start but you should, instead, consider a database back end such as Interbase.
Pad	A	1	This field determines whether the table this record controls uses a padded key or not. The location of the padding is also determined. The values can be N (No padding) which is the default, B (pad the beginning of the value) or E (pad the end of the value). The integer that the record will normally return will then have the pad character appended as needed. Padding can only be used with the method AKReturnFieldAndKeyWPadding.

PadLength	S		This field determines the total size of the alpha value returned for a padded key. The padding needed is determined by the size of the returned key integer subtracted from the value in this PadLength field. The value can not exceed the size of the key field controlled by this record. It is best to set it to the full size of the alpha field of the controlled key. Padding can only be used with the method AKReturnFieldAndKeyWPadding.
PadCharacter	A	1	This field determines the alpha character to add to a padded key. The value is added to the integer value returned for a key field controlled by this record. This value defaults to 0 but extreme care should be used if padding the end of the value rather than the front. If you use 0 as the end pad character in a 5 character field, for example, 1 will become 10000, and so will 10, 100, 1000 and 10000. Obviously this would cause key violations. This would not be the case with a nonnumeric character. X, for example, would produce 1XXXX, 10XXX, 100XX, etc

Adding AutoKey to Your Applications

To add AutoKey to your applications you must build the IDtable.db to include the aliases, table names and key fields in those tables which AutoKey will control. You then must deploy the AutoKey Library, AutoKey.ldl and the IDtable.db, IDtable.px and IDtable.val files. Make sure you place the correct IDtable. Also, make sure the library and table are in an alias, or aliases that your application knows about (a public or project alias in the idapi.cfg file which your application uses or aliases declared by your application at startup). You then need to add code to your application to call the AutoKey functions. Basics are described below.

In your Uses method (you only need the prototypes which you will actually use on any given form):

Uses Objectpal
AKKeyValSetWithReturn(Kint longint,IDTableAlias string)longint
AKKeyValSetNoReturn(Kint longint,IDTableAlias string)
AKKeyValSetbyUIO(Kint longint,IDTableAlias string, stAKUIOName string)
AKKeyValSetbyUIOwithReturn(Kint longint,IDTableAlias string, stAKUIOName string) longint
AKSetByUIOReturnFieldAndKey(Kint longint, IDTableAlias string) dynAnyType
AKReturnFieldAndKeyWPadding(Kint longint, IDTableAlias string) dynAnyType
RDAToolbar()
endUses

You can use any one of the library methods here, depending on your needs. The RDAtoolbar() method allows you to call and use the custom toolbar (the same toolbar that pops up when you

open the Administration Form). You can call the toolbar with the code shown below in the open or arrive method section.

In your Var method:

Var
AutoKey library
rdatb toolbar

;//if you plan to use the custom type you need to declare a dynarray to return the value to
dynFieldAndKey dynarray[] anytype

;//or you can use the custom type and simply declare a variable for that instead
dyatFieldandKey dynAnyType
endVar

You can use any variable you wish for the library variable here but this helps keep library calls clear. The same applies to other variables.

Type
;//because the method
;//AKSetByUIOReturnFieldAndKey(Kint longint, IDTableAlias string) dynAnyType
;//returns a dynarray you must declare a special type since
;//paradox can not directly declare and pass dynarrays between methods

dynAnyType = dynarray[] anytype
endType

In your open or arrive method:

if not AutoKey.open(":YourAlias:autokey.ldl") then
errorshow(“Library not open”)
endif

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

Then you need calling code as described above in the library methods section of this documentation or any other calling code. AutoKey will manage placing the keys and keeping track of them whenever you call one of the AutoKey methods

To call the toolbar you need calling code such as:

if not rdatb.attach("RDA Toolbar") then
AutoKey.RDAToolbar()
rdatb.attach("RDA Toolbar")

endif

Calling the methods

The calls to your methods are only limited by your programming skills and the specific limitations of the methods described above. The descriptions of method calls here are only a set of basic examples. More ideas and demonstrations of various method calls for various relationships of tables and various designs of forms, containership and approaches to handling data entry are available in the AutoKey Demo application. It is available in the AutoKey Deluxe pack or separately at our online store. The demo application runs in all versions of AutoKey so you only need one copy of it no matter how many versions of AutoKey 2.0 you have. Many of the methods in the demo application can be copied and pasted to your own applications and then modified for the specific references as needed.

Licensing

RDA believes in clear and simple licensing. Each Full license for RDA's AutoKey allows the developer to use AutoKey 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 AutoKey library is installed (LAN, WAN or stand alone PC). There are no user limits at a site and RDA's AutoKey library may be used for all applications housed on that server.

Distribution licenses must be purchased for additional sites. (Distribution licenses are only $10/site.) For example, if you build an application which includes RDA's AutoKey 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 AutoKey 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 AutoKey (for different versions of Paradox running at the same site) must purchase one license per site for each version of AutoKey. The full end user licensing agreement is included with the product or available by e-mail request.

Contact us to discuss volume terms or other licensing issues.