What's new in Version 2.0!

Everything!<g> A lot has changed and whilst the underlying principals are the same we have changed just about everything in detail.

Starting with the

Security Table.

The format and fieldtypes have changed.

1.SID - Alpha 50.
2 OID - Numeric (actually a Longint)
3 OList - BLOB
4 Name - Aplha 30
5.Type - Numeric (actually an integer)
5 CHK - Numeric (actually longint)
6 LastDate - Date (actually TDateTime)

I have kept the standard table as Paradox4 Table to ensure that its compatible with all your applications (CrystalReport4 doesnt work against PDox5 tables) but DACM will happily work with a PDox5 table and the OID,Type,CHK and LastDate fields can be redefinded to their more appropriate field types.

Why did we change - Olist to BLOB allows us to expand the size of Groups being allocated to an Object, OID to longint again increases maximum potential size and as you will see below it enables us to simplify and increase the power of DACM by having this field as a longint. Type to an integer, well its occasionly easier to handle an integer than a char, specifically its more useful in a Case statement which is often where the TYpe field may get used. CHK is a sort of checksum to prevent tampering with the data in the Table.
We have tried to move away from encrypting the data in the table and move towards simply verifying that the data hasn't been tampered with in order to make it easier for you to access that data for custom reports etc. without needing to run through decryption routines within the DACM.

Now only the SID is encrypted, to hide the Password from prying eyes.

Whenever DACM accesses the Table it will check the CHK field and can determine from it wether the record has been altered in some unauthorised way. The CHK calculation is based on the SID,OID and Olist fields, these fields cant be changed in anyway without this calc being wrong if this happens DACM warns you and in the case of checking permissions returns nil permissions! Just to emphasize the point you cant "cut and paste" from one record to another (which you can do with MS Access) to gain access to an object or to change you permissions and neither does increasing permissions for one group but decreasing permissions on another group work. CHK is not just a simple XOR of values!

For Paradox Tables we also now suport Password protecting the Table, so you can set your own password and then set the Password property of the DACM to this value and it will automatically open the table without user intervention and of course then others will not be able to open the table independently of your application. This does add an important level of security for the table against all but the most technicaly minded intruder so we encourage you to use it.

At this point I would now regard the Table as tamper proof against anyone except a security expert or serious computer hacker. If you feel threatened by either of these DACM is the wrong product to protect your data.

One final point on this. We can prevent soemone from gaining unauthorised access to the system, we of course cannot stop someone causing malicious damage by deleting the security Table.
For your own sake should regularly back up the security table!

TSecurity

We have introduced a new global security object, the type is ***TSecurity*** and a global variable ***Security*** has been declared.
Whenever Delphi loads the global security object is created and can be accessed by the variable Security simply by adding the DACMGR unit to the uses clause of your units.
This means that security is ACTIVE immediatlely, for the more technical of you <g>this means even before the application is running. Note that we can log someone on BEFORE the first form has been fully created and before you get to that line in the project source that says Application.Run.

Most DACM 1.0 **METHODS** have been transfered to the Security Object, whilst most **PROPERTIES** have been left in the DACM component.

All procedures directly accessing the underlying table are in DACM whilst the security object contains procedures that only indirectly access the table.

Why the change-Firstly its more secure, there is never a time when Security isnt enabled from when the Exe loads to when it terminates, secondly its easier to use.

Security.CheckPermissions

than

Unit1.Form1.VSSecurity1.CheckPermissons

This is also important for future developement as the security object will be able to manage more than one Security Table, we see possible use being in mobile computing where a User will logon to a security table on there laptop, dial up to head office, log on to the head office security table and then work with the application against head office files.

It also (I think!) makes it easier to derive new version of the DACM ***component*** tailored for specifc Databases as the DACM component contains all the functions that directly access the underlying table.
For example all the APPEND* procedures are in the security object, they do not access the table directly but call a procedure WRITERECORD in the DACM.

Note one side effect in creating this global security object is that it CANT BE DERIVED FROM, if you want to change it you must recompile a replacement DCU this gaurantees that it works the way it says it will work or not at all. We wont go into but for some work we do this is considered useful.
It also means we are setting some standards and saying this part of DACM security shouldnt be touched unless you have good reason!!
This dosent stop you from deriving from the DACM Component.

We have changed the definition of the ***SID*** to a 50 Character string, 30 for the name, and 20 for the PIN. This just gives you more flexibility, you can now also set a minimum PIN length that forces users to supply/create a certain length of PIN, by default the length is 0 which by the way makes it legitimate (but unsafe) to use only Names and a blank PIN (this has some advantages

when DACM is used in conjuntion with another security system).

The ***OID*** has changed to a longint and the implications of this are far reaching.
A longint is 4 bytes, a pointer is also 4 bytes, so anywhere a pointer can go so can the OID!
OK that means TLIST's! (and TSTRINGS) . You can therefore use AddObject method of any TSTRINGS object with the OID as the pointer, NOTE (really hit you over the head with this) the pointer doesnt point to an OID the pointer IS the OID .....makes life a LOT easier.

We actualy only use three of the four bytes to identify an object leaving the high byte of the high word free.....what for???? well Permissions!

We have changed ***Permissions*** into a set 0..7 of byte, that is you can have 8 different permissions ranging from 0 to 7 and they can all be treated as a set and stored in a single byte. Making it a set type gives you all the power of Pascal sets (see delphi help on SETS).
As we have defined it as a byte and not our own predefined permissions you can your self define constants like pReadOnly=0,pModify=1,etc and then access permissions using these Permissions ie *if pReadOnly in Permissions then* ..

Therefore when we pack the OLIST Blob with data we can get the OID and the permissions into a single 4 byte block.
The whole process of manipulating a security record then becomes much easier, we have defined a utility record TOIDRec which allows us to access individual bytes and more importantly allows the syntax Permissions:=TOIDRec(OID).Permissions.
We have include routines in DACM to Read and Write to the BLOB field, in both cases the return type is a TLIST with the pointer=OID. Wherever you see the OList in DACM it will be a TLIST and you can use all the standard TLIST methods like IndexOf(OID), Add(OID), Items[10]. This makes manipulating the permissions list really straight forward.

We have also provided a procedure ***IsMemberOf*** which checks wether the current user is a member of a certain group and ***IsInList*** which checks wether the supplied User is in the supplied list. This last routine is written in ASM to speed the process up. They are heavily used by the security system.

We have added a new function ***VerifyUI*** which displays an Input Box for the user to re-enter there PIN and have this verified against the loggedon users PIN. You can use this if there are certainVIP functions you wish to have protected like approving a transaction where on EACH transaction you ask the user to enter there password to approve the transaction. ***Verify*** does the same thing but doesnt disply the InputBox instead you must get the string yourself and pass it in as a parameter to this function.

We have added an option that allows you to ***supress*** the logon dialog, instead a logon event is generated with parameters UserName and UserPIN, on returning these are expected to contain a valid UserName and UserPIN which will be used to perform a silent logon to the security system. This can be used to where you wish to logon to another system where (ulitmatley) the UserName and PIN are set.

for example
MSMail, so you get the user to input ther Name and PIN and then call MSMails Logon function and if sucessful you pass the Name and PIN back to DACM which will log you on without needing another dialog. Obviously this requires that the Name and PIN are cordinated between the systems so we remove the capabilty of a User to change his PIN instead the Administrator can set both Name and PIN for a user (as long as they remeber!!!!).
This process is also used to now provide support logon to SQL servers, ie there is only one logon displayed which uses the supplied Name and PIN to logon to both DACM and the server.

If you want the standard LogOn but want PIN's administred centrally there is a published property

to control this.

The ***AutoLogOn*** property has been removed instead DACM autologs on the first time you call check permissions. The same effect can be achieved by using the new SecurityObject component described below, which calls check permissions as it is created.

We have added a new event ***OnDACMEvent*** which is fired whenever somethings happens, this could be a LogOn or PermissionCheck or Changing the PIN or even Writing Records to the table. This enables you, amongst other things, to log all activities to an event log table. To help this process whenever a record is changed in the table we record the Date and Time (only Date for PDox4 Tables).

TSecurityObject

We have also added a new component ***TSecurityObject***, it has one published property ObjectSID, which is the SID of an object you want to protect, by default when dropped on a from it pick up the forms name. When loaded this component checks the Permissions for the current logged on user and then fires an OnPermission event. You can react to this event and for example set the navigator buttons on a form based on these permissons.
At run time you can also access the Permissions property to check the permission levels or recacluate the permissions by calling the components refresh method.

This component is also intended as a base class from which you can derive other more useful components to protect more specific objects.

Generally we have replaced boolean sucess/fail results with Exceptions - ESecurity. Where the User can retry we catch the exception and display a message box, if the exception will terminate the application we pass it over to delphi to display there GPF style message box, if the error is serious but not fatal we let delphi display its GPF box (ie if we detect a record has been tampered with) to make it unusual enough that someone (hopefully) will tell you about it.

We have provided routines to fill ListBoxes with list of Users, Groups or Objects or all three, (see GetAll, GetListType in the DACM component).

Minor improvements have been made to the UserInterfaces, the Maintenance Listbox no longer have the annoying flicker as the fill, the setting of Permissions no longer needs you the press the SET button, you dont need to enter a Password when creating a Group , etc.

All messages in the system have been moved out to a resource file, if you have a resource editor you can change these to give you international language support. WE also use this approach for the text of the Permissions dsiplayed in the Maintenance dialog.
The values of these resource string are in the file DACCONST. If they conflict with your applications existing resource strings you can change the values by change a Base constant, although we are reserving all values between 18-19000 for this and any future VSL applications.

Finally, we have a small unit to convert DACM1 tables to DACM2. Back everything up before using it. Use a new blank security Table then the prg will auto convert an old table into the new format.

Thank you for your support in purchasing DACM 1.

V2 is free to all owners of V1.(as will the 32bit v3)

Regards Rob Edgar.