ALBI Investment Group Services (Pty) Ltd t/a AIGS
Byls Bridge Office Park, Building 14, Block B
Corner Olievenhoutbosch Road / Jean Avenue, Centurion
South Africa
PostNet Suite 447, Private Bag X033, Rivonia, 2128
Tel: +27 (0)11 267 8600
Fax +27 (0)11 465 0385

Dynamic Data Masking

Date: May 2025

First, some introductory information about Dynamic Data Masking (DDM). Simply put, it is securing data in the database at a logical level, defined by the administrator.
Let's imagine a situation where a developer creates an application that displays sensitive data such as a phone number, address or even a PESEL number or monthly earnings. On the other hand, there are people in the company who should see this data. We can solve this at the application development level or do it globally at the database level. This solution forces each user to log in to the database and, depending on the permissions, gives access to selected fields in the tables.

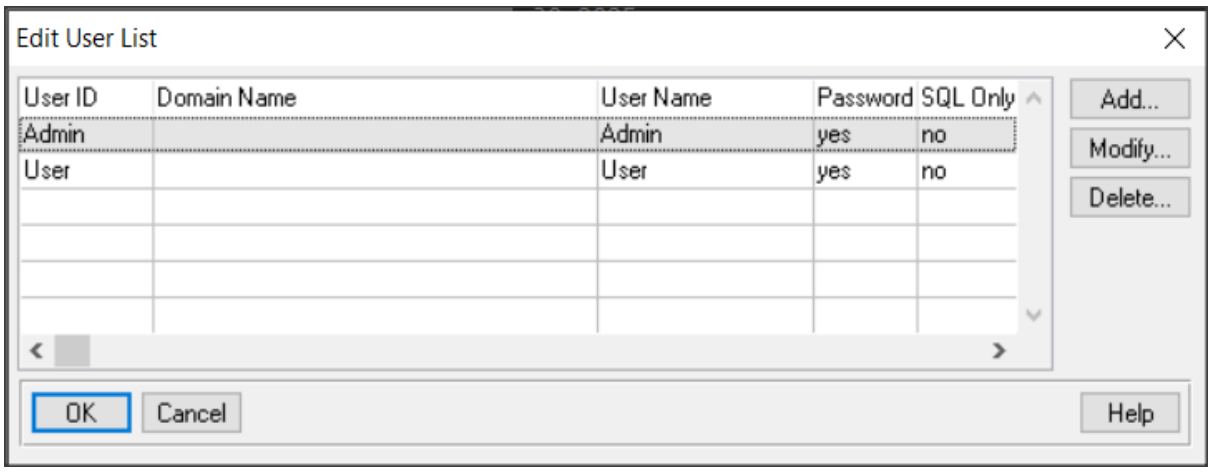
The DDM administrator (or security administrator) can configure a mask for table fields that hides sensitive data in the query result set. It also controls the access permissions of users to view unmasked values of specific fields.
To use DDM, you need to have a license for **The OpenEdge Advanced Security (Progress OEAS)** add-on and an Enterprise database server. The minimum version is **12.8.4**.

Now we create a test database, a copy of the sports2020 database and enable the DDM function.
proutil sports2020 -C enableddm
We check if this function is added (not active yet).
proutil sports2020 -C describe

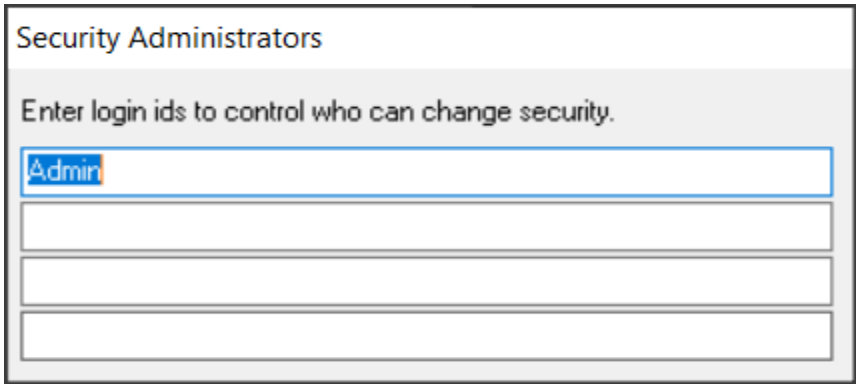
```
Database Features

   ID   Feature                          Active  Details
 ----   -------------------------------  ------  -------
    5   Large Files                      Yes
    9   64 Bit DBKEYS                    Yes
   10   Large Keys                       Yes
   11   64 Bit Sequences                 Yes
   33   Dynamic Data Masking             No
```

The first step is to add users to the database. In the online documentation, this is done by the program in the object ABL. I will do it as standard in the Data Administration -> **Admin -> Security -> Edit User List** tool.
I add users: Admin, User.

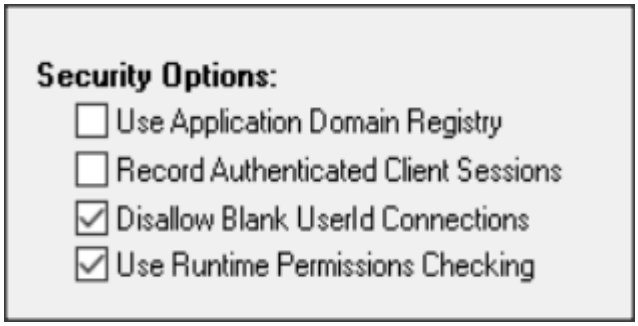
Of course, you can use user accounts defined not only in the database but also in external authentication systems for managing access to masked data.



Log in as **Admin**, then navigate to **Admin → Security → Security Administrators** to assign the user as a Security Administrator.



It's recommended to block access for unauthenticated users during runtime and to enforce permission checks on user accounts at runtime as well. **Admin -> Database Options**



To control access to masked data, you need to define user roles and map these to database users. This can be done in ABL for users defined directly in the database.
The following sample code creates the role **myRole** which we will use to define the permissions granted to users to unmask data:

```
USING OpenEdge.DataAdmin.*.

VAR DataAdminService service.
VAR IRole oRole.
VAR LOGICAL lResult.

service = NEW DataAdminService(LDBNAME("DICTDB")).

oRole = service:NewRole("myRole").
oRole:Description = "Role for DDM Admin".
oRole:IsDDM = true.
lResult = service:CreateRole(oRole).

DELETE OBJECT service.
```

The next step is to create **an authorization tag** and associate it with the role created in the previous step. The authorization tag establishes a connection between the user-defined DDM roles and the table fields to which the mask is to be applied. The tag name must start with **#DDM_SEE_**.

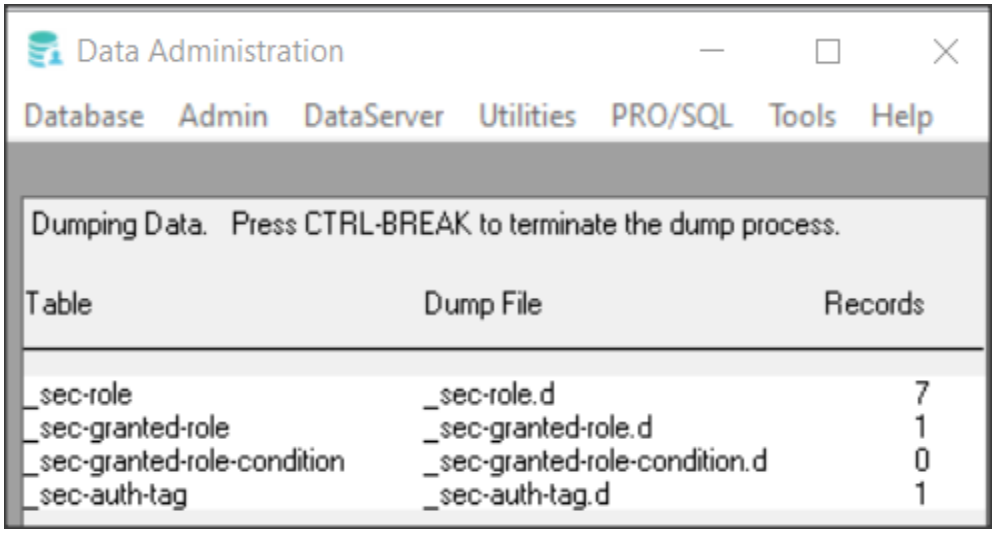
The following code associates the **#DDM_SEE_ContactInfo** authorization tag with **the myRole** role:

```
USING OpenEdge.DataAdmin.*.

VAR DataAdminService service.
YES IAuthTag oTag.
VAR LOGICAL lReturn.

service = NEW DataAdminService (LDBNAME("DICTDB")).

     oTag = service:NewAuthTag("#DDM_SEE_ContactInfo").
     oTag:RoleName = service:GetRole(" myRole "):Name.
     oTag:description = "To see contact info".
     lRETURN = service:CreateAuthTag(oTag).
```

And finally, we assign the defined role to the Admin user.

```
USING OpenEdge.DataAdmin.*.

VAR DataAdminService service.
VAR IGrantedRole oRole.
VAR IUser oUser.
VAR LOGICAL lResult = FALSE.

service = NEW DataAdminService(LDBNAME("DICTDB")).
oRole = service:NewGrantedRole().
oRole:Role = service:GetRole(" myRole ").
oUser = service:GetUser("Admin").

IF VALID-OBJECT(oRole:Role) AND VALID-OBJECT(oUser) THEN DO:
  oRole:Grantee = oUser:Name.

   IF oUser:Domain:Name NE "" THEN
      oRole:Grantee = oRole:Grantee  + "@" + oUser:Domain:Name.

    oRole:CanGrant = false. // Cannot grant to others.
    // Granting Role to Admin
   lResult = service:CreateGrantedRole(oRole).

END.
```

The next step is to define a mask for the selected fields and test examples. Now I would like to show where to look for information about the defined roles and tags.
These elements have been added to the database meta-schema, in specific new VST tables.
First, we can dump the data to files using the option in Data Administration **Admin -> Dump Data and Definitions -> Security Permissions** .
Below you can see the first lines of the two files in which the data we defined is.



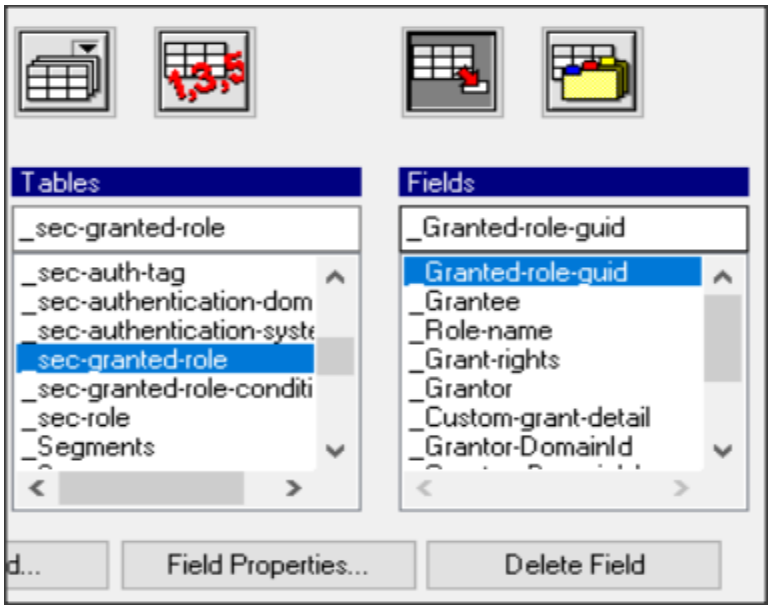
_sec-auth-tag.d

```
"myRole" "#DDM_SEE_ContactInfo" "Can see contact info"
...
```

_sec-granted-role.d

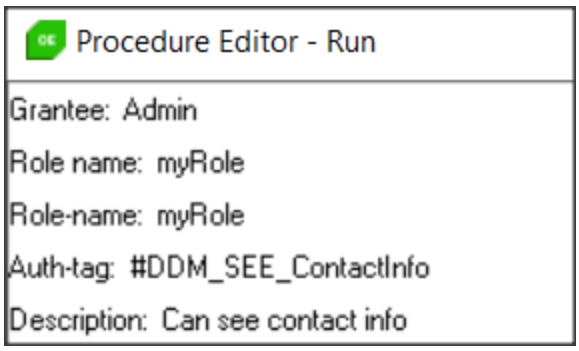
```
"f82629bb-b26d-6b83-d314-c5d870ea3fee" "Admin" "myRole" no "Admin" ""
...
```

The information in the files comes from the hidden VST tables below.



Using them we can generate the necessary information ourselves as in the example below.

```
FIND FIRST _sec-granted-role NO-LOCK.
   DISPLAY  _grantee _role-name  WITH SIDE-LABELS.

FIND FIRST _sec-auth-tag NO-LOCK.
   DISPLAY  _sec-auth-tag._role-name _auth-tag SKIP
            _description WITH SIDE-LABELS.
```



The next step is to define masks for selected fields. These masks hide some or all of the field content from unauthorized users. We have four types of such masks: default, partial, literal, null.

All DDM settings in ABL are implemented via **DataAdminService** methods. We start with the default mask, in which we have a small field for manoeuvre. Character fields are marked as **XXX** (the number X depends on the masked data), decimal fields **0.00** etc. The mask is set via the **"D:"** prefix. We will start with the **Balance** field.

```
USING OpenEdge.DataAdmin.DataAdminService FROM PROPATH.

DEFINE VARIABLE service AS DataAdminService NO-UNDO.
DEFINE VARIABLE lResult AS LOGICAL NO-UNDO.

service = NEW DataAdminService(LDBNAME("DICTDB")).
lResult = service:setDDMConfig("Customer","Balance","D:","#DDM_SEE_ContactInfo").
```

Let's set the same mask also to **the SalesRep** and **State** character fields.

For the masking settings to be visible, you need to activate the DDM service in the database with the command:
proutil sports2020 -C activateddm .

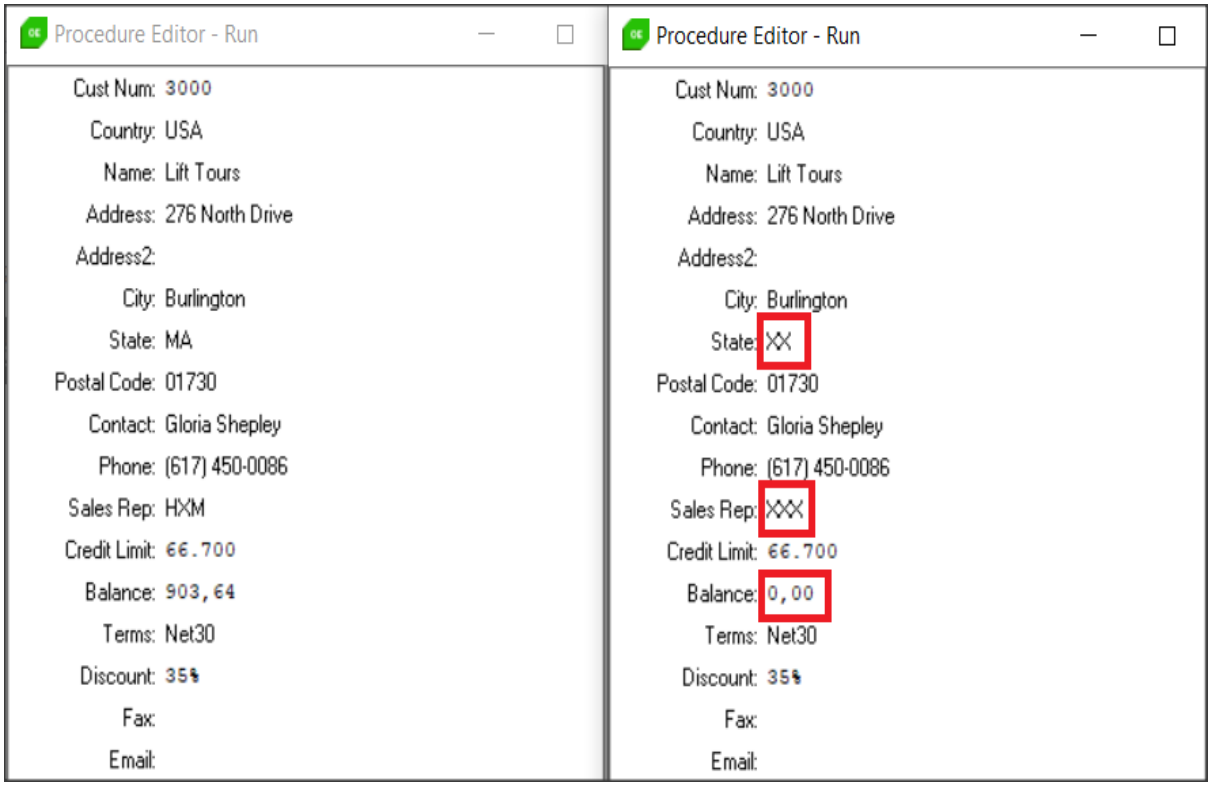
```
OpenEdge Release 12.8.4 as of Mon Aug 26 13:33:38 EDT 2024
Feature Management: The feature Dynamic Data Masking has been activated. (11184)
The DDM feature activation has completed successfully. (20697)
```

For testing, we will use a simple procedure **findCust.p** .

```
// findCust.p
FIND FIRST customer.
DISPLAY customer EXCEPT comments WITH 1 COLUMN.
PAUSE.
```

We run two client sessions with the sports2020 database by logging in once as **Admin** and once as **User**. Below you can see a comparison of data for both sessions - I marked the differences in

red. The **literal**



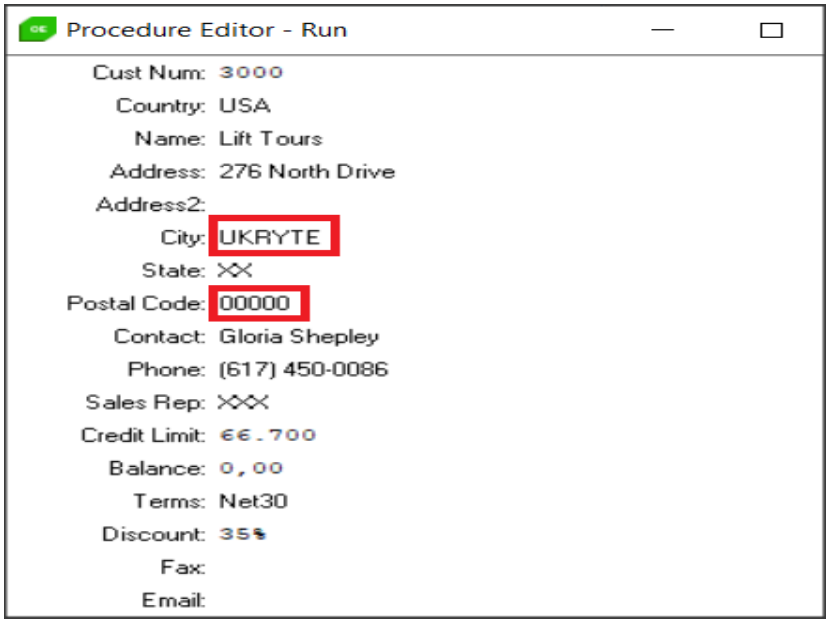
The MASK option is used when you want to replace the original data with a custom value. It's important to remember that **type compatibility** must be maintained—each mask must match the data type of the field it's applied to.

For example:

- For a CHARACTER field like City, you can use a custom mask like "L:HIDDEN".
- For a Postal Code field (e.g. INTEGER or CHARACTER), you might use "L:00000".

The "L:" prefix indicates a **literal mask**, meaning the provided value will always be shown instead of the actual data.

```
USING OpenEdge.DataAdmin.DataAdminService FROM PROPATH.

DEFINE VARIABLE service AS DataAdminService NO-UNDO.
DEFINE VARIABLE lResult AS LOGICAL NO-UNDO.

service = NEW DataAdminService(LDBNAME("DICTDB")).
lResult = service:setDDMConfig("Customer","City","L:UKRYTE","#DDM_SEE_ContactInfo").
lResult = service:setDDMConfig("Customer","PostalCode","L:00000","#DDM_SEE_ContactInfo").
```



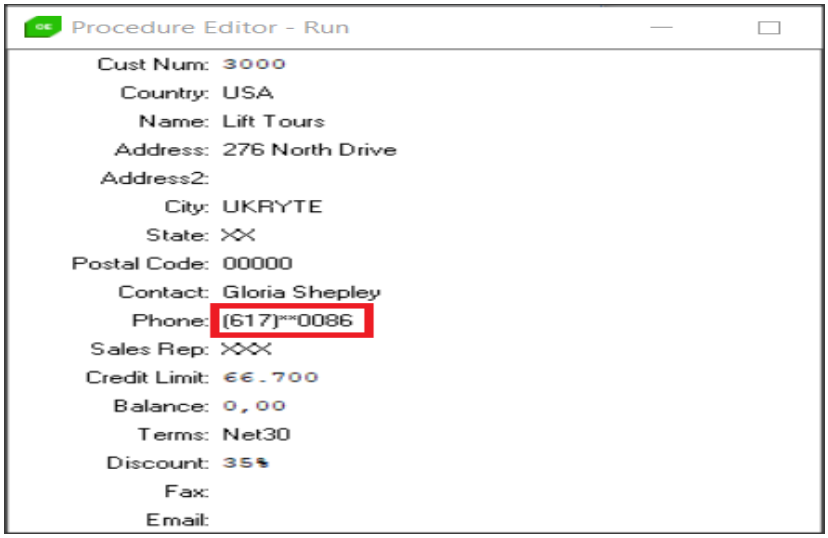
It's time to move on to the most interesting mask, the partial one, which gives the most possibilities, but is only used for character fields.
The mask syntax looks like this: **P:prefix,char[:maxchars][,suffix]**

prefix – specifies the number of unmasked characters that can be displayed at the beginning of the field.
char [:maxchars] – specifies a single ASCII character to use to hide the column value, **maxchars** (optional value) – specifies the maximum number of characters to mask using the given ASCII character, the rest of the string is truncated.
[,suffix] – (optional value) – specifies the number of characters at the end of the field that can be unmasked.

```
USING OpenEdge.DataAdmin.DataAdminService FROM PROPATH.

DEFINE VARIABLE service AS DataAdminService NO-UNDO.
DEFINE VARIABLE lResult AS LOGICAL NO-UNDO.

service = NEW DataAdminService(LDBNAME("DICTDB")).
lResult = service:setDDMConfig("Customer","Phone","P:5,*:2,4","#DDM_SEE_ContactInfo").
```

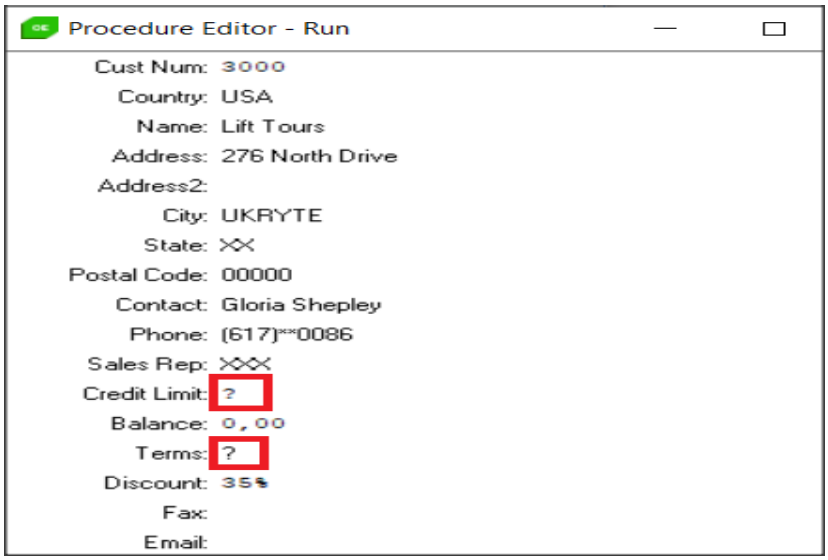
We test the Phone field. It works as follows: the first 5 characters are unmasked, as are the last 4 characters, what is inside is masked with a maximum of 2 characters " * ". If the field length is

only 10 characters, only one asterisk will be displayed, etc.



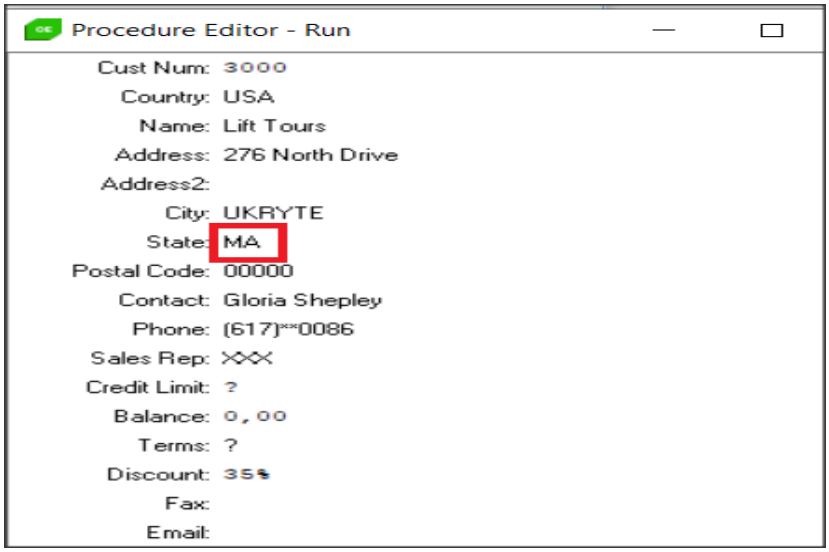
The last type of mask is null, which can be used for any data type. Masked values are displayed as " **?** ". We set the null mask using the **N:** prefix.

```
USING OpenEdge.DataAdmin.DataAdminService FROM PROPATH.

DEFINE VARIABLE service AS DataAdminService NO-UNDO.
DEFINE VARIABLE lResult AS LOGICAL NO-UNDO.

service = NEW DataAdminService(LDBNAME("DICTDB")).
lResult = service:setDDMConfig("Customer","CreditLimit","N:","#DDM_SEE_ContactInfo").
lResult = service:setDDMConfig("Customer","Terms","N:","#DDM_SEE_ContactInfo").
```



If we need to remove masking from a field, we need to use the **unsetDDMMask** method. Let's remove the mask from the State field, for example.

```
USING OpenEdge.DataAdmin.DataAdminService FROM PROPATH.

DEFINE VARIABLE service AS DataAdminService NO-UNDO.
DEFINE VARIABLE lResult AS LOGICAL NO-UNDO.

service = NEW DataAdminService(LDBNAME("DICTDB")).

lResult = service:unsetDDMMask("Customer","State").
```



Using the DataAdminService methods you can manipulate roles, authorization tags, etc. We can view, for example, the tag and mask for a given field.

```
USING OpenEdge.DataAdmin.*.
VAR DataAdminService oDAS.
VAR CHARACTER cMask.
VAR CHARACTER cAuthTag.

oDAS = new DataAdminService (ldbname("DICTDB")).
oDAS:GetFieldDDMConfig ("Customer","Phone", OUTPUT cMask, OUTPUT cAuthTag).
MESSAGE cMask SKIP cAuthTag
VIEW-AS ALERT-BOX.
```

Below you can see the data for **the Phone** field.