

# ProMatrix Corporation

## VPME 9.1 – Project Upgrade Modifications Guide

(7/18/2006)

### Purpose

VPME Enterprise 9.1 is a major upgrade of VPME that includes many significant changes and improvements to the way VPME applications work. When you upgrade an application created in a previous version of VPME to VPME 9.1, you will need to modify the application to work with the VPME 9.1 changes.

This “Project Upgrade Modifications Guide” lists and describes twenty-three (23) modifications that you may need to make when upgrading an application to VPME 9.1. The modifications that you will need to make depend on what functionality you have used in your VPME application and the extent to which you have customized the application.

This Modifications Guide should be used as a supplement to the “[Converting a Previous Version VPM Application to VPME 9.1](#)” document. Also, before converting a previous version VPME application to VPME 9.1, you should review the [VPME Enterprise 9.1 – What’s New Documents](#) that describe the many changes and enhancements in VPME 9.1.

### Modifications Guide

- “Pre” and “Post” Methods:** The “Pre” and “Post” methods of forms (PreSave, PostSave, etc.) have been removed in VPME 9.1. Therefore, if you have a form from a previous version that contains code in any of the “Pre” or “Post” methods and import that form into a VPME 9.1 project, that code will not exist in the imported form. You will need to retrieve that code and place it in an appropriate place in the VPME 9.1 project. Appropriate places include:
  - The form’s methods, such as AddNew, Copy, Delete, Restore, and Save. For example, PreSave and PostSave code could be placed in the form’s Save method as follows:

```
LOCAL llReturn
* Place PreSave code here
llReturn = DODEFAULT()
IF llReturn    && Save was successful
    * Place PostSave code here
ENDIF
RETURN llReturn
```
  - The Record-Level Data-Driven Developer Code stored in the VPME data dictionary through the Data Builder. For more information about writing Data Builder code see the Record-Level Data-Driven Developer Code section of the Building Data: Tables chapter of the User’s Guide as well as the Tips for Writing Business Rules and Data Builder Code section of the Data Handler & Business Rules chapter of the User’s Guide.
  - The methods of the Business Rules class. For more information about writing Business Rules code see the Data Handler & Business Rules chapter of the User’s Guide.
- Key Generation:** If, in a project being upgraded to VPME 9.1, you have fields that have a call to the VPME generated key program S<Prefix>DGK1 in the Default expression, you should check the key generation for those fields in VPME 9.1 after running the System Table Data Import routine. For each field the call will have been converted into a Default type of Generate Sequential with a record created in the Generated Keys table (SDATADDGK). If you used any of the generation customization parameters in the call to S<Prefix>DGK1 in the prior version you may need to make changes to the generation routine in VPME 9.1. See the GenerateKey and GenerateKey\_Increment methods of the VPMDDataHandler class if you find that you need to modify the key generation routine.
- Thermometer Form:** The way in which a Thermometer form is run has changed and therefore you will need to modify any of your code that runs a Thermometer form.

- a. The Thermometer form is now brought up by calling the RunThermometerForm method of the Application object. The RunThermometerForm method returns the object reference of the Thermometer form so the form should be run as follows:

```
<variable name> = _SCREEN.oApp.RunThermometerForm("<title>")
```

4. **Mover Form:** The way in which a Mover form is run has changed and therefore you will need to modify any of your code that runs a Mover form.

- a. The source and destination arrays that are loaded before the Mover form is brought up are no longer memory variable arrays. The arrays are now properties of the Application object, `_SCREEN.oApp.aMoverFormSource` and `_SCREEN.oApp.aMoverFormDestination`.
- b. The Mover form is brought up in the same way, by calling the RunMoverForm method of the Application object (`_SCREEN.oApp.RunMoverForm()`), except that the names of the two arrays are no longer passed as the first two parameters.

5. **Data Handling Methods:** Data handling methods (`DataChanged`, `IsNew`, `OpenTable`, `OpenView`, etc.) have been moved from the Application object to the Data Handler object. Some method names have been changed and the parameters that are passed to the methods have also been changed. See the Data Handler Class chapter of the Technical Reference manual for an explanation of the methods and properties of the Data Handler object and how to call and reference them.

6. **Encryption Methods:** Methods that perform the encryption, decryption, and check sum functionality have been moved from the Security object to the Data Handler object. Any calls to these methods in your code need to be modified. For example, from within a form the encryption method would be called as follows:

```
ThisForm.oVPMDDataHandler.EncryptString(<string to be encrypted>)
```

7. **Business Rules:** Changes have been made to the way the Business Rules object is instantiated.

- a. Calls to business rules methods are now different because the Business Rules object is now instantiated by the Data Handler object and the object reference is now stored in a property, `oBusinessRules`, of the Data Handler object. In a form a Business Rules method would now be called as follows:

```
ThisForm.oVPMDDataHandler.oBusinessRules.<method name>()
```

- b. Business Rules code can now take advantage of the fact that it is now being run in the same data session as the form for which the Business Rules method is being run. This means that Business Rules code can:
  - i. Reference cursors open in the form's data session.
  - ii. Reference properties of the form.
  - iii. Run methods of the form.
- c. Business Rules code can now run the methods of the Data Handler object to manipulate data. The object reference of the Data Handler object is passed to the Business Rules methods as a parameter value.

8. **System Data Changes:** The names and structures of system tables and views have been changed. Also, some system data is now stored in different places.
- a. Your code will need to be changed to reflect new database, table, view, field, and index tag names. Most system tables are now in either the SDATA or SVPM database. All system views are now stored in either the SDATA\_V or SVPM\_V database.
- i. S<Prefix>CC0.dbf is now SVPM!SVPMConditionFields.dbf
  - ii. S<Prefix>CC1.dbf is now SVPM!SVPMConditions.dbf
  - iii. S<Prefix>CC2.dbf is now SVPM!SVPMConditionDetail.dbf
  - iv. S<Prefix>DD0.dbf is now SDATA!SDATADDD.dbf
  - v. S<Prefix>DD1.dbf is now SDATA!SDATADDTV.dbf
  - vi. S<Prefix>DD2.dbf is now SDATA!SDATADDF.dbf
  - vii. S<Prefix>DF3.dbf is now SDATA!SDATADDFH.dbf
  - viii. S<Prefix>DGK.dbf is now SDATA!SDATADDGK.dbf
  - ix. S<Prefix>DInd.dbf is now SDATA!SDATADDIT.dbf
  - x. S<Prefix>DMV.dbf is now SVPM!SVPMSystemControls.dbf
  - xi. S<Prefix>DMVU.dbf is now SVPM!SVPMApplicationControls.dbf
  - xii. S<Prefix>ELog.dbf is now SDATA!SDATAErrors.dbf
  - xiii. S<Prefix>FAPD.apd is now SVPMApplicationDefinition.apd
  - xiv. S<Prefix>FCO.dbf is now SVPMCalledObjects.dbf
  - xv. S<Prefix>FCT1.dbf is now SVPMAdminTools1.dbf
  - xvi. S<Prefix>FCT2.dbf is now SVPM!SVPMAdminTools2.dbf
  - xvii. S<Prefix>FMnu.dbf is now SVPMMenus.dbf
  - xviii. S<Prefix>FPJD.pjd is now SVPMProjectDefinition.pjd
  - xix. S<Prefix>IL.dbf is now SVPM!SVPMLanguages.dbf
  - xx. S<Prefix>IM.dbf is now SVPM!SVPMMessageTranslations.dbf
  - xxi. S<Prefix>IS.dbf is now SVPM!SVPMStringTranslations.dbf
  - xxii. S<Prefix>Istr.dbf is now SVPM!SVPMStrings.dbf
  - xxiii. S<Prefix>SCG.dbf is now SVPM!SVPMControlPermissionGroups.dbf
  - xxiv. S<Prefix>SCGD.dbf is now SVPM!SVPMControlGroupDefinitions.dbf
  - xxv. S<Prefix>Sec.dbf is now SVPM!SVPMSecurityData.dbf
  - xxvi. S<Prefix>SFG.dbf is now SVPM!SVPMFieldPermissionGroups.dbf
  - xxvii. S<Prefix>SFrm.dbf is now SVPM!SVPMControlSecurityForms.dbf
  - xxviii. S<Prefix>SMG.dbf is now SVPM!SVPMMenuPermissionGroups.dbf
  - xxix. S<Prefix>SMnu.dbf is now SVPM!SVPMMenuOptions.dbf
  - xxx. S<Prefix>SUsr.dbf is now SDATA!SDATAUsers.dbf
  - xxxi. S<Prefix>UAct.dbf is now SVPM!SVPMActivity.dbf
  - xxxii. S<Prefix>UAud.dbf is now SDATA!SDATAAudit.dbf
  - xxxiii. S<Prefix>UMes.dbf is now SVPM!SVPMMessages.dbf
  - xxxiv. S<Prefix>URpt.dbf is now SVPM!SVPMReports.dbf

- b. Your code will need to be changed to reflect new field types. In particular, logical fields have been changed to numeric fields to allow for easier movement of system tables to remote backends.
- c. Set Integrity information that was stored in the VAL\_DATA field of the S<Prefix>DD2 table is now stored in its own Set Integrity table (SDATADDSI).
- d. Picklist information that was stored in the VAL\_DATA field of the S<Prefix>DD2 table is now stored in its own Field Picklists table (SVPMDDFP).
- e. The remainder of the information that was stored in the VAL\_DATA field of the S<Prefix>DD2 table is now stored in individual fields in the Fields table (SDATADDF).
- f. If you have run the System Table Data Import routine and imported data dictionary information, your field-level Cleanup Code will have been placed in the mPostFieldValidationCode field of the SDATADDF table (see the PostValidation code on the Code page of the Data Builder for a field). As your Cleanup Code was imported it will have been modified to accommodate the way that PostValidation code should return a value and specify a message to be displayed. However, you should check to make sure that the code will perform as initially intended.

**Note:** The code is executed directly from the memo field using the EXECSCRIPT() function and therefore is no longer placed in the program S<Prefix>VC (the S<Prefix>VC program no longer exists).

- g. If you have run the System Table Data Import routine and imported data dictionary information, your field-level validation code will have been placed in the mFieldValidationCode field of the SDATADDF table (see the validation code on the Integrity page of the Data Builder for a field). As your validation code was imported it will have been modified to accommodate the way that validation code should return a value and specify a message to be displayed. However, you should check to make sure that the code will perform as initially intended.

**Note:** The code is executed directly from the memo field using the EXECSCRIPT() function and therefore is no longer placed in the program S<Prefix>VC (the S<Prefix>VC program no longer exists).

- h. Primary Key tags were previously identified in the S<Prefix>DD1 table in the PrimaryTag field. That field contained the name of the Primary Key tag for the table or view. Now, Primary Key tags are identified in the Index Tags table (SDATADDIT), by a 1 in the nPrimary field. Also, the list of fields in the primary key tag that was previously stored in the PrimaryFld field in the S<Prefix>DD1 table is now stored in the mExpressionFields field of the SDATADDIT table.
  - i. Remote view information that was stored in the S<Prefix>DRV table is now stored in fields in the SDATADDTV table.
  - j. The 3-character project prefix that was previously stored in the S<Prefix>FPJD.PJD table is now stored in the SVPMAApplicationDefinition.APD table.
9. **Field Uniqueness:** The Unique checkbox on the Field Properties page of the Data Builder in previous versions no longer exists. Instead, the developer must define Candidate Keys in the Data Builder. Candidate Key definitions are stored in the table SDATADDCK.
  10. **Related Pages Forms:** In Related Pages forms the lRelatedPagesWithViews property no longer exists. Instead, there is a new lRelatedPagesWithViewsAndCAs property that needs to be set to .T. if the form uses views or cursoradapters for the non-ISA cursors.
  11. **Logo and Copyright Notice:** The code that displayed the logo and copyright notice in previous versions was contained in the programs S<Prefix>ULgo and S<Prefix>ULg2. That code is now contained in the mLogoCodeG, mLogoCodeM, mCRCode1, mCRCode2, and mCRCode3 memo fields in the SVPMAApplicationDefinition.APD table. If you have modified those programs in the prior version you will need to reapply those changes to the code in the memo fields in VPME 9.1 through the Options – Logo and Icons form.
  12. **System Forms:** All system forms are now instantiated from form classes. System forms (SCX and SCT files) no longer exist. The classes from which system forms are instantiated are specified in properties of the Application object (ProApp class), Security object (ProSec class), Report Manager (Report\_Manager and Report\_Manager\_Run classes), Data Manager (DataManager class), and forms that are based on the Form\_ class or subclasses. If you modified any of the system forms in a previous version of VPME then you will need

to either modify the form class in VPME 9.1 or create a subclass and modify that subclass. If you create a subclass you will need to specify the name of your subclass in the appropriate properties. For example, if you have subclassed the Data Manager class (DataManager) you will need to specify the name of your subclass in the cDataManagerForm property and possibly the cDataManagerForm\_ClassLibrary property of the Application object.

13. **Button Code Moved:** The button code on single-record forms (Form\_SingleRecord class) and one-to-many forms (Form\_Toolbar\_OneToMany class) has been moved from the Valid event to the Click event. If you have added code to the Valid event of buttons on these types of forms you may need to move that code to the Click event. Also, if you have added code to the Click event you will need to add a call to the DODEFAULT() function.
14. **Search Path:** The search path of an application is set differently in VPME 9.1. There is now a separate method of the Application object, SetPath, that sets the application's search path. The default search path now includes the default directory (where the EXE resides) and any subdirectories, plus any directories listed in the mPath field of the SVPMAApplicationDefinition.apd record. The mPath field can be edited on the Directories form and can include multiple directories separated by semi-colons.
15. **IsNew Method:** The IsNew method (now in the Data Handler object) can now identify new records in certain situations that it could not in previous versions. If you have situations where records are added to a non-buffered table or where records are added to a table for which deleted records are recycled you need to add a field, nIsNew (N 1), to those tables. For more information see the comments in the IsNew method of the VPMDDataHandler class.
16. **Field Validation:** There is a new control method, IdentifyValidationValue, that can be used to identify the value to be validated, if the value to be validated is not the default value, which is the value in the control's Value property. Code entered into the IdentifyValidationValue method should return the value to be validated.

When a control is bound to a field that is a part of a compound key, the IdentifyValidationValue method must be used to return the compound (concatenated) value to be validated. For example, in the sample application the Plan and Coverage fields make up a compound key. The IdentifyValidationValue code for the Plan field is:

```
RETURN PADR(This.Value,3)+PADR(dvsaeben.coverage,2)
```

17. **Picklist and Valid Filter:** The Filter Expression editbox on the Integrity page of the Data Builder in previous versions no longer exists. If you used this editbox in a previous version to filter the records in Picklist and Valid cursors, there is now a different way of implementing these filters. See the How to Filter Validation and Picklist Cursors section of the Creating Forms: Tips & Techniques chapter of the User's Guide for an explanation of how to set these filters.
18. **SetViewVariables Method:** There is no longer a SetViewVariables method of the Application object. Instead, view and CA filter variables are assigned values in the methods where they are used: OpenView, OpenCursorAdapter, RequeryView, and RefreshCursorAdapter. If you have code that calls the SetViewVariables method you need to instead pass the filter variable information directly to one of the methods listed above. For example, to requery a view filtered on two variables the code would be as follows:

```
DIMENSION laFilterVariables[2,2]
laFilterVariables[1,1] = "<name of first filter variable>"
laFilterVariables[1,2] = <first filter variable value>
laFilterVariables[2,1] = "<name of second filter variable>"
laFilterVariables[2,2] = <second filter variable value>
ThisForm.oVPMDDataHandler.RequeryView("<view
alias>",@laFilterVariables)
```

19. **View Filters:** When a project from a previous version is upgraded and a database is copied to the new VPME 9.1 project, the views in the database should be edited and the Find view filters modified so that the "=" operator is changed to the "LIKE" operator.
20. **Non-Data-Bound Controls:** The method of providing data dictionary functionality for controls that are not bound to a field has changed in VPME 9.1. After importing forms into a VPME project from a previous version those non-data-bound controls for which data dictionary functionality had been previously defined must be redefined using the new methodology. See the Defining Data Dictionary Properties for Non-Data Controls

section of the Creating Forms: Tips & Techniques chapter of the User's Guide for an explanation of how to set up this functionality.

21. **One-to-Many Forms:** One-to-many forms have a new property, `ICopyChildWithParent`, that when `.T.` (the default is `.F.`) allows child records to be copied along with a parent record. There is another new property, `IAskToCopyChildMessage`, that when `.T.` (the default) causes a message to be displayed prior to a parent record being copied. The message asks the user if they want to copy the child records along with the parent record. After importing forms into a VPME 9.1 project from a previous version set these properties of the imported one-to-many forms as appropriate.

**Note:** This child record copying functionality should only be enabled if the PK values of the parent cursor are automatically generated (sequential or GUID). That way, when the parent record is copied the new PK value will be generated and can be placed in the FK field of the copied child records.

**Note:** If `ICopyChildWithParent` is set to `.T.`, record-level validation must be turned off for the FK field in the child cursors. The new parent record will not yet have been saved at the point where the FK fields are validated.

22. **User Data Properties:** When a user logs into your application certain information from that user's record in the User table (`SDATAUsers`) is placed in properties of the Security object (`_SCREEN.oApp.oSec`). In VPME 9.1 these properties have been renamed, so if you have any code that references these properties you will need to modify that code to reference the new property names. For a list of the new property names see the User Data Record Layout and `oSec` Properties section of the Security chapter in the User's Guide. Please note that many of the property values are longer than they were in previous versions. For example, the User ID (`UserData_cUserID`) is now 20 characters in length instead of 10 characters.

The User record properties are only available in your application, so if your code that references these properties can be run within VPME, like when you test a form through the Object Builder, your code needs to take this fact into consideration. Your code can take this into consideration as follows:

```
IF _SCREEN.oApp.cAppPre = "APP"  
    * Run in VPME  
ELSE  
    * Run in application  
ENDIF
```

23. **Control Properties:** When you copy a form to a VPME 9.1 project from a previous version you will need to set the value of some of the control properties.
  - a. For labels that are linked to a field/control on the form the following properties need to be set if you want the Caption property of the label to be updated with the current caption from the data dictionary upon form instantiation:
    - i. `cName_DDD` (new) – The name of the database or cursoradapter library that contains the table, view, or cursoradapter specified in `cName_DDTV`.
    - ii. `cName_DDTV` (was `cDDTable`) – The name of the table, view, or cursoradapter that contains the field specified in `cName_DDF`.
    - iii. `cName_DDF` (was `cDDField`) – The name of the field for which the label's caption has been entered in the data dictionary.
  - b. If any of the other controls on the form had the `cDDTable` and `cDDField` properties filled in you will need to fill in the `cName_DDD`, `cName_DDTV`, and `cName_DDF` properties. These three properties do not normally need to be filled in because they will be filled in automatically at runtime if the `ControlSource` property contains an `<alias>.<field name>` that allows VPME to find the field's data dictionary record.
  - c. If you used in a previous version the `cLookupTableOrView` property of a textbox or combobox to identify the table, view, or cursoradapter to be used to provide the lookup description of a FK value, you will now need to use the `cLookupDDDName` and `cLookupDDTVName` properties to identify that table, view, or cursoradapter.