



PDS Advanced Report Writing Guide



With the Advanced Report Writer (ARW), you can create customized reports from the ground up, using elements of report design and programming.

You can embed code in reports to control and format the information that prints. You can also use Pascal scripting language to create additional layout windows and control events around reports. Although some programming knowledge is helpful, it's not required to create ARW reports.

 **Note**

This guide assumes that you're familiar with the PDS program you're working in, basic mouse operation, as well as standard Windows menus, toolbars, and command buttons.

Get Started

- [Creating an Advanced Report](#)
- [Report Properties and Settings](#)
- [The Report Editor](#)
- [Objects](#)
- [Report Tree and Data Tree](#)
- [Templates](#)
- [Sub-Reports](#)
- [Groups](#)
- [Regions](#)
- [Embedding Code and Data](#)
- [Embedding Lists in Letters](#)
- [Creating Multiple Letters in One](#)
- [Printing Letters in Multiple Languages](#)
- [Scripting Language](#)
- [Accessing and Adding Screens](#)
- [Special Sorts](#)
- [Exporting Fields Using Scripting](#)
- [Importing Data With Scripting](#)
- [Appendix](#)
- [Exercises and Example Project](#)

Creating an Advanced Report

Just like creating a regular report, there are two ways to create an advanced report. You can copy an existing report and edit it in the ARW, or add a new advanced report from scratch.

Copying an Existing Report

It's easier to modify an existing report than to create a new one in the Advanced Report Writer.

1. On the Reports tab, locate a report that's similar to what you need.
2. Select the report, and click **Copy**.
3. Select the copied report, and click **Adv. Script**. If the Type isn't already **Adv Report**, change it and click **Save/OK > Close**.
4. Click **Next** until you reach the layout window.
5. Click **Modify the Report**.
6. The Advanced Report Writer opens for you to edit your report.

See the rest of this guide for more on editing your report.

Adding a New Advanced Report

If there are no reports similar to what you need, you can create a new one.

1. On the Reports tab, click the report type you want to add.
2. In the Select Report window, locate the group where you want to add the report.
3. Click **Add**, then on the **Advanced Reports** tab, click the type you want to create.
4. The Advanced Report Writer opens for you to edit your new report.

See the rest of this guide for more on editing your report.

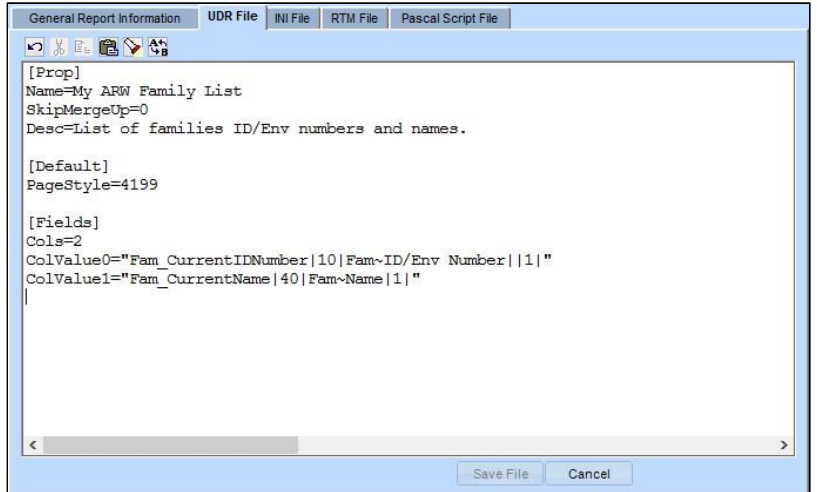
Report Properties and Settings

Each advanced report has a .udr file and an .ini file that contain information about the report's properties and settings. You can edit these as needed.

Properties in UDR Files

The .udr file provides report properties, defaults, and other data. Select the report, click **Adv. Script**, and click the **UDR File** tab.

Each section is preceded by the type in square brackets, such as [Prop] for the Properties section. Each line is a different setting with a format of **Object =Setting**, such as *Name=Family Directory*.



Properties [Prop]

- **Name=** report name
- **Desc=** report description
- **SkipMergeUp=** 0 (false) or 1 (true)

Defaults [Default]

In this section, you can assign values you want the report to use the first time it is run.

- **SortBy=** Name, ID Number, or Second ID Number
- **ForceSortBy=** Name (The default sort is by Name and only name can be selected.)
- **Landscape=** 0 (landscape off) or 1 (landscape on)
- **DisableScaling=** 0 (enable scaling) or 1 (disable scaling; this works best for forms and certificates)
- **PageStyle=** XYZ
- **MarginStyle=** XYZ
- **LHStyle=** XYZ (Letterhead style)
- **DateFmtStyle=** XYZ (Date Format style)
- **InsideAddrStyle=** XYZ (Inside Address style)
- **ClosingStyle=** XYZ
- **LabelStyle=** XYZ

After you run the report, the .ini file holds the values you entered. Those values can in turn be used to set XYZ, which the report then uses as the default for that style. For example, *PageStyle=362* defaults the report to Compress Print. Check out other [UDR Options](#) to control your reports.

Information from the Last Run [LastInfo]

This section holds initialization variables and values used by the Pascal Script File. For more information about these parameters, see [Embedding Code and Data](#).

Field Information [Fields]

Listing reports contain a line for each field and its settings.

- **Cols=** number of columns
- **ColValue#=** field | width or indent | heading || indent total or 1 | label

For columns that are "New Column", **Indent total** is equal to 1. For columns that are "Same Column and Line", **Indent total** is the total width of the column, including widths from the main New Column and indent values from Same Column and Line.

For example, if you want column 1 to be a New Column with a width of 10 and column 2 to be a Same Column and Line with a width of 3, enter the following:

- **ColValue1=** field|10|heading||1|label
- **ColValue2=** field|3|heading||13|label

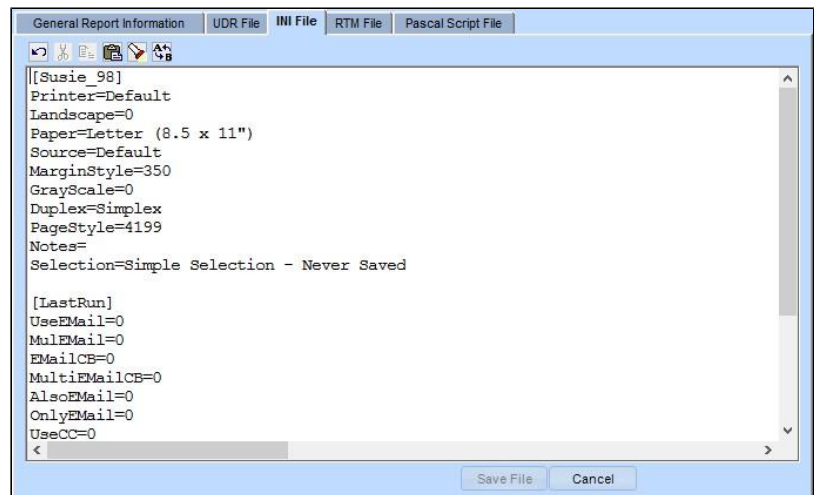
A tilde (~) represents a soft carriage return in the heading.

Settings in INI Files

The first time you run an advanced report, an .ini file is created. This file saves and updates the printer, report, style, and other settings each time you run the report. Select the report, click **Adv. Script**, and click the **INI File** tab.

Computer Name

One of the sections is the computer name, such as "[Susie_98]" in the example above. This section saves the last printer settings used by this computer, including the printer, orientation, paper size, paper source, margin style, gray scale, page style, and notes.



Last Info [LastInfo]

This section contains variables and values used in the Pascal Script File from the last time the report was run.

Last Run [LastRun]

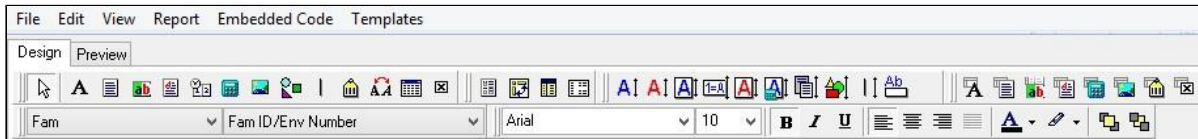
This saves the built-in selection settings from the last time the report was run, such as **UseSepStmt** (use separate statement) and **UseEMail** (prefer email).

The Report Editor

When you add a new advanced report or modify one, the ARW editor displays. The editor contains toolbars and the workspace.

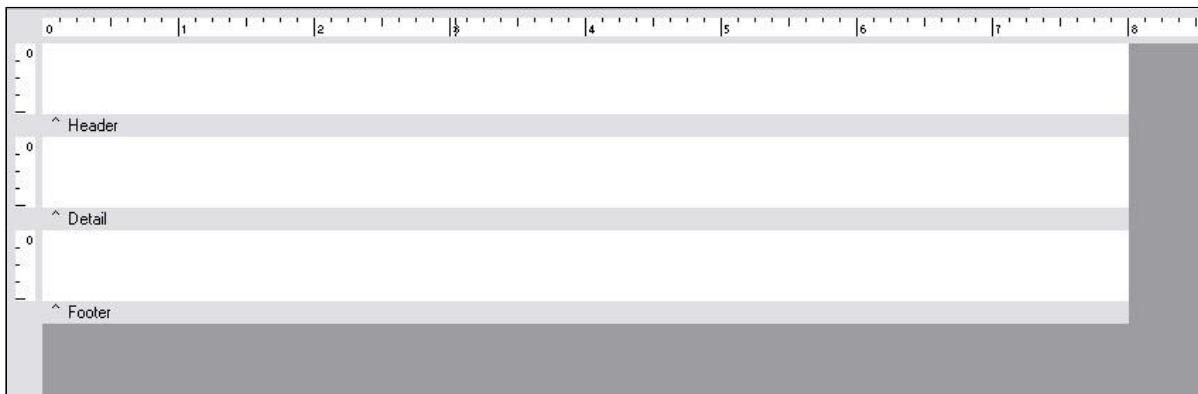
Toolbars

You use the buttons on the toolbars to insert [objects](#) into your report and format report elements. To view or hide a toolbar, click **View > Toolbars**, and select or clear the toolbar.



The Workspace

When you open a new report, the workspace is divided into three bands: Header, Detail, and Footer. Objects in the Header display at the beginning of each page of the report, and objects in the Footer display at the end of every page. Objects in the Detail band print once for each item inside this band.



You can add two other bands, Title and Summary. Click **Report > Title** or **Summary**. The Title displays at the top of the report, before the Header on the first page. The Summary displays at the very end of the report, before the Footer on the last page.

To resize workspace bands, click and drag the gray bars up or down. You can set the band's height to readjust to match its contents or stay static. Right-click the gray bar and select an option:

- **DynamicHeight** – The vertical size of the band fluctuates depending on the objects inside. This eliminates white space between printed families.
- **StaticHeight** – The vertical size of the band remains constant, regardless of the size of the objects inside. This creates extra white space between printed families.

Objects

Objects are pieces of information you can display in the report.


Tip

Click **View > Toolbars** to display additional toolbars to see all button options.


Generic Objects

Generic objects are not connected to a particular data field.


Label

- A label is an embedded piece of text that doesn't change when you print the report.
- Click the label button , and enter the label text in the input box.
- You can apply any text formatting (bold, italics, underline, color, highlight) on the label text, and you can change the font style and size.


Memo

- A memo is a large box that can have multiple lines of text.
- Click the memo button .
- After you insert the memo object, right-click it, select **Lines**, then enter the text. Click **Load** to read in text from an outside .txt file. Click **Save** to save the memo as a .txt file.


System Variable

- A system variable is a field like date or time.
- Click the system variable button , and select the type of variable you want.
- You can change the format of the variable. Right-click the object, select **Display Format**, then select a display option.


Variable

- A variable is a numeric or string calculation that isn't in the data or in system variables.
- These can be used to create values from other values or fields. For example, a directory list has the first letter of the last name at the top of the column. That letter is a variable calculated from the name object.
- Click the variable button .
- To set the calculation, right-click the object and select **Calculations**. The Pascal programming dialog box displays where you can enter [embedded code](#).


Image

- Click the image button . Right-click the object, select **Picture**, and select the image you want.
- You can insert images such as scanned letterheads, signatures, clipart, and so on.

Shape

- Click the shape button , and select the type of shape you want.


Line

- Click the line button , and select where you want it to display.


Data-Sensitive Objects

Data-sensitive objects display actual program data in your report.

DBText

- The DBText object is directly connected to your database so it can display record information.
- Click the DBText button , and select the type of field category (called the "data pipeline") and the field name.
- The data itself is usually larger than the object size. Right-click the object and select **AutoSize** to automatically fit the data.
- If the data field you selected has a special format, such as dates or phone numbers, right-click it and select **Display Format**.


DBMemo

- A DBMemo object is the same as a DBText field, but it's larger. These are designed for fields that use multiple lines of text, such as phone lists, talent/ministry lists, and so on.
- Click the DBMemo button .

DBCalc

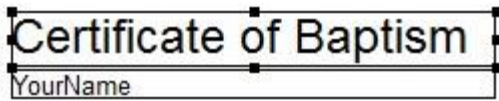
- The DBCalc object counts or totals a data-sensitive numbers, like financial fields.
- Click the DBCalc button, and select the field category and data field you want to base the calculation on. Then, right-click the object, select **Calculations**, and select the Calculate Type.
- When you run the report, the result of the calculation displays on the selected field.
- This is normally added to the Summary band so the program can calculate based on all the data in the Detail band.

DBImage

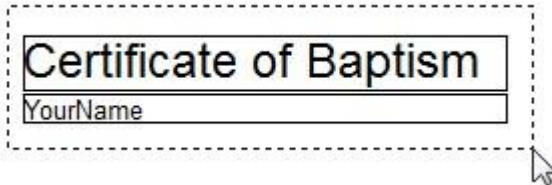
- The DBImage object displays a picture from your database, such as a member's picture.
- Click the DBImage button , and select the field category and data field you want. It should be an image-based field, such as Mem Picture.

Selecting an Object

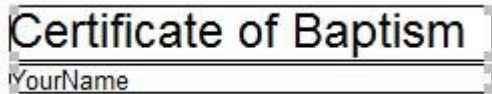
To select an object you added to the workspace, click the object. This creates a dark border around it, with small black boxes in the corners and sides.



To select several objects at once, hold the **Shift** key as you click the objects. Or, you can click-and-drag a selection box around them.



When you let go of the mouse, all items in the box are selected. The selected objects have small gray boxes in the corners.



To deselect an object or objects, click any empty portion of the workspace.

Selecting Embedded Objects

Some objects are embedded within others. In this case, when you try to select the embedded object, you actually select the entire thing.





To select an object embedded within another, hold the **Ctrl** key before using the mouse. If you accidentally select the outside object, hold down the **Shift** key and click the object to deselect it.

Moving an Object

After you insert and select an object, the next step is to move it where you want it to display in the report. You can click and hold it, drag it where you want it, then release the mouse button.

In order to line up an object with those near it, the ARW editor has minor adjustment buttons. To display these in the editor, click **View > Toolbars**, and select **Nudge**, **Size**, and **Align or Space**.

Nudging

Use the nudge buttons to move an object exactly one pixel up , down , left , or right . You can also hold down the **Ctrl** key and use the arrow keys on your keyboard.

Growing and Shrinking

Use the following buttons to grow or shrink several objects to the same size.



Shrink Width to Smallest – Reduces widths of selected objects to match the narrowest one.



Grow Width to Largest – Increases widths of selected objects to match the widest one.



Shrink Height to Smallest – Lowers heights of selected objects to match the shortest one.



Grow Height to Largest – Raises heights of selected objects to match the tallest one.

Aligning and Spacing

Use the following to align or space several objects. Objects align with the first object you select.



Align Left – Aligns objects along their left edges.



Align Middle – Aligns objects in a vertical stack around a common center.



Align Right – Aligns objects along their right edges.



Align Top – Aligns objects along their top edges.



Align Center – Aligns side-by-side objects along a common center.



Align Bottom – Aligns objects along their bottom edges.



Space Horizontally – Puts an equal amount of horizontal space between selected objects.



Space Vertically – Puts an equal amount of vertical space between selected objects.



Center Horizontally – Centers objects between the left and right margins.



Center Vertically – Centers objects between the top and bottom margins.

Right-Clicking an Object

You can right-click an object to view and change its options or properties. An ellipsis [...] in the property name indicates additional controls.

- **Bring to Front/Send to Back** – Moves object in front of or behind other objects.
- **Double/Single** – (Lines only) Changes the line to a single or double line.
- **Autosize** – Resizes object horizontally to match its contents. See also [Stretch](#).
- **BlankWhenZero** – Prints a blank line instead of "0" if the object's value is zero.
- **Calculations...** – For DB fields, select the type of calculation (Sum, Count, Max, Min, Average). For variables, modify the value or formula.
- **CalcOrder** – Indicates what order to calculate groups.
- **CharWrap** – Breaks words in the middle of the word when wrapping. See also [WordWrap](#).
- **DisplayFormat...** – For DBText fields, set the format of the field, such as date format, number format, and so on.
- **ForceJustify** – For Memo fields, justifies text to the left or right.
- **KeepTogether** – Does not split words when wrapping text inside a memo box.
- **Lines...** – In a Memo field, inserts the text of the memo.
- **LookAhead** – (Variables, DBCalc fields, and System Variables) Select when you need to display the calculation result or variable before the objects that are used in it. For example, displaying the DBCalc field Count in the Header, instead of the Footer or Summary. To use this, you must tell the report to run through your database twice. Click **Reports > Pass Setting > Two Pass**.
- **ParentHeight** – (Lines only) Make object the same height as the workspace.
- **ParentWidth** – Make object the same width as the workspace.
- **Position...** – Indicates position of object in the workspace, as well as object's height and width.
- **ReprintOnOverflow** – If any stretched object goes to a second page, it reprints on the next page.
- **ReprintOnSubsequent** – Reprints object the first time it occurs on subsequent pages.
- **ResetGroup** – Resets object's value to zero each time the group's breaking field value changes.
- **ShiftRelativeTo...** – (Memos and Sub-Reports) Keeps the position of object static with respect to another object.
- **ShiftwithParent** – (Memos and Sub-Reports) Keeps the position of object static with respect to the workspace band it's in.
- **StretchwithParent** – Resizes object horizontally to match the workspace.
- **Stretch** – Resizes object vertically to match its contents. See also [Autosize](#).
- **Timing...** – (Variables only) Determines when variable is calculated relative to other variables.
- **Transparent** – Makes object transparent so objects behind it can be seen.
- **Visible** – Makes object visible. Turn this off if you're using a variable for calculations but don't want to see the variable.
- **Wordwrap** – Does not break in the middle of the word. Wraps whole words only. See also [Char Wrap](#).

Report Tree and Data Tree

You can use the Report Tree and Data Tree to find your objects and associate fields with them.

Report Tree

The Report Tree displays an outline of the report and the objects. Click **View > Toolbars > Report Tree**.



The Report Outline section breaks down the different sub-reports. The Object Outline displays each band and the objects within them.

Useful Information

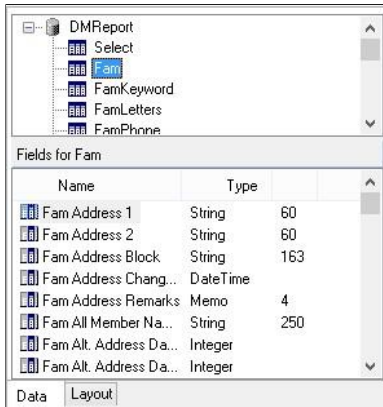
To rename sub-reports and objects, right-click the name and enter a more descriptive name (such as FamilyName). Don't use spaces in the name.

Data Tree

The Data Tree displays information about fields. It's divided into the Data tab and the Layout tab. Click **View > Toolbars > DataTree**.

Data Tab

The Data tab displays available fields and field categories.

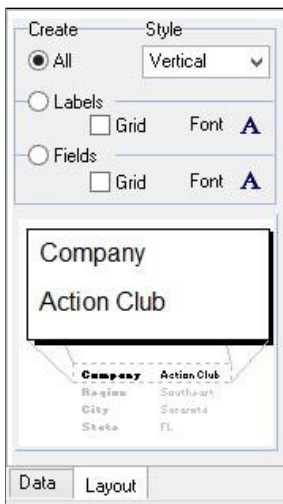


The top section outlines the data categories, such as Fam, FamKeyword, and so on. The bottom section lists the fields that belong to the selected category and information about the field.

You can drag-and-drop a data field directly from the Data Tree into your report. The program automatically creates a label for the field, which you can modify.

Layout Tab

You can indicate if and how a label is included when a data field is dropped from the Data Tree.



Select **All** to add a label at the same time the field is created. Select a **Vertical** or **Tabular** style. Select **Grid** to display the field and/or label in a grid. Click **Font** to change the font style and color.

Or, if you only want a field to be created, select **Field** and set its properties. If you only want a label to be created, select **Label** and set its properties.

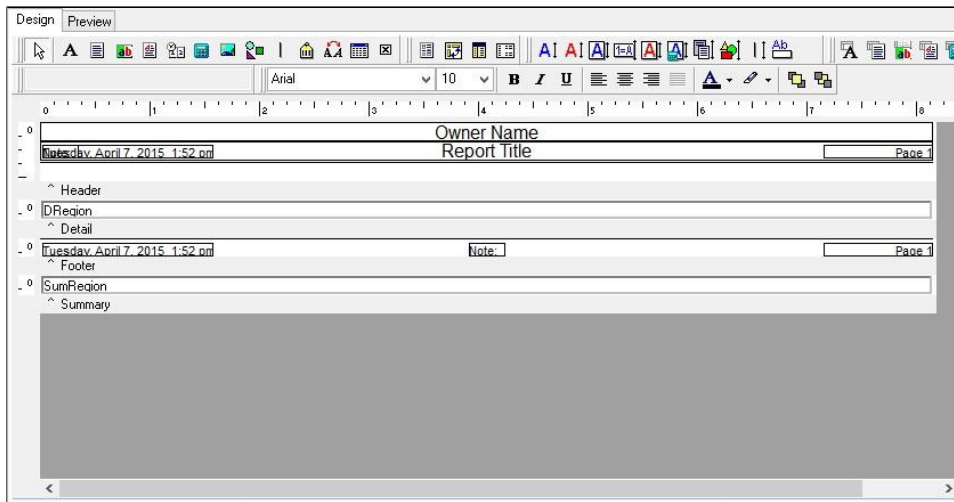
Templates

Use report format templates to quickly set up a report with predefined headers and footers. Open the advanced report writer editor, and click **Templates** on the title bar.

Available Templates

PDS Easy List Template

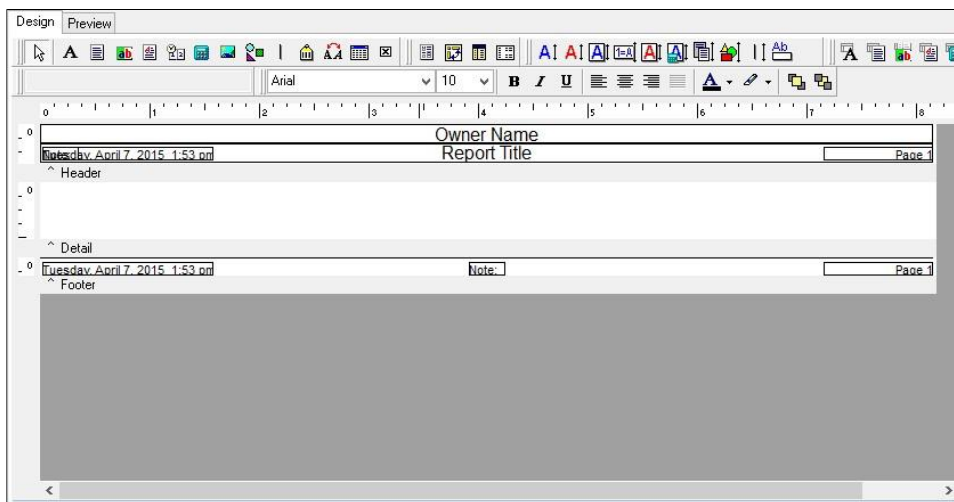
Use this template to apply the header and footer information of predefined listing reports to your ARW report. In your report, click **Templates > PDS Easy List**. After making selections, click **Build**.



You can modify, rearrange, or remove any of this. The template provides a starting point.

PDS Header-Footer Template

This template places the listing style header and footer around the Detail bands you created. Click **Templates > PDS Header/Footer**.



The Owner Name label displays the parish name as it's entered in the License Information window. The Report Title label displays the report name as it's entered in the Name field on the report Overview window. Depending on the page style, you can position the date and page number (system variables), and the note (label) in the header or footer. The template allots fields for either.

Fonts and Page Style

All regular objects in an advanced report have a customized font. If you want a field to have the same font you selected for the page style, right-click the object and select **Font Changes**.

The page style is set in your report's layout window.


General Information	Fonts	Lines and Shading
Title Font:	<input type="text" value="Arial,12"/>	<input type="button" value="Set Font"/>
Heading Font:	<input type="text" value="Arial,10"/>	<input type="button" value="Set Font"/>
Detail Font:	<input type="text" value="Arial,10"/>	<input type="button" value="Set Font"/>
Date/Page Font:	<input type="text" value="Arial,8"/>	<input type="button" value="Set Font"/>

Preview

To see how your report will print, click the **Preview** tab. To print the report, click the Printer icon. To view the report at different sizes, click the sizing icons. To scroll through pages of the report, click the arrow buttons.

Sub-Reports

A sub-report is a report within a report. For example, you can have a main report with a list of family names and a sub-report with a list of family members in each family.

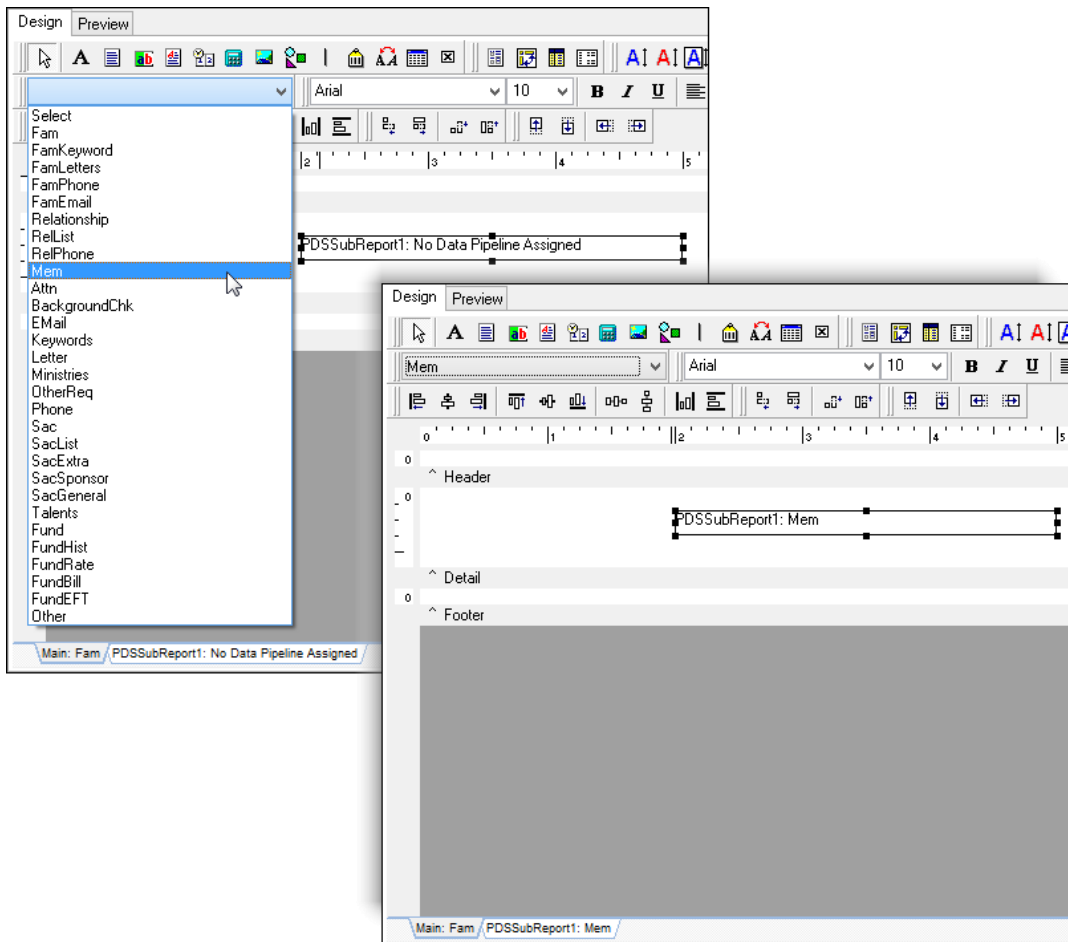
Click the SubReport button , then click where you want to add it in the workspace. A tab displays at the bottom of the window, initially called "PDSSubReport1: No Data Pipeline Assigned".

The sub-report automatically sizes to match the workspace width. To manually resize it, right-click the object, and click **ParentWidth**.

You can nest sub-reports inside other sub-reports.

Data Pipeline

The data pipeline determines which data fields your sub-report can use. Click the sub-report object and select the data category from the drop-down list.



Modifying a Sub-Report

To modify the contents, click the sub-report's tab at the bottom of the workspace.

You can insert objects in a sub-report the same way you do in the main report. Anything in the Detail band is repeated once for each item.

Groups

A group is a special type of band that contains a Detail band surrounded by a Header and Footer. You can use a group to keep the objects of the Detail band together on a page, or to indicate when to start the next Detail band on the next page.

For example, let's say you have the family name, address, and email in the Detail band. As the report prints, a family's information may be split between two pages. If you create a group around the Detail band, the family information will always print together.

Adding a Group

To add a group, click **Report > Groups**, and select a field as the **Break On** point. When the report is processed, the program sees this field and knows to keep that group together.

- **Data Field** – Includes all available fields within the data pipeline in the Groups drop-down list, even if they're not used in the report.
- **Custom Field** – Includes only objects in the report in the Groups drop-down list.

In the Groups drop-down list, select a field, then click **Add**. The field name displays as "Group[x]:*fieldname*" or "Group[x]:*objectname*". The number within the brackets [] is the group's break hierarchy. Groups with a lower number take priority over groups with higher numbers. For example, Group[0] is kept together first, followed by Group[1], then Group[2], and so on.

You can add multiple group break points, and you can mix Data Field and Custom Field groups.

Other Group Options

You can select how the new group prints on the report.

- **Start new page** — Starts each group on a new page.
- **Reset page number** — If you start each group on a new page, this resets the page number.
- **New page when less than** — When the space remaining at the bottom of the page (not including the bottom margin) is less than the value you enter, a new page is started.
- **Keep group together** — Keeps everything in the group together on one page.
- **Reprint group headers on subsequent pages** — Prints the group header information at the beginning of each page. For more information, see the section below.

Group Header and Footer

These print at the top and bottom of each group, just as the Header and Footer prints at the top and bottom of each page. For example, if the Group Header has a series of labels to identify columns, these labels print above each family.

In the example below, **Fam Unique ID** is selected as the group **Break On** value.

The screenshot shows a software interface with a report layout. At the top, there are fields for 'Owner Name' and 'Report Title'. Below this is a 'Header' section with three columns: 'ID', 'Name', and 'Phones'. The 'Name' column is further divided into 'Address' and 'E Mail'. A 'Group Header(0): Fam Unique ID' is shown below the main header. The first data row (ID 1) displays the following information:

ID	Name	Phones
1	Van Loon, Jeff(Jeane), M/M P.O. Box 322 Peoria, AZ 85338 jvanloon@yahoo.com	(602) 278-9932 Home (602) 344-2334 Cell (602) 388-4324 Unl. Alternate

Below the data row is a 'Detail' section, followed by a 'Group Footer(0): Fam Unique ID' and a 'Footer' section.

Each unique family is treated as a group with the header printed above it.


		Owner Name Report Title
ID	Name Address E Mail	Phones
1	Van Loon, Jeff(Jeane), M/M P.O. Box 322 Peoria, AZ 85338 jvanloon@yahoo.com	(602) 278-9932 Home (602) 344-2334 Cell (602) 388-4324 Unl. Alternate
ID	Name Address E Mail	Phones
2	Harcourt, Elizabeth, Mrs. 8745 N Forest Grove Ln Aspen, CO 81611	(602) 646-1937 Unl. Home (303) 925-2222 Alternate Addr.

Regions

A region is a group of objects that are handled as a unit. It differs from a group because it doesn't encompass the entire Detail band and has no break points. It differs from a sub-report because it doesn't have its own tab or its own Header, Detail, or Footer bands, and it cannot have a different data pipeline than the band it's in.

All objects in the region share the properties you give to the region. For example, if the region is visible, then all objects in the region are visible.

Adding a Region

Click the Region icon , and click in the workspace. Then, drag-and-drop objects into the region. If you move the region, all objects inside move with it. Drag-and-drop objects outside of the region to remove them.

By default, regions have an outline. To remove it, right-click the region and select **Transparent**.

Right-Clicking a Region

You can right-click a region to view and change its options or properties. An ellipsis [...] in the property name indicates additional controls.

- **Bring to Front/Send to Back** – Moves region in front of or behind its objects. If you don't select **Send to Back**, when you move the region over other objects, those objects are blocked.
- **KeepTogether** – Keeps all parts of a region together on one page.
- **ParentHeight** – Makes region the same height as the workspace.
- **ParentWidth** – Makes region the same width as the workspace.
- **Position...** – Indicates region's position in the workspace, as well as region's height and width.
- **ReprintOnOverflow** – If any stretched region goes to a second page, it reprints on the next page.
- **ShiftRelativeTo...** – Keep region's position relative to another sub-report or object.
- **ShiftWithParent** – Keep region's position relative to the workspace.
- **Stretch** – Expand region to encompass its objects when the report prints.
- **StretchWithParent** – Expand region to match the workspace when the report prints.
- **Transparent** – Turn off the outline around a region.
- **Visible** – Make region and its contents visible or hidden.

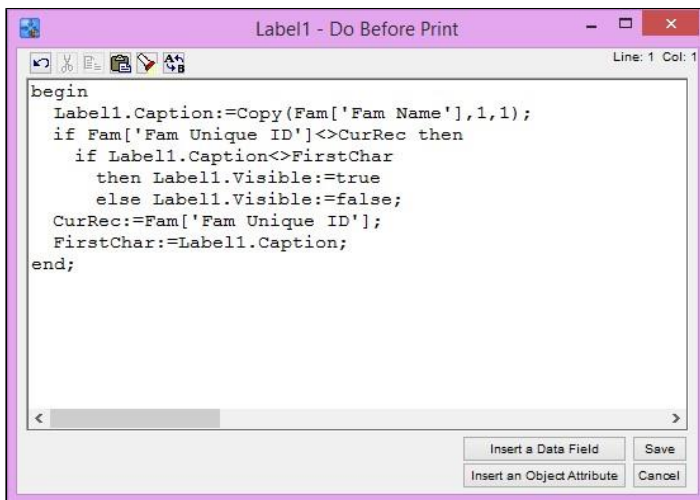
Embedding Code and Data

You can embed pieces of [code](#) in your advanced reports. Embedded code is a list of instructions for the computer. You can also embed [data](#). The code uses and manipulates the data to perform complicated functions in the report.

Code Editor

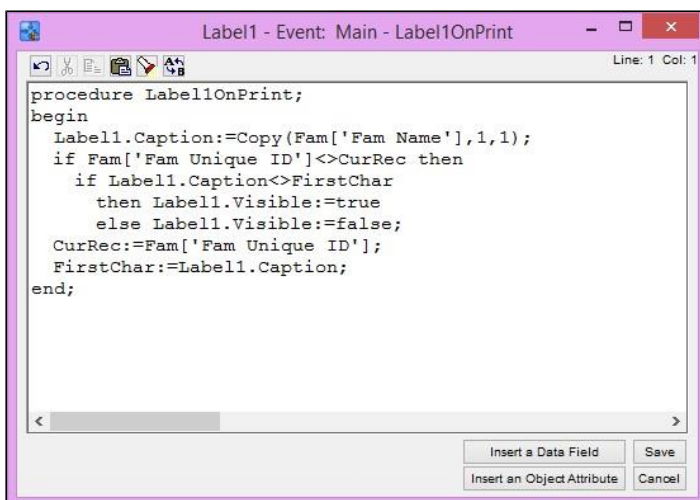
You can enter code for an event or object. There are a few ways to get to the code editor. Right-click an object and select **Do Before Print** or **Get Text**. If the object is a variable, right-click it, and select **Calculations**. To open the code editor for global variables, click **Embedded Code > Define Global Variables** or click **Embedded Code > Initialize Global Variables**.

Below is an example of what might display in the Do Before Print code editor.



```
begin
Label1.Caption:=Copy(Fam['Fam Name'],1,1);
if Fam['Fam Unique ID']<>CurRec then
  if Label1.Caption<>FirstChar
    then Label1.Visible:=true
    else Label1.Visible:=false;
CurRec:=Fam['Fam Unique ID'];
FirstChar:=Label1.Caption;
end;
```

After you enter codes for events and objects, click **Embedded Code > View All Embedded Code** and select an event or object to view. When opened this way, the code editor indicates which event or object the code belongs to in the procedure name. Below is the same code as the example above, but viewed from the View Embedded Code menu. Note the procedure name is added as "Label1OnPrint" at the top.



```
procedure Label1OnPrint;
begin
Label1.Caption:=Copy(Fam['Fam Name'],1,1);
if Fam['Fam Unique ID']<>CurRec then
  if Label1.Caption<>FirstChar
    then Label1.Visible:=true
    else Label1.Visible:=false;
CurRec:=Fam['Fam Unique ID'];
FirstChar:=Label1.Caption;
end;
```

Code

PDS uses the Pascal language to create advanced reports. In Pascal, code is divided into routines. Each routine can be a single statement or a group of statements. Each statement must end with a semicolon. It helps to indent parts of code so routines are easier to read.

Most of the code embedded in a report is associated with an event or an object. Right-click an object to see the attributes and the events, which display at the bottom. Select an event in the list to open the code editor for that event.

By default, the code editor has a BEGIN...END statement, and it also inserts additional code to help you with a particular object. For example, in the Get Text event of a label, the code editor automatically inserts a TEXT:= statement since you're most likely going to set the text of the label.

BEGIN-END Statements

Use BEGIN...END statements to group other statements or to create a block of statements. Indent the statements inside the block for easier reading.

Local Variable Declarations

To create local variables, add a declaration statement before the very first BEGIN in an event. Begin the variable declaration with "var" then list the variable names and types. Separate names and types with a colon, and separate individual variables with a semicolon. If you have multiple variables of the same type, enter the names separated by commas, then a colon, then the type.

```
var
    MyName: string;
    ItemCount, ItemMax: integer;

begin
end;
```

Assignment Statements

Use assignment statements to set a value into a variable or attribute. It can be a simple value or a complex calculation. Enter the variable, followed by a colon and equal sign, then the value or calculation.

```
a:=5;           {assign a value}
Total:=Total+100.00; {assign a calculation}
Label1.Caption:='The Family is inactive'; {assign a string}
```

IF-THEN Statements

An IF statement tells the program that if a condition exists, run a set of instructions.

This statement is made up of the word "if", a comparison or calculation that can be evaluated as true or false, the word "then", and finally a statement or group of statements to run if the IF condition is true.

```

if (Mem['Mem Age']>=21) then           {If a member is age 21 or older then...}
    begin
        Labell.Caption:='Can Supervise';      {they can supervise.}
        Labell.Visible:=true;
    end;

```

You can also use an ELSE statement to indicate what to do when the IF condition is not true.

```

if (Mem['Mem Age'].<21)
    then
        begin
            Labell.Caption:='Can Supervise';
            Labell.Visible:=true;
        end;
    else {Else, if the member is under 21...}
        begin
            Labell.Caption:='Cannot Supervise';    {they can't supervise,}
            Labell.Visible:=true;
            Labell.Font.Color:=clRed;              {and label displays text in red.}
        end;

```

CASE Statements

If you have several IF statements, use a CASE statement to make your code easier to read. This can replace a series of IF...THEN statements.

Series of IF-THEN-ELSE statements

```

if i=1
    then month:='January'
else if i=2
    then month:='February'
else if i=3
    then month:='March'
else if i=4
    then month:='April'
else if i=5
    then month:='May'
else
    month:='Other';

```

Instead of the code above, you can use a CASE statement:

```

case i of
    1: month:='January';
    2: month:='February';
    3: month:='March';
    4: month:='April';
    5: month:='May';
    else month:='Other';
end;

```

WHILE Loops

Use a WHILE loop to run a set of instructions multiple times while a condition exists.

```
i:=10-length(s);
while i>1 do
  begin
    s:='0'+s;
    i:=i-1;
  end;
```

The code above takes a variable string "s" and pads it with leading zeroes, making the value "s" at least ten characters long.

REPEAT-UNTIL Loops

The REPEAT-UNTIL loop is similar to the WHILE loop, but it tests the condition after running the statement, and it repeats the statement(s) until the condition is true.

```
Repeat
  s:='0'+s;
  i:=i-1;
until i=0;
```

FOR Loops

If you need to run a group of statements a specific number of times, use the FOR loop. You must declare an integer variable to use in the FOR statement.

```
s:='';
for i:=1 to 9 do
  begin
    s:=s+chr(48+i);           {chr(48) is the zero character, 0}
  end;

{this code is the same as saying s:='123456789'}
```

You can use "down to" if you want to go backwards.

```
s:='';
for i:=9 down to 1 do
  begin
    s:=s+chr(48+i)
  end;

{this code is the same as saying s:='987654321'}
```

If you want to jump in increments other than 1, use "step".

```
s:='';
for i:=1 to 9 step 2 do
  begin
    s:=s+chr(48+i)
  end;

{this code is the same as saying s:='13579'}
```


Built-In Procedure Statements

There are two built-in procedures: **MessageBeep** and **ShowMessage(str)**. The MessageBeep generates a beep sound, and the ShowMessage() displays a dialog box with the message you enter in the parentheses. It can be a literal string, a string variable, or a string calculation.

There are other built-in routines, but they deal with [scripting language](#).

Conditions, Calculations, and Comments

Conditions

The conditions used in the IF, WHILE, and REPEAT statements can be simple comparisons or complex calculations. The following are common comparison operations used in IF statements:

- > Greater than
- >= Greater than or equal to
- = Equal to
- < Less than
- <= Less than or equal to
- <> Not equal to

You can also use the following Boolean operations:

- **And** — Both or all conditions must be true.
- **Or** — Either or any conditions must be true.
- **Xor** — Only one of the conditions can be true.
- **Not** — None of the conditions can be true.

You must use parentheses when combining Boolean operations with comparison operations. For example, to use `i>2` and `y<3`, you must enter:

```
(i>2) and (y<3)
```

Calculations

You can insert calculations in assignment statements and conditions. It can be made up of a single item, called a literal, such as "5", "Test", "true", or "10.2".

More commonly, a calculation is an equation with reference to object attributes or to a host of built-in routines.

Comments

A particular section of code can seem obvious at first but may be confusing later. To make code easier to read and understand, it's good practice to enter comments. This is especially helpful if others view your code. To enter a comment, enter text inside brackets {}.

```
{In this code, only those over 21 can supervise.}  
  
if (Mem['Mem Age']>=21)  
    then Label1.Caption:='Can Supervise';
```

Testing, Viewing, and Printing the Code

Testing the Code

You can test the code for syntax errors and items, such as uninitialized variables. Click **Embedded Code > Test All Embedded Code**.

The following are common errors:

- Leaving off a semicolon at the end of a statement.
- Including a semicolon on the line before an **ELSE** statement.
- Setting the attribute in one case, but not resetting it in the alternate case.

Useful Information

For example, you want the color of a label to be red when a value is over \$100. You must set the label color to red, but you must also set the color to black in other cases. Otherwise, once the program finds the first value over \$100, all remaining records will be red too, even if the values are less than \$100.

- Misspelling the field name.
- Not setting up or initializing global variables.

Sometimes inserting a Boolean field inside an IF, WHILE, or REPEAT statement does not work. Instead of inserting the Boolean field inside the statement, insert the value into a string first.

Example of using Boolean inside IF-THEN

```
var
  v: string;
begin
  v:=Inv['Inv Is Reconciled'];
  if v='Yes'
    then Labell.Caption:='reconciled'
    else Labell.Caption:='';
end;
```

Viewing the Code

To view embedded code you created, click **Embedded Code > View All Embedded Code**, and select the event you want to view.

Printing the Code

You can print all embedded code in one report. This is useful for easily finding errors and effectively tracking the flow of variables. Click **Embedded Code > Print All Embedded Code**.

Data

There are many types of data. It can be a number, a string of letters for a name, or a financial amount. The following are useful types of data:

- **Integer** – A whole number between *-2,147,483,648* and *2,147,483,647*.
- **String** – A series of characters, which can include numerical digits.
- **Currency** – A decimal financial amount between *-922,337,203,685,477.5808* and *922,337,203,685,477.5807*.
- **Byte** – A single 8-bit byte, *0* to *255*.
- **Boolean** – A true or false value.

You can do different things with each type of data. For example, you can subtract 2 from an integer, but you cannot subtract 2 from a string of numbers. Some of the built-in operations are detailed in the [Calculations](#) section.

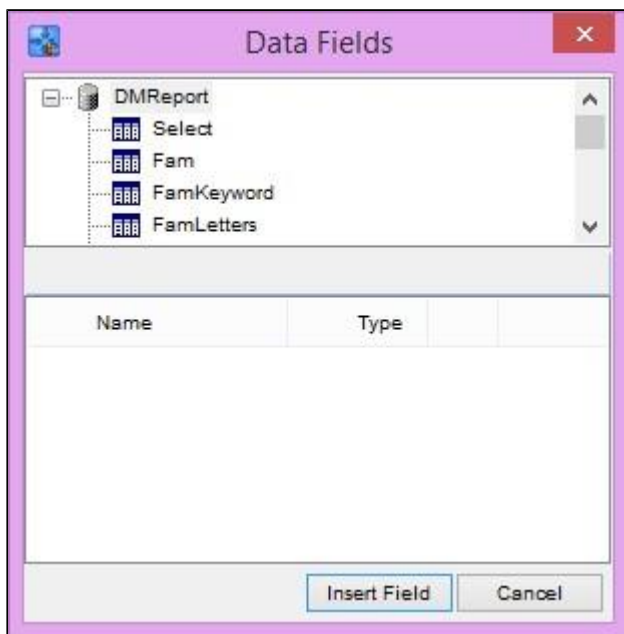
Storing Data

In reports, you can store data in three places: data fields, objects, and data variables.

Data Fields

You assign values to data fields. For example, you assign an age to a member. The data field for member age is **Mem[Mem Age]**.

This data field is from the Mem table and the field is Mem Age. In the code editor, click **Insert a Data Field** to view a list of all available data fields.



Objects

Objects are areas you insert in a report that hold information. For example, a label is an object that holds the text of what is printed for that label.

Attributes

The pieces of information that an object holds are called attributes. These can include the color, text, visibility, position, name, and other properties. Right-click an object to see all its attributes. Or, in the code editor, click **Insert an Object Attribute** to view a list of available objects and attributes.

Every object has a name, such as Label, DBText, Line, and a unique number. For example, the first label you add is named "Label1", the second is "Label2", and so on. You can reference objects in the code using its name. To change an object's name, click **View > Toolbars > Report Tree**, right-click the object, and click **Rename**. Object names can only contain letters.

Note

When you rename an object, you must also update the name in any embedded code you already entered. Otherwise, an error displays and the report won't run.

You can modify an attribute based on a condition or calculation. For example, you can change a label's text to red if the Mem Age field value is less than 5 using an IF-THEN statement.

Code Example to Change Color Attribute

```
procedure Label1OnPrint;
begin
    Label1.Caption:=Copy(Mem['Mem Name'],1,1);
    if Mem['Mem Age']<5
        then Label1.Color:=clRed;
end;
```

Events

Objects also have their own list of events that you can add code to. For example, a label has an event called GetText, and you can attach code that runs when the computer needs to know what the text of the label is.

Data Variables

Variables are temporary boxes that hold the data you define and enter in the code.

Useful Information

There is also a type of object called a variable. These are not the same thing.

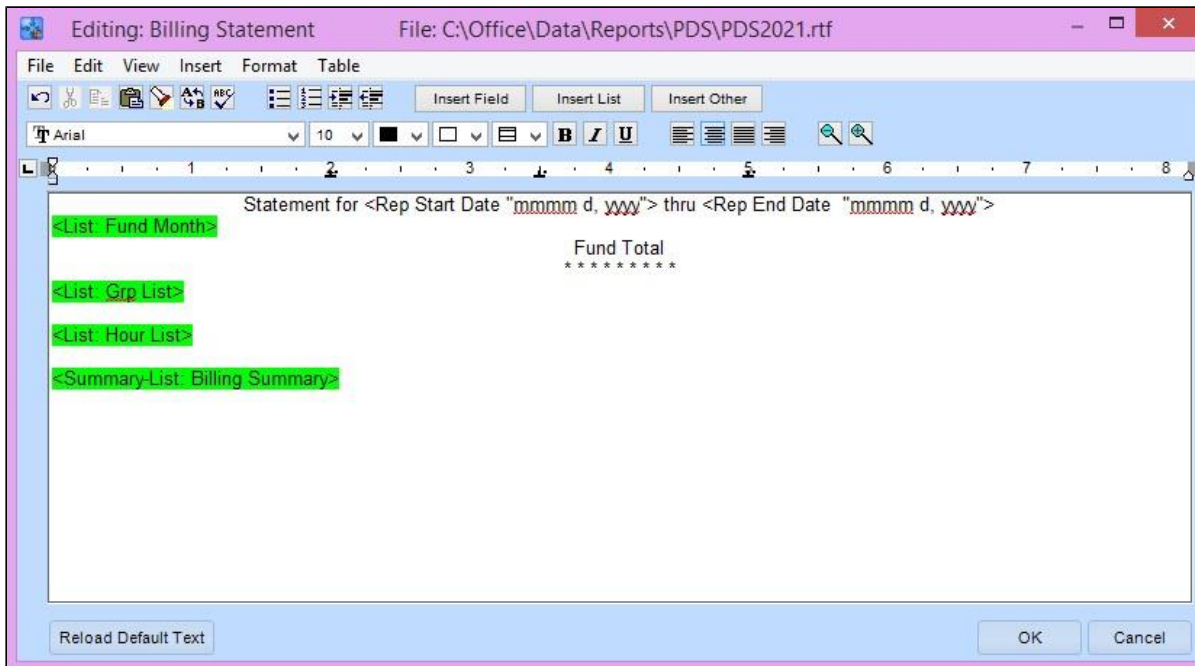
There are two types of variables that only differ in how long they exist.

- **Global variables** exist as long as the report is running. To define or initialize a global variable, click **Embedded Code > Define Global Variable** or **Initialize Global Variables**.
- **Local variables** are created in the event code and only exist as long as the event is used.

Embedding Lists in Letters

An embedded list is an advanced report inserted into a letter.

For example, the financial report "Billing Statement" has several embedded lists inside a letter.



To insert an embedded list into a letter, enter the name of the ARW report in triangle brackets <> as **<List: reportname>**. For example, <List: All Pledges> prints the total of all families for all pledges for the selected funds.

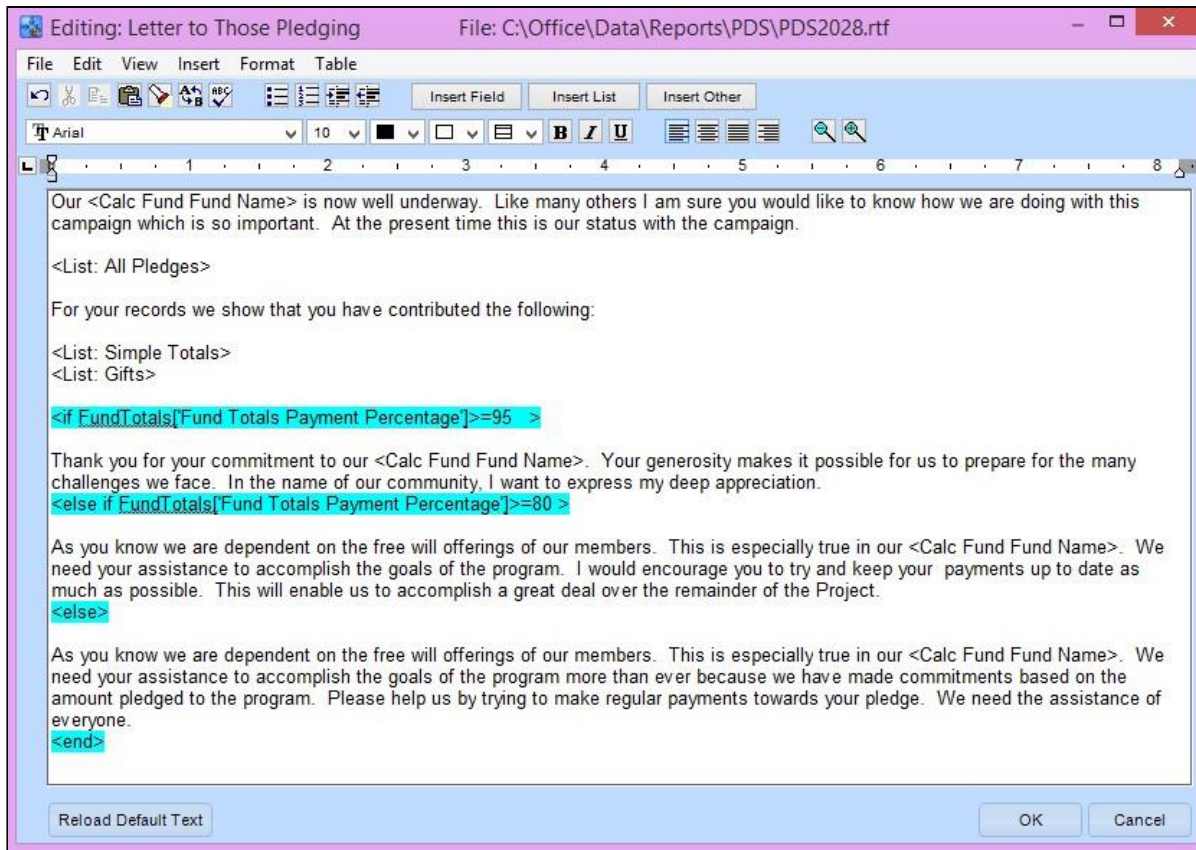
To view the underlying structure of an embedded list, click **File > Edit List Selections**, select the embedded list, and click **Open**. You can modify the list in the editor.

You can also create new embedded lists. You must save it as **List-filename.RTM** in the program's report directory, such as **C:\PDSOffice\Data\Reports\PDS**.

Creating Multiple Letters in One

You can use the IF-ELSE-END structure in a letter to create multiple letters in one, printing different paragraphs of text depending on the conditions you set up.

For example, the financial report "Letter to Those Pledging" is set up this way. Different paragraphs print depending on the family's Fund Totals Payment Percentage.



If the percentage is greater than or equal to 95, the first paragraph prints in the letter. If it's between 80 and 85 percent, the second paragraph prints. And if it's less than 80, the third paragraph prints. This way, you can print the same letter for all families instead of one letter for each percentage.

IF Statement Format

In a letter, the IF statement format is a little different than in an advanced report. The IF is surrounded by triangle brackets <>, and fields must have the format **Category['fieldname']**. The comparison symbols used in letters are the same as in advanced reports:

- > Greater than
- >= Greater than or equal to
- = Equal to
- < Less than
- <= Less than or equal to
- <> Not equal to

Using ELSE, AND, and OR

Use ELSE and ELSE-IF statements to print a different paragraph when the IF condition isn't met.

Example using ELSE-IF

```
<if Fam['Fam Number of Children']>0>
    You are invited to the Children's Day celebration.
<else if (Fam['Fam Number of Children']=0) and (Fam['Fam Number of Members']>0)>
    You are invited to the Adult celebration.
<else if Fam['Fam Number of Members']=1>
    You are invited to the Singles Dance.
<end>
```

Make sure you account for all conditions. Otherwise, a blank line prints for some families.

Example using ELSE to cover unmet conditions

```
<if Fam['Fam Number of Children']>1>
    You are invited to the Children's Day celebration.
<else if Fam['Fam Number of Children']=1>
    You are invited to the Families with Only Children celebration.
{If you stop there, a blank line will print for families with no children.}
{You need one final ELSE statement to cover all other cases:}
<else>
    You are invited to the Non-Parents celebration.
<end>
```

Use AND and OR to test multiple conditions in an IF statement. Enclose conditions in parentheses.

Example using AND

```
{Both conditions must be true for the text to print.}
<if (Fam['Fam Number of Children']>0) and (Fam['Fam City']='Phoenix, AZ')>
    You are invited to the Children's Day celebration in Phoenix Municipal Park.
<else if (Fam['Fam Number of Children']>0) and (Fam['Fam City']='Florence, SC')>
    You are invited to the Children's Day celebration in Florence City Park.
<end>
```

Example using OR

```
{Either or any of the conditions can be true for the text to print.}
<if (Fam['Fam Suffix']='Jr.') or (Fam['Fam Suffix']='Sr.')>
    You are invited to the Association of Suffix Holders (ASH) celebration.
<else if (Fam['Fam Suffix']='II') or (Fam['Fam Suffix']='III')>
    You are invited to the Association of Seconds and Thirds (AST) celebration.
<end>
```


Printing Letters in Multiple Languages

You can print the same letter for each family in their selected language. Use LANGUAGE statements to enter different versions of a letter in the same report.



Useful Information

The program doesn't translate text into different languages, but there are many free websites that translate for you.

Requirements for Using Multiple Languages

The following are requirements to keep in mind if you want to use the Multiple Languages feature:

- In the Members window, you must select the language for the head of each family you intend to use in the letter.
- If you have members who have a main language and secondary language (for example, English/Spanish, Spanish/English), list the language you want to use in letters first.
- You must spell the language exactly the same in each combination.

Modifying Reports

You can modify the following parts of the report to print in different languages.

Inside Address Style

In the Letter Layout window, select the Inside Address Style you want to adjust, then click **Edit Style**. Click **Edit Text**, and enter the text for each language you want to use. For example:

```
<Language: Spanish(Male)>
  <if pos('Sr. é Sra.',Fam['Fam Formal Sal'])>0>
    Estimados <Fam Formal Sal>,
  <else if pos('S/S',Fam['Fam Formal Sal'])>0>
    Estimados <Fam Formal Sal>
  <else>
    Estimado <Fam Formal Sal>
<end>

<Language: Spanish(Female)>
  Estimada<Fam Formal Sal>,
<end>

<Language: English>
  Dear <Fam Formal Sal>,
<end>
```

The example code tells the program to do several things:

- Look for the first male family member with "Spanish" selected in the Language field.
- Print the word "Estimados" if the salutation contains Sr. é Sra.
Print the plural "Estimados" if the salutation contains S/S.
- Print the singular "Estimado" if the salutation doesn't contain S/S.
- Print the feminine "Estimada" for females.
- Print "Dear" if the member has "English" selected in the Language field.

You can include as many languages as you want. Insert a language field, such as **<Language: French>**, **<Language: Vietnamese>**, and so on. Then enter the text of the inside address in that language.

Text of the Letter

In the Letter Layout window, click **Modify the Body of the Letter**. The example below prints text in Spanish and English.

```
<Language: Spanish>
    Este es un ejemplo de carta escrita en español.
<end>

<Language: English>
    This is a sample letter written in English.
<end>
```

You can include as many languages as you want. Insert a language field, such as **<Language: French>**, **<Language: Vietnamese>**, and so on. Then enter the text of the letter in that language.

Closing Style

In the Letter Layout window, click **Edit Style** under Closing Style, and click **Edit Text**. The example below prints the closing in Spanish and English.

```
<Language: Spanish>
    Sinceramente,
    Sr. Robert Jones
    Administrador de Oficinas
<end>

<Language: English>
    Sincerely,
    Mr. Robert Jones
    Office Administrator
<end>
```

You can include as many languages as you want. Insert a language field, such as **<Language: French>**, **<Language: Vietnamese>**, and so on. Then enter the text of the closing in that language.

Scripting Language

With embedded code, you can control the internal elements of a report. But with scripting, you can also control outside elements, like:

- Ask users about what's going to be in the report, so the report is customized. This eliminates the need to have multiple similar reports.
- Create an export or import of data. In this case, you might not generate any sort of printout.
- Sort or manipulate the data in a way that's not designed into the system.

When you run the report, the script is activated. PDS uses the Pascal scripting language, which is what you used when you embedded code. However, there are some differences.

Accessing the Script

In the Select Report window, select the advanced report you want to modify, and click **Adv. Script**. Click the **Pascal Script File** tab. In this text box, you can create or edit the script code. The following are differences between writing script and embedded code:

- All the code of a script is in one location. There isn't a separate window for each routine.
- You must enter all of the code, including the procedure declarations.
- Field names are different.
- Errors may not display until you run the report. Click **Check Code** to find simple syntax errors.

Scripting Objects

In reports, objects are items you insert into the ARW editor. In scripts, objects are more complicated. Everything on your screen and everything in the program tables are objects.

For example, the family table is an object that's stored in another object called a "datamodule" (DM). To go to the first family, enter the following line in the script:

```
PDSMDData.FamTbl.First; // Tells program to access the PDSMDData datamodule,  
                        // find the FamTbl object, then run "First" routine.
```

Scripting Events

In the script, you can attach code to events that happen as the program runs. Include a procedure or function in the script with the name of the event you want to handle.



Tip

To automatically insert the header, being line, and end line, click **Add Event** and select the procedure or function you want.

Procedures and Functions You Can Use

These are listed in the order the program executes them.

Procedure or Function	When the Program Executes It
procedure AfterCreateReport;	Called after you select the report.
procedure AfterLoadReport;	Called after you load the report.
procedure CreateExternalData;	Called after AfterLoadReport. This gives you a chance to set up tables for the report. This isn't where you add records to the tables since you haven't attached selections yet, but you can create the tables.
function BeforeReport: boolean;	Called after you click Preview. This way, you can detect any problems with selected values and cancel the report. To cancel, enter "Result:=false;".
procedure BeforeSelectReport;	Called before you attach simple filters.
procedure AfterSelectReport;	Called after you attach simple filters.
procedure BeforeSelectionsBuilt;	Called before additional selections are built. You can modify the SQL in the SelectTbl here to add any filters that are needed.
procedure AfterSelectionsBuilt;	Called after you add additional selections. Several things are done in this event since all tables have attached selections.
function PrintDefaultReport: boolean;	Called before you print. Use this with the BeforeSetupPrint event to tell the program that this isn't a normal report. If you're running an export or import (where no report is printed), add this event then enter "Result:=false;". Add BeforeSetupPrint with "Result:=false;" so the preview window doesn't display.
function BeforeSetupPrint: boolean;	See above description.
procedure BeforePrintReport;	Called before you print.
procedure AfterPrintReport;	Called after you print the report.
procedure AfterReport;	Called after the report, even if it wasn't designed to print anything (like an export report).
procedure FreeExternalData;	Called at the end of the report. Free any data tables that you created in the script (usually in the CreateExternalData event).
procedure BeforeDestroyReport;	Called at the very end.

Two Special Events You Can Use in PDS Church, Formation, and School Office

Procedure	When the Program Executes It
-----------	------------------------------

procedure OrderMailTbl;	Called after you add an entry to the Mail table. The Mail table is used for special sorts where you include billing addresses and courtesy copies. To speed up the program, this procedure isn't called unless there's a line in the UDR file that reads "OrderMailTbl=1". You also need to set "ForceMailTbl=1". Inside the OrderMailTbl event, set PDSMDData.MailTblKey.AsString to the value you want to sort by.
procedure CallAfterCalc;	Called after the Family table finishes its calculations for each family. To speed up the program, this procedure isn't called unless there's a line in the script that reads "UseCallAfterCalc:=true;". This is usually in the AfterCreateReport event.

Script Code Statements

In the [Code](#) section, there's information about Pascal statements. In addition to the statements discussed in that section, scripting language has a few more statement types.

With Statements

Use a With statement to set several object attributes.

Label1 Attributes
<pre>// Note that the label object has several attributes Label1.Caption:='Test'; Label1.Top:=100; Label1.Left:=200; Label1.Width:=100;</pre>

Instead of listing each attribute, use a With statement:

With Statement
<pre>With Label1 do begin Caption:='Test'; Top:=100; Left:=200; Width:=100; end;</pre>

If you have several objects, you can nest With statements, or you can separate objects with commas. Also use a With statement for an attribute that has its own attributes, such as font.

Try-Finally and Try-Except

Use these to handle error conditions.

The Try-Finally statement tells the program to "Try" to do something, and no matter what happens, perform what is in the "Finally" section.

Try-Finally Statement

```
// This code will "Try" to create a file named A:\test.txt, then write a few lines
// of text to it. No matter what happens (for example, if the disk is full),
// the program will "Finally" run the CloseFile statement.
var
    FOut: Text;
begin
    Assign(FOut, 'A:\Test.txt');
    Try
        Rewrite(FOut);
        Write(FOut, 'Test');
        Writeln(FOut, ' The end of a line');
    Finally
        CloseFile(FOut);
    end;
end;
```

The Try-Except is similar, but the "Except" statement only runs if an error occurs.

Try-Except Statement

```
// If you "Try" to divide two numbers, you must have an "Except"
// in case the denominator is zero.
var
    i, j: integer;
begin
    i:=5;
    j:=0;
    Try
        i:=i div j;
    Except
        // If there was no "Except" statement, an unexpected error would display.
        ShowMessage('Cannot do '+
            IntToStr(i)+'/'+IntToStr(j));
    end;
end;
```

Try-Finally and Try-Except are rarely required, but they make reports better able to tolerate problems without giving the user unexpected error messages.

Exit, Break, Continue, and Goto

These statements control the flow of the program when a situation arises.

"Exit" tells the program to close the current procedure or event and return to whatever called it.

Use the Break and Continue statements inside of loops. "Break" tells the program to jump out of the loop.

"Continue" tells the program to go to the bottom of the loop.

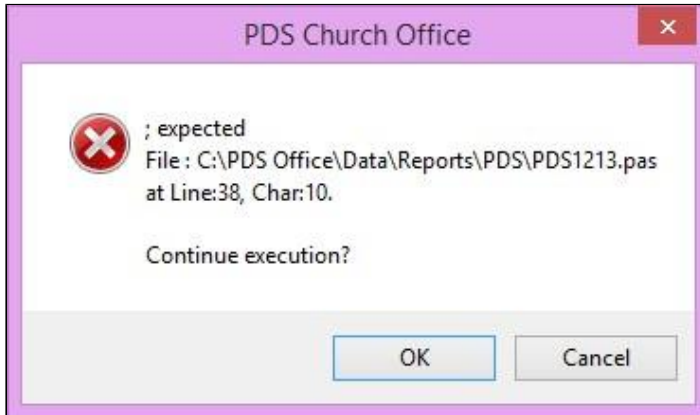
Use the Goto statement to put a label in, then go to that label.

Example of Exit, Break, Continue, and Goto

```
// You rarely use all four statements together,
// so this example doesn't perform an actual function.
Procedure Test;
Label TryBackwards;      // define a Label for the Goto
var
  i, j: integer;
begin
  if PDSMDData.FamTbl.RecordCount=0
    then exit;           // get out of this routine completely
  j:=0;
  for i:=1 to 5 do
    begin
      PDSMDData.FamTbl.First;
      if PDSMDData.MemTbl.RecordCount=0
        then continue;  // get the next i
      while not PDSMDData.FamTbl.Eof do
        begin
          if PDSMDData.MemTbl.RecordCount=1
            then break;  // get out of the while loop
          if PDSMDData.MemTbl.RecordCount=1
            then goto TryBackwards; // jump to the label
          j:=j+1;
          PDSMDData.FamTbl.Next;
        end;
      TryBackwards:
        PDSMDData.FamTbl.Last;
        while note PDSMDData.FamTbl.Bof do
          begin
            j:=j+1;
            PDSMDData.FamTbl.Prior;
          end;
    end;
end;
```

Detecting Errors

In the Advanced Script dialog box, on the Pascal Script File tab, there is a **Check Code** button. This checks for simple syntax errors, but it can't detect complex errors. When you run the report, if there's an error, a dialog box displays with as much information as possible.



This tells you what to fix in the code. In the example above, the program tells you it expected to read a semicolon (;) on line 38, character 10.

Common Problems

The following are a few things you need to be aware of in scripting:

- **The program can crash more when you write scripts.** There are less protections, so the report is more likely to do something that freezes the program. PDS recommends using newer operating systems.
- **It's easy to make a mistake when you enter an event name.** The program doesn't tell you when an event name is misspelled or incorrect. If you have this problem, put a **ShowMessage** command in the procedure when you first add the event to make sure everything you think is running is actually running.
- **If you don't initialize your variables, you will have problems when writing scripts.**

Accessing and Adding Screens

You can use scripts to [ask users questions](#), then you can [use the answers](#) to customize a report. In this section, you will learn about script routines and objects.

You can learn a great deal from the program's predefined reports. In Member Involvement reports, you can see how to make layout screens.

Simple Messages and Questions

In the [Scripting Events](#) section, you learned how to notify the user with a ShowMessage statement. An easy way to get information from a user is to prompt the user with a dialog box that has buttons they can click, such as "Yes", "No", "Cancel", "OK", and so on. To display a dialog box with button options, use the built-in function,

PDSMessageDlg:

```
PDSMessageDlg(S, MessageType, ButtonSet, HelpContext);
```

- **S** is the message you want to display.
- **MessageType** controls the dialog box caption and the icon that displays.
 - mtWarning contains a yellow exclamation point.
 - mtError contains a red stop sign.
 - mtInformation contains a blue "i".
 - mtConfirmation contains a green question mark.
 - mtCustom has no icon. The message box caption is the application's executable file name.
- **ButtonSet** lists buttons that display in the dialog box. These are in the **MkSet** function call.
 - mbYes shows "Yes".
 - mbNo shows "No".
 - mbOK shows "OK".
 - mbCancel shows "Cancel".
 - mbAbort shows "Abort".
 - mbRetry shows "Retry".
 - mbIgnore shows "Ignore".
 - mbAll shows "All".
 - mbNoToAll shows "No to All".
 - mbYesToAll shows "Yes to All".
 - mbHelp shows "Help".
- **HelpContext** is the function that returns which button the user clicks.
 - mrNone – The user clicked the "X" in the top right corner of the dialog box.
 - mrYes – User clicked "Yes".
 - mrNo – User clicked "No".
 - mrOK – User clicked "OK".And so on for the rest of the buttons.

Useful Information

For the purposes of this guide, always enter **0** for the HelpContext function.

For example, to prompt the user with a message that asks if they're sure, enter the following:

```
if PDSMessageDlg('Are you sure?', mtConfirmation, MkSet(mbYes,mbNo), 0)=mrYes
    then ShowMessage('You said Yes')
    else ShowMessage('You did not say Yes');
end;
```

If you need to have more than one button, you can save the result in an integer. For example:

```
var
    i: integer;

begin
    i:=PDSMessageDlg('What should we do?', mtError, MkSet(mbRetry, mrIgnore), 0);
    if i=mrRetry
        then ShowMessage('You clicked Retry')
    else if i=mrIgnore
        then ShowMessage('You clicked Ignore')
    else ShowMessage('You clicked the X');
end;
```

If there's more than one question or if the response requires the user to enter text, you can [create a new window](#) in the report wizard. This is referred to as the second layout window.

Screen Routines in the ScrnIO

To simplify scripting, the program comes with a library of common routines. The Screen Input/Output library (ScrnIO.pas) is located in the Reports folder of the program. At the top of the script, enter the following:

```
Uses  
    scrnio;
```

Useful Information

The examples below use the variable names "S", "L", "E", and "N". You don't have to use these variable names. You can use the names of the variables you have declared or literals of the right type. You also don't have to put in the variable types, such as "string" or "TCheckbox". They are only used here as examples.

ScrnIO adds the following routines to the script:

NewScreenPage; — Adds a new window to the report wizard and sets everything up to use it. This routine initializes all internal variables used to position fields.

AddHeading(S: string; L: TLabel); — Adds an "S" heading label. The actual label object is returned to you in the variable you declare as "L". For example:

```
AddHeading('Talent/Ministry Options:', My Label);
```

The code above creates the following:

Talent / Ministry Options:

AddHeadingCheckBox(S: string; E: TCheckBox); — Adds a heading label with a checkbox.

AddLabel(S: string; L: TLabel); — Adds a simple label to the window.


Talent List

AddEdit(S: string; L: TLabel; E: TEdit; N: integer); — Adds a label and edit box. "N" is equal to the maximum number of characters entered.

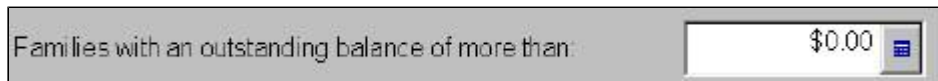
Starting Month:

AddPhone(S: string; L: TLabel; E: TDBPhone; N: integer); — Adds a label and edit box formatted for a phone number.

AddDate(S: string; L: TLabel; E: TDBDateBtn; N: integer); — Adds a label and edit box that is formatted for a date and has a calendar button.

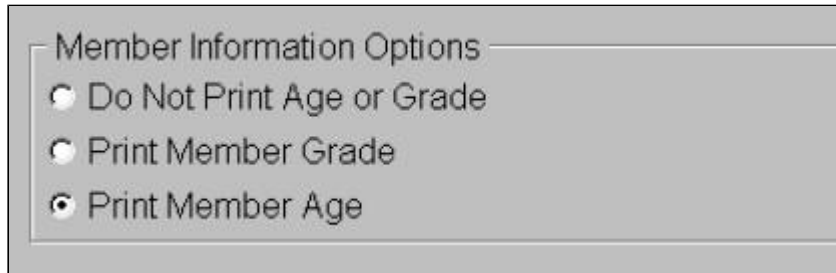
Date for the Alternate Address: 

AddAmt(S: string; L: TLabel; E: TDBAmount; N: integer); — Adds a label and edit box that is formatted for an amount and has a calculator button.

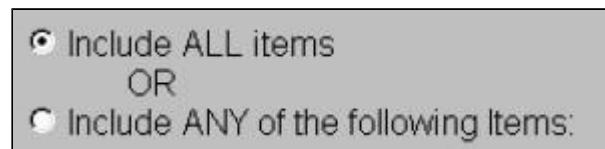


AddTotal(S: string; L: TLabel; E: TDBAmount; N: integer); — Adds a label and edit box formatted for an amount. It's similar to AddAmt, but it doesn't have a calculator button, and the field is unavailable so a user cannot enter an amount. Use this to display an amount that is the sum of other boxes in the window.

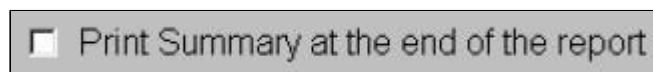
AddRadioButtonGroup(S: string; List: string; E: TRadioGroup); — Adds a radio button group. "S" is the group caption and "List" is the list of radio buttons. Separate each list item with a comma, for example, 'Red, Yellow, Blue'. The object is returned in the last item. You can determine which radio button the user selects in the E.ItemIndex, where "E" is the name of the TRadioGroup object you declared and used as the third parameter.



AddIncAll(S1,S2: string; IncOr: boolean; E1,E2: TRadioButton); — Adds a pair of radio buttons that indicate to include ALL or include ANY of the items in a table. You can turn off the "OR" that displays between the radio buttons with **IncOr**.



AddCheckBox(S: string; E: TCheckBox); — Adds a checkbox with a caption.



AddRadioButton(S: string; E: TRadioButton); — Adds a radio button with a caption.



AddDropDown(S: string; L: TLabel; E: TComboBox; N: integer); — Adds a drop-down list with a caption. "N" is equal to the maximum number of characters.



AddLine(N: integer); — Adds a line of "equals signs" as a separator. "N" is the number of equals signs.

AddBlank; — Adds a blank line below the last object placed on the window.

AddPage(Pg: LMDAssistPage; N: integer); — Adds a new page to the report wizard. It's similar to NewScreenPage, but it returns the new page object in the variable you enter as "Pg". "N" is the position of the new page and is usually set to "3". Use this if you need access to the page object.

Screen Objects

The routines listed in the [Screen Routines in the ScrnIO](#) section create objects, then add them to the screen. Before you can use any of the routines, you must declare the objects you're going to have on the screen. You declare the object you want to use in a "var" statement at the top of the script. The following are common object types:

- **TLMDAssistPage** – Page in the wizard.
- **TLabel** – Label on the screen.
- **TEdit** – String edit field.
- **TComboBox** – String edit field with a drop-down list.
- **TDBDateBtn**– Date edit field with a calendar button. You can attach this to a database field.
- **TDBAmount**– Amount edit field with a calculator button. You can attach this to a database field.
- **TDBPhone** – Phone edit field. You can attach this to a database field.
- **TCheckBox** – Single checkbox with a caption.
- **TRadioButton** – Single radio button with a caption.
- **TRadioGroup** – Set of radio buttons with a caption.
- **TButton** – Simple button.
- **TPDSDBGrid** – Grid. You can attach this to a database field.

Other Objects

You can place other objects on a screen, such as buttons and grids. You can find examples of these objects in the Member Involvement reports.

Like embedded code, objects in the script can have events, too. For example, if you add a button to a screen, you can also add an event that executes when a user clicks the button.

Code to enter in the script to create a button

```
procedure MyBtnOnClick(Sender: Tobject);
begin
    ShowMessage('Button clicked');
end;

procedure AfterReportLoad;
var
    MyBtn: TButton;
begin
    NewScreenPage;
    MyBtn:=TButton.Create(CurForm);
    MyBtn.Caption:='My Button';
    MyBtn.Top:=10;
    MyBtn.Left:=10;
    MyBtn.Width:=30;
    MyBtn.Height:=10;
    MyBtn.Parent:=CurPage;
    AttachEvent(MyBtn, 'OnClick', 'MyBtnOnClick', false);
    {AttachEvent connects an event to the procedure you want to run for an object.}
end;
```

When you create an object that isn't in the Screen Input/Output library (ScrnIO), you must set its attributes. The main attribute you must set is the "Parent", which tells the program where the object displays. Usually, this is the new page that you created, called "CurPage".

Communicating with the Report

After you create a new window in the report wizard where users can enter information, you must have a way to relay that information to the report. Add the following two routines to the script:

```
procedure PDSSetParam(i: integer; s: string);  
function PDSGetParam(i: integer): string;
```

These routines send information to the report, and the report can even send information back to the script. The integer "i" is equal to the number of parameters, which you can set from 0 to 32000. Each value of "i" is a different parameter, and each parameter is a string. These routines are available in the script and in the embedded code. For example, write the following in the script:

```
begin  
    PDSSetParam(0,EditName.Text);  
end;
```

"EditName" is an edit object that displays on the screen. In embedded code, you could have a GetText event of a label. For example:

```
begin  
    Text:=PDSGetParam(0);  
end;
```

The report prints the value entered on the screen. The program can pass any type of value between the script and the report, but it converts the value to a string before sending it. Usually, the program uses it as a string in the embedded code, but sometimes it needs it in its original form, so it converts it back in the embedded code. This is often true for dates when you want to use the date in some sort of comparison.

The most common place to put the setting of the parameter in the script is in the AfterSelectReport event, because when the program gets to that event, the user cannot change anything on the screen. If there are many parameters to pass, you can create a procedure, such as the following:

```
procedure SetParams;  
begin  
    {PUT THE ITEMS INTO PARAMETER FOR THE REPORT}  
    PDSSetParam(0,Uppercase(OwnerName));  
    if SepPageCByes.Checked  
        then PDSSetParam(1,'Yes')  
        else PDSSetParam(1,'No');  
    if PageNumCByes.Checked  
        then PDSSetParam(2,'Yes')  
        else PDSSetParam(2,'No');  
    if FonCByes.Checked  
        then PDSSetParam(4,'Yes')  
        else PDSSetParam(4,'No');  
    PDSSetParam(5,MemGroup.ItemIndex);  
    PDSSetParam(6,AgeGradeGrp.ItemIndex);  
end;  
  
procedure AfterSelectReport;  
begin  
    SetParams;  
end;
```

You must also add a call to the new SetParams routine at the bottom of the AfterLoadReport event, after the code to write to the INI file. You need this call so the preview report will work inside the Advance Report Writer editor.

After you insert the SetParams routine, you must include embedded code in the report so the report will use the parameters.

Other Objects

You can place other objects on a screen, such as buttons and grids. You can find examples of these objects in the Member Involvement reports.

Like embedded code, objects in the script can have events, too. For example, if you add a button to a screen, you can also add an event that executes when a user clicks the button. To create a button, enter the following in the script:

```
procedure MyBtnOnClick(Sender: TObject);
begin
  ShowMessage('Button clicked');
end;

procedure AfterReportLoad;
var
  MyBtn: TButton;
begin
  NewScreenPage;
  MyBtn:=TButton.Create(CurForm);
  MyBtn.Caption:='My Button';
  MyBtn.Top:=10;
  MyBtn.Left:=10;
  MyBtn.Width:=30;
  MyBtn.Height:=10;
  MyBtn.Parent:=CurPage;
  AttachEvent(MyBtn, 'OnClick', 'MyBtnOnClick', false);
end;
```

The AttachEvent procedure connects an event to the procedure you want to run for a given object.

When you create an object that is not in the Screen Input/Output library (**ScrnIO**), you must set its attributes. The main attribute you must set is the "Parent", which tells the program where the object displays. Usually, this is the new page that you created, called "CurPage".

Positioning Objects

When you create an object, you must set the top, left position, width, and height of the object as it displays on the screen. In the Screen Input/Output Library (ScrnIO), the following variables are declared. Use these to set object attributes:

- **FontH** – Height of the font.
- **LabelT** – Top of the next label object. Each time you place a ScrnIO object on the screen, the program moves this down the amount you enter for LabelN. If you use this variable, increment it by LabelN. For example, **LabelT:=LabelT+LabelN;**
- **LabelH** – Height of a standard label object.
- **LabelL** – Left position of the label object.
- **LabelN** – Increment to the next line. Usually, this equals the EditN.
- **EditT** – Top of the next edit object. Each time you place a ScrnIO object on the screen, the program moves this down the amount you enter for EditN. If you use this variable, increment it by EditN. For example, **EditT:=EditT+EditN;**
- **EditH** – Height of a standard edit object.
- **EditL** – Left position of the edit object.
- **EditW** – Width of the edit object.
- **EditN** – Increment to the next line. Usually, this equals the LabelN.

Change these values in order to control where the object is placed. For examples of this, view the Member Involvement reports.

 **Useful Information**

You can position objects without using these variables. However, if you use these variables, they adjust according to the user's screen size. If you position objects any other way, make sure you test the screen at different sizes.

Special Sorts

When performing special sorts, you must use scripts. There are several tables you can use to sort:

- A Select Table is built into the program for sorts using SQL (Structured Query Language).
- A Mail Table is used for ZIP Code sorts, email sorts, courtesy copy sorts, billing address sorts, and any family/member sort that doesn't have SQL.
- Memory Tables are database tables that reside in program memory.
- Database Tables hold more information than memory tables, and you can connect them to existing tables.

Select Table

The easiest way to perform a special sort is to set up the SelectTbl. This is a table built into the program for sorts using SQL. If it's set to active, the program knows to use it.

Example: Sort a member report by date family registered using SQL

```
procedure AfterSelectReport;
begin
    DM.SelectTbl.Sql.Text:=
        'Select '+
        ' Mem.MemRecNum, '+
        ' Fam.DateRegistered '+
        'from '+
        ' Mem, Fam '+
        'where '+
        ' Fam.FamRecNum=Mem.FacRecNum '+
        'order by '+
        ' DateRegistered';
    DM.SelectTbl.Active:=true;
end;
```

To make the code easier to read, enter the SQL statements on different lines. Also, place spaces at the end of each line so the words don't run together in your report.

In SQL, you must include a field that links the SelectTbl to the main table of the report type you're writing. For example, if you're writing a family report, you must include the FamRecNum field, and if you're writing a member report, you must include the MemRecNum field. The program uses that field to connect with all other tables. The program gets the first record of the SelectTbl, then uses the FamRecNum or MemRecNum field to find the correct family or member record. Once the program locates the record, it links it to the other tables.

You can also add conditions to the WHERE clause of the SQL, which tells the report to filter the families or members before it prints. Add conditions when you want to filter data without the user entering a selection.

Mail Table

In PDS Church, Formation, and School, you can perform a special sort with the MailTbl. This is a table created in the user's temporary folder, and it isn't based on SQL.

For each family or member in your report, a record is added to this table. It's used for ZIP Code sorts, email sorts, courtesy copy sorts, billing address sorts, and any family or member sort that doesn't have SQL. The table can have more than one entry per family.

To use the MailTbl, add the following lines to the [Default] section of the UDR File:

```
OrderMailTbl=1
ForceMailTbl=1
```

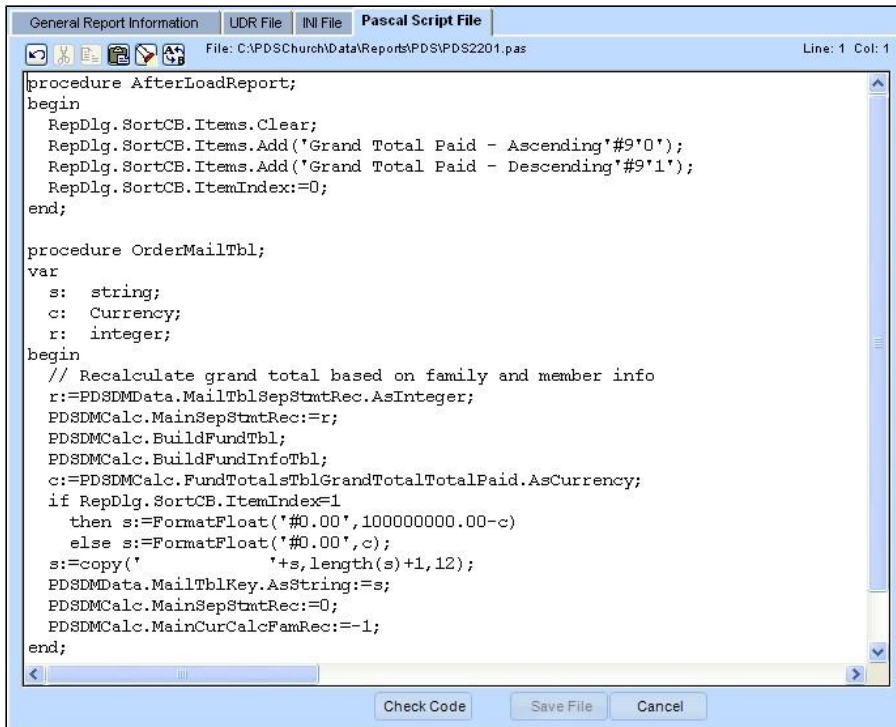
The OrderMailTbl event executes each time the program is ready to add an item to the MailTbl. Then, you can enter you own key value...

```
PDSMDData.MailTblKey.AsString
```

...and set it to a value or calculation. When this event executes, all tables are set up and ready to use, including the FamTbl and MemTbl.

Example

You can view an example of this in PDS Church Office by copying the financial report, Listing in Order by Amount, and clicking the **Pascal Script File** tab.



The screenshot shows a window titled 'Pascal Script File' with the following code:

```
procedure AfterLoadReport;
begin
  RepDlg.SortCB.Items.Clear;
  RepDlg.SortCB.Items.Add('Grand Total Paid - Ascending'#9'0');
  RepDlg.SortCB.Items.Add('Grand Total Paid - Descending'#9'1');
  RepDlg.SortCB.ItemIndex:=0;
end;

procedure OrderMailTbl;
var
  s: string;
  c: Currency;
  r: integer;
begin
  // Recalculate grand total based on family and member info
  r:=PDSMDData.MailTblSepStatRec.AsInteger;
  PDSMCalc.MainSepStatRec:=r;
  PDSMCalc.BuildFundTbl;
  PDSMCalc.BuildFundInfoTbl;
  c:=PDSMCalc.FundTotalsTblGrandTotalTotalPaid.AsCurrency;
  if RepDlg.SortCB.ItemIndex=1
  then s:=FormatFloat('#0.00',100000000.00-c)
  else s:=FormatFloat('#0.00',c);
  s:=copy('          '+s,length(s)+1,12);
  PDSMDData.MailTblKey.AsString:=s;
  PDSMCalc.MainSepStatRec:=0;
  PDSMCalc.MainCurCalcFamRec:=-1;
end;
```

In this report, the script resets the Sort Order drop-down list so it includes only two items. The UDR File tab displays:

```
[Default]
CanUseSepStmt=1
ForceMailTbl=1
OrderMailTbl=1
PageStyle=360
```

The code above tells the program to use separate statements, build and use the mail table to sort, and call for script routines for each entry in the MailTbl. It also sets the default page style to Compressed Format.

Memory Tables

You may need to store data temporarily in a report. You could store it in a variable, but sometimes the data is too large. In that case, use a memory table, which is a database table that resides in program memory. It isn't stored on a disk, but you can save and load memory tables if needed.

For example, in a financial statement, you include a note that has the family's current rate and terms, as well as the date of the last donation. You could make an ARW report that does this, but it will calculate the values in a sub-report or an embedded list. This makes it difficult to have the values display where you want them.

You can extend the report to have new fields, which is where the memory table comes in. Access a memory table the same as accessing other data fields. You can even use the tables in selections.

Declaring, Initializing, and Assigning Variables

Before using memory tables, you must declare, initialize, and assign the variables you need. To implement a table, you need three variables: the table, the datasource, and the pipeline.

Code to declare variables

```
var
  {Table tells the program what the fields are and how database is ordered and indexed}
  NewFieldTbl: TkbmMemTable;
  {Datasource is a special connection that connects database to screen fields}
  NewFieldDS: TDataSource;
  {Pipeline is the connection between datasource and report builder}
  NewFieldPL: TppDBPipeline;
```

You must initialize the variables in the AfterCreateReport procedure. Otherwise, the report stops.

Code to initialize variables

```
procedure AfterCreateReport;
begin
  {Set variables to "nil", which means they have nothing in them}
  NewFieldTbl:= nil;
  NewFieldDS:= nil;
  NewFieldPL:= nil;
end;
```

After initializing, you must assign values.

Code to assign variables

```
procedure CreateExternalData;
begin
  {Create a new memory table}
  NewFieldTbl:= TKbmMemTable.Create(DMReport);
  with NewFieldTbl do
  begin
    {Set up the memory table}
```

```

        FieldDefs.Clear; {We don't need this line, but it doesn't hurt}

        {Set up the field to sort by}
        FieldDefs.Add('Order', ftInteger, 0, False);

        {Set up the new fields to calculate}
        FieldDefs.Add('TermsAndRate', ftString, 40, False);
FieldDefs.Add('Terms', ftString, 40, False);
FieldDefs.Add('Rate', ftCurrency, 0, False);
FieldDefs.Add('LastDate', ftDate, 0, False);

        {Set up the index (not really used for this table)}
        IndexDefs.Add('', 'Order', MKSet(ixPrimary));
        NewFieldTbl.IndexFieldNames:= 'Order';
        CreateTable;
        Active:= true;
end;

{Create the data source}
NewFieldDS:= TDataSource.Create(DMReport);
NewFieldDS.DataSet:= NewFieldTbl;

{Create the pipeline}
NewFieldPL:= TppDBPipeline.Create(DMReport);
NewFieldPL.Name:= 'NewFieldPL';
NewFieldPL.UserName:= 'NewField';           {this is the name users see}
NewFieldPL.DataSource:= NewFieldDS;
NewFieldPL.AutoCreateFields:= true;

{Add in the first and only record}
NewFieldTbl.Append;
NewFieldTbl.FieldByName('Order').AsInteger:=0;
NewFieldTbl.FieldByName('TermsAndRate').AsString:= '';
NewFieldTbl.FieldByName('Terms').AsString:= '';
NewFieldTbl.FieldByName('Rate').AsCurrency:= 0;
NewFieldTbl.Post;
        {Tell program to call CallAfterCalc after moving from one family to the next}
        UseCallAfterCalc:= true;
end;

```

After running the report, you need to get rid of the table using the FreeExternalData event.

Code to free external data

```

procedure FreeExternalData;
begin
    {Only removes items that aren't equal to "nil"}
    if assigned(NewFieldTbl)
        then NewFieldTbl.Free;
    if assigned(NewFieldDS)
        then NewFieldDS.Free;
    if assigned(NewFieldPL)
        then NewFieldPL.Free;
end;

```

Attaching and Building the Tables

After adding a new table, you must make it accessible to reports and selections.

Code to add tables to program

```
{Add tables to the program's list of tables}
procedure AttachNewTables;
begin
    AddTable(NewFieldTbl, NewFieldDS, 'NewField', 'Order', 'Fam', 'FamRecNum', '');
    {The first parameter (NewFieldTbl) is the table you want to add}
    {Second (NewFieldDS) is the table's datasource}
    {Third ('NewField') is the name that displays to users}
    {Fourth ('Order') is the indexed field you want to use}
    {Fifth ('Fam') is the table that this table is linked to}
    {Sixth ('FamRecNum') is field in table that matches third parameter's value}
    {The seventh parameter ('') is always blank}
end;
```

Now define what should happen after a family or member record is added to the table.

Code to build the table

```
{Calculate the value of the new fields}
procedure CallAfterCalc;          {Called every time we get a new family}
var
    s, s1: string;
    i: integer;
    d, d1: date;
begin
    {Loop through the CalcFund and CalcFundHist tables to find the latest payment}
    {And loop through CalcFundAct to find the first activity with recurring charge}
    {Then store it in the NewField table}
    s:= '';
    d:= 0;

    with PDSDMCalc.CalcFundTbl do
    begin
        First;
        while not eof do
        begin
            if PDSDMCalc.CalcFundHistTblActFund.AsInteger<9 then
            begin
                d1:= PDSDMCalc.CalcFundHistTblFEDate.AsDateTime;
                if d1>d
                    then d:= d1;
            end;
        Next;
        end;
    end;
    {Get the last recurring description}
    if s='' then
    with PDSDMCalc.CalcFundActTbl do
    begin
        First;
        while (s='') and not eof do
        begin
            s1:=trim(PDSDMCalc.CalcFundActTblRecurDesc.AsString);
            if s1<>''
                then s:=s1;
        Next;
    end;
end;
```

```

                end;
            Next;
        end;
First; {Move back to the first record in case anything depends on that}
end;

{There's only a single record, so edit it and put in the new values}
NewFieldTbl.Edit;
NewFieldTbl.FieldByName('Order').AsInteger:= PDSMDData.FamTblFamRecNum.AsInteger;
NewFieldTbl.FieldByName('TermsAndRate').AsString:= s;
i:= pos(',',s);
if i<>0 then
    begin
        NewFieldTbl.FieldByName('Terms').AsString:= copy(s,i+1,40);
        s:= CleanAmount(copy(s,1,i));
        NewFieldTbl.FieldByName('Rate').AsCurrency:= StrToCurr(s);
    end;
else
    begin
        NewFieldTbl.FieldByName('Terms').AsString:= s;
        NewFieldTbl.FieldByName('Rate').AsCurrency:= 0;
    end;
if d=0
    then d:=now;
NewFieldTbl.FieldByName('LastDate').AsDateTime:= d;
NewFieldTbl.Post;

end;

```

In PDS Church Office, there are built-in optimizations that speed up reports. After a family is read, it is not recalculated. However, there may be times you want to recalculate it after it's connected.

Code to recalculate records

```

{You can trick the program into thinking it hasn't seen the first record yet
procedure PrintDefaultReport;
begin
    {Force the program to recalculate just before report starts to print}
    {If we don't do that, the first record is wrong}
    PDSDMCalc.MainCurCalcFamRec:= 0;
    PDSMDData.FamTbl.First;
end;

```

Database Tables

A database table holds more information than a memory table, and you can connect it to an existing table. For example, in the Member Involvement reports, there are tables that retrieve the ministries and ministry statuses. You can use these tables in a set of grids to select ministries to print. In this case, you don't need to have a pipeline because the table is just for use in the layout window, not in the report.

In the example below, the code states the following:

- The database table is connected to the MinType.db file. The table is connected to the same database that the family table is connected to, and it's indexed by its description.
- There are three field objects (FieldName). One is attached to the Description; one is attached to the MinDescRec, which is a unique integer identifier for each record in the description; and one is a calculated Boolean field called "Selected".

Useful Information

Instead of making three separate variables, MinSelFld was used three times because you don't need use the field variables themselves.

```
{Build Table}
MinTbl:=TwwTable.Create(CurForm);
MinTbl.Active:=false;
MinTbl.TableType:=ttParadox;
MinTbl.DatabaseName:=PDSMDData.FamTbl.DatabaseName;
MinTbl.TableName='MinType.db';
MinTbl.IndexFieldNames='Description';
MinTbl.FieldDefs.Update;
MinSelFld:=TBooleanField.Create(CurForm);
MinSelFld.FieldName='Selected';
MinSelFld.Calculated:=true;
MinSelFld.Dataset:=MinTbl;
MinSelFld:=TStringField.Create(CurForm);
MinSelFld.FieldName='Description';
MinSelFld.FieldKind:=fkData;
MinSelFld.Dataset:=MinTbl;
MinSelFld:=TIntegerField.Create(CurForm);
MinSelFld.FieldName='MinDescRec';
MinSelFld.FieldKind:=fkData;
MinSelFld.Dataset:=MinTbl;
MinDS:=TwwDataSource.Create(CurForm);
MinDS.DataSet:=MinTbl;
MinDS.Active:=true;
```

Exporting Fields Using Scripting

In reports, you can export fields. To export information in a precise way and create complex formations, use scripting.

Exporting Events

When you create an export report, there are three additional events:

- **BeforeBuildExport: boolean** – Called before the program runs the export. Use this with the PrintDefaultReport event to tell the program this is not a normal export. You will write to the file, so none of the built-in code for writing is needed.
- **AfterBuildExport** – Called after the program runs the internal export. If you are handling the export, then it's called right after the BeforeBuildExport event.
- **FinishBuildingExport: boolean** – Usually, after the export, a dialog displays notifying you that the export is complete. If you don't want a default dialog box to display, set this event to false.

Simple Export

Using the three additional events, you can create a simple export.

1. In family reports, click **Add > Export**.
2. In the List of Fields to Print, click **Save/OK** to exit. You can come back to add fields.
3. In the Overview window, click **Back**.
4. In the Select Report window, click **Adv. Script**, then click the **Pascal Script File** tab.
5. Enter the following code:

```
var
    f: TextFile;

function PrintDefaultReport: boolean;
begin
    result:=false; {We will create the file in AfterBuildExport}
end;

function BeforeBuildExport: boolean;
begin
    result:=false; {We will create the file in AfterBuildExport}
end;

procedure AfterBuildExport;
begin
    {Begin the export}
    AssignFile(f,RepBlg.ExpFileName.Text);
    Rewrite(f);
    try
        with PDSMDData.FamTbl do
            begin
                First;
                while not EOF do
                    begin
                        write(f,PDSMDData.FamTbl['NameFormat1']);
                        writeln(f,'');
                        Next;
                    end;
            end;
        finally
            CloseFile(f);
        end;
    end;
end;
```

Useful Information

This code does the following:

- Creates a TextFile.
- Assigns the name that you have on the Export Properties window, then creates the file. If the file already exists, it overwrites it.
- Inserts family names into the file, each family beginning on a new line.
- Closes the file. In the script, you could have included any fields or performed any calculation to obtain the information to export.

6. Click **Save File**.

Improving the Export

In the simple event, dialog boxes display as the program builds the file so you know where you are in the process. If you cancel, it completes the process before canceling it. To avoid this, add the following code before the Next statement.

Code to avoid completing the process before canceling

```
{... code from procedure AfterBuildExport...}
begin
    write(f, PDSMDData.FamTbl['NameFormat1']);
    writeln(f, '');
    {Add this code before the Next statement;}
    ExportLabel.Caption:='Family: '+ PDSMDData.FamTblName.AsString;
    {This sets the caption of the dialog box's export label}
    Application.ProcessMessages;
    if ExportLabel.Caption:='Stopping...'
    {The program watches this label until it says "Stopping"}
        then exit;
    Next;
end;
end;
finally
    CloseFile(f);
end;
end;
```

You may want to hide a few items in the Export Properties window.

Code to hide button and drop-down lists

```
{Add this code after the AfterBuildExport procedure}

procedure AfterLoadReport;
begin
    {Make the report dialog look better}
    RepDlg.Label33.Caption:='My Export: ';
    RepDlg.GroupBox15.Visible:= false; {hide Modify List of Fields to Export button}
    RepDlg.Label35.Visible:= false; {hide label}
    RepDlg.ExpTypeCB.Visible:= false; {hide drop-down list}
end;
```

Formatting Your Export

To position fields in the file, use a fixed-width format. You must create helper routines that output your fields in the fixed format then pad them to the proper width.

Code to output information in fixed-width format

```
{Helper routine that outputs strings with a fixed width of N bytes}

procedure ExportStr(s: string; n: integer);
var
  i: integer;
begin
  s:= copy(s,1,n);
  while length(s)<n do
    s:=s+' ';
  for i:=1 to n do
    Write(f,s[i]);
end;
```

Code to right-justify fields

```
{Helper routine that outputs and right-justifies integers}

procedure ExportInt(i: integer; n: integer);
var
  s: string;
begin
  s:=IntToStr(i);
  s:=copy(s,1,n);
  while (length(s)<n) do
    s:=' '+s;
  for i:=1 to n do
    Write(f,s[i]);
end;
```

For fixed-width fields, work with a "File of Bytes" instead of a TextFile. TextFile assumes the data is broken down by lines and have a 32,000 byte limit to the line length. File of bytes isn't limited.

Code for helper routine and file of bytes

```
{In this example, there are two fields: name and number of members }
{The name is 40 bytes. The number of members is 20 bytes wide and right-justified}

var
    f: File of bytes;

function PrintDefaultReport: boolean;
begin
    result:= false; {We will create the file in AfterBuildExport}
end;

function BeforeBuildExport: boolean;
begin
    result:= false; {We will create the file in AfterBuildExport}
end;

procedure ExportStr(s: string; n: integer);
var
    i: integer;
begin
    s:= copy(s,1,n);
    while length(s)<n do
        s:=s+' ';
    for i:= 1 to n do
        Write(f,s,[i]);
end;

procedure ExportInt(i: integer, n: integer);
var
    s: string;
begin
    s:=IntToStr(i);
    s:=copy(s,1,n);
    while (length(s)<n) do
        s:=' '+s;
    for i:=1 to n do
        Write(f,s[i]);
end;

procedure AfterBuildExport;
begin
    {Build the export}
    AssignFile(f,RepDlg.ExpFilename.Text);
    Rewrite(f);
    try
        with PDSMDData.FamTbl do
            begin
                First;
                while not EOF do
                    begin
                        ExportStr (PDSMDData.FamTbl['NameFormat1'],40);
                        ExportInt (PDSMDData.FamTbl['NumMem',10);
                        Next;
                    end;
            end;
    finally
        CloseFile(f);
    end;
end;
```

Adding a Summary

If you want to tell users what was exported, you can display a printable summary.

Code to create and display a summary

```
procedure BuildSummary;
var
    ErrorRepDlg: TErrorRepDlg;
begin
    {Show next form}
    ErrorRepDlg:= TErrorRepDlg.Create(Self);
    try
        PDSFixupDlg(ErrorRepDlg);
        ErrorRepDlg.RTFText.ReadOnly:= false;
        ErrorRepDlg.DetailBtn.Visible:= false;
        ErrorRepDlg.SummaryBtn.Caption:= '&Print';
        ErrorRepDlg.RTFText.Clear;
        MainRTF:= ErrorRepDlg.RTFText;
        MainFontName:= 'New Courier';

        ClearTabs;           {clear tab stops}
        SetTab(3*1440,0)     {tab stop at 3 inches}
        SetTab(4*1440,0)     {tab stop at 4 inches}

        {Here is what we want to say}
        println('Summary');
        println('');
        println(DateToStr(Now));
        println('');
        println('');

        {Get the dialog ready}
        ErrorRepDlg.ReportTitle:= 'Export Data for XYZ';
        ErrorRepDlg.ReportSettingsFileName:= trim(DMReport.RepTblFileName.AsString)+'.ini';
        ErrorRepDlg.ProcExpLabel.Caption:= 'Export Completed';
        ErrorRepDlg.PrintExpLabel.Caption:= '';
        ErrorRepDlg.RTFText.ReadOnly:= true;
        ErrorRepDlg.RTFText.CPPosition:= 0;
        PDSShowDlg(ErrorRepDlg);

    finally
        PDSFreeDlg(ErrorRepDlg);
    end;
end;
```

Then, at the end of the AfterBuildExport routine, add the following:

```
BuildSummary;
```

Finally, add a FinishBuildExport event that tells the program not to display the last dialog box:

Code to hide last dialog box

```
function FinishBuildExport: boolean;
begin
    Result:= false; {We already showed the summary, so we're finished}
end;
```

Exporting to OLE

You can export to a file, but you can also export to an OLE server, like Microsoft® Excel or Word.

Code to export to a spreadsheet

```
{This is a short routine that opens Excel and creates a spreadsheet}

function PrintDefaultReport: boolean;
begin
    result:= false; {We will create the file in AfterBuildExport}
end;

function BeforeBuildExport: boolean;
begin
    result:= false; {We will create the file in AfterBuildExport}
end;

procedure AfterBuildExport;
var
    v: OLEVariant;
begin
    v:= CreateOleObject('Excel.Application');
    v.Visible:= true;
    v.Workbooks.Add;
    v.ActiveSheet.Cells(5,1).Font.Size:= 20; {Gives the text a font size of 20}
    {Insert text in cell position "5,1" which is spreadsheet cell A5}
    v.ActiveSheet.Cells(5,1):= 'PDS Ledger Data';
end;
```

Importing Data With Scripting

You can import data using a script. Creating an import is similar to creating an export, but instead of exporting the data to a disk, it's read into the program. You can import text and fixed-width data, display progress to the user, and add a summary to the end of the import.

Useful Information

You must have extensive knowledge of the data you're importing. Someone without programming knowledge can import simple items, but you need a programmer when importing complicated items.

You can assign the file variable to the file name. Instead of doing a "rewrite" (which is what you performed in the export to delete the file and create a new file), do a "reset", which opens the file. First, you must check to make sure the file is there.

In the import, the program loops through the file and uses "readln" to read the next line into the string. Then, you can use the information. For example, assume the disk contains the families' ID numbers, a comma, and a number that is put in the families' geographical area. You must read in the line, break it into two parts, look up the family by ID, then insert the new value.

Example of importing data

```
var
    f: TextFile;

function PrintDefaultReport: boolean;
begin
    result:=false; {We will create the file in AfterBuildExport}
end;

function BeforeBuildExport: boolean;
begin
    result:=false; {We will create the file in AfterBuildExport}
end;

procedure AfterBuildExport;
label OpenTheFile;
var
    s: string;
    id, area: string;
    I: integer;
begin
    {Build the export}
    OpenTheFile: AssignFile(F,'a:\temp.txt');

    try
        Reset(f);
    except
        if PDSMessageDlg('Cannot open file.',mtError,mkSet(mbCancel,mbRetry),0)=mrRetry
            then goto OpenTheFile
            else exit;
        end;
    try
        while not eof(f) do
            begin
                try
                    Readln(f,s);
                    I:=pos(' ',s);
                except
                    I:=0;
                end;
                if I<>0 then
                    begin
                        Id:=copy(s,1,I-1);
                        {The ID number is in ParKey and is right justified}
                        Id:=copy(' '+id,length(id)+1,10);
                        Area:=Copy(s,I+1,length(s));
                        if PDSMDData>FamTbl.Locate('ParKey' ,id,mkset(loCasInsensitive)) then
                            begin
                                PDSMDData.FamTbl.edit;
                                PDSMDData.FamTblAreaNumber.AsString:=area;
                                PDSMDData.FamTbl.post;
                            end;
                    end;
            end;
        finally
            CloseFile(f);
        end;
    end;
end;
```

Manipulating the Data

You can manipulate data without pulling the information in from a file. For example, if you entered family names in the program incorrectly, you could change all family names data manipulation.

You can perform many advanced tasks with scripts. Below is one example.

Example of manipulating data

```
{This is a small report that adds a financial entry}
{to fund 1 Offering of $999.54 to family ID number 1}

uses
    AdvRep;          {Uses a procedure found in AdvRep.pas}

procedure AfterCreateReport;
begin
    if AddHistEntry(false,
                    '1',
                    'ParKey',
                    '1',
                    StrToDate('01/28/15'),
                    999.54,
                    'Offering',
                    '1234',
                    0)
    then showmessage('Success')
    else showmessage('Failed');
end;
```

Appendix

The appendix contains lists and tables related to ARW reports.

- [Built-In Routines](#)
- [Formation Embedded Lists](#)
- [UDR Options](#)

Built-In Routines

See the code below associated with each built-in routine:

Boolean Operations

Operation	Description
b1 and b2	
b2 or b2	
b1 xor b2	exclusive or
not b2	

Conversions

Conversion	Type
Chr(i)	string
CurrToStr(c)	string
DateTimeToStr(dt)	string
DateToStr(d)	string
FloatToStr(f)	string
IntToStr(i)	string
RGB(ir,ig,ib)	integer
StrToCur(s)	currency
StrToDate(s)	Tdate
StrToDateTime(s)	TDateTime
StrToFloat(s)	extended
StrToInt(s)	integer
StrToIntDef(s,i)	integer
StrToTime(s)	TDateTime
TimeToStr(t)	string

Date-Time Operations

Operation	Type
CurrentDate	TDate
CurrentDateTime	TDateTime
CurrentTime	TDateTime
DayOfWeek(d)	integer
DecodeDate(d,iy,im,id)	
DecodeTime(t,ih,im,is,ims)	
EncodeDate(iy,im,id)	TDate
EncodeTime(ih,im,is,ims)	TDateTime

Formatting

Code	Type
FormatCurr(s,c)	string
FormatDateTime(s,dt)	string
FormatFloat(s,f)	string
FormatMaskText(s1,s2)	string
FormatString(s1,s2)	string

Fund History Activity Types

Value	Description
-2	System Rate Change
-1	Rate Change
0	Payment - Deductible
1	PayDown - Deductible
2	Additional Gift - Deductible
3	Non-Cash - Deductible
4	Quid Pro Quo - Non-deductible
5	Payment - Non-deductible
6	PayDown - Non-deductible
7	Credit - Non-deductible
8	Payment from Last Year - Non-deductible

9	Recurring Charge
10	Charge
11	Refund - Deductible
12	Refund - Non-deductible
13	Hours Pledged
14	Hours Completed
15	Hours Remaining
16	Balance
17	Group Total
18	Ignore

Math Routines

Routine	Type
ArcTan(x)	extended
Cos(x)	extended
Cosh(x)	extended
Cotan(x)	extended
Exp(x)	extended
Frac(x)	extended
Int(x)	extended
IntPower(x,i)	extended
Ln(x)	extended
Power(x1,x2)	extended
Round(x)	integer
Sin(x)	extended
Sqr(x)	extended
Sqrt(x)	extended
Tan(x)	extended
Tanh(x)	extended
Trunc(x)	integer

Member Type Values

In embedded code, Mem.['Mem Type Number'].

In script, PDSMDData.MemTbl['MemberNum'].

Value	Description
0	Head
1	Spouse
2	Adult
3	Young Adult
4	Child
5	Other

Fund Period Values

In embedded code, CalcRate.['Calc Rate Terms Period'].

In script, PDSMDData.CalcRateTbl['FDPeriod'].

Value	Description
0	Weekly
1	Weekly on Sunday
2	Weekly on Monday
3	Weekly on Tuesday
4	Weekly on Wednesday
5	Weekly on Thursday
6	Weekly on Friday
7	Weekly on Saturday
8	Monthly
9	BiMonthly - every 2 months
10	SemiMonthly - twice a month
11	Quarterly
12	Semiannually - twice a year
13	Annually
14	Special

PDS Built-In Routines

Routine	Type
MainOverflow	
MainHeaderVisible	boolean
MainHeaderSetVisible(b)	
MainFooterVisible	boolean
MainFooterSetVisible(b)	
PDSGetCrossTabRowCount	integer

PDS Scripting Routines

Routine	Type
PDSGetParam(i)	string
PDSSetParams(i,s)	
CallFunc(s)	string
CallFundP1(s,i1)	string
CallFundP2(s,i1,i2)	string

Simple Math Operations

Operation	Description
i1+i2	add
i1-i2	subtract
i1*i2	multiply
i1/i2	divide
i1 div i2	integer division
i1 mod i2	remainder
-i1	unary minus

String Operations

Operation	Description
s1+s2	string concatenation
Capitalize(s)	string
Copy(s,i1,i2)	string

Delete(s,i1,i2)	
Fixed(s)	string
FixedName(s)	string
FixedAddress(s)	string
FixedCity(s)	string
Insert(s1,s2,1)	
Length(s)	integer
LowerCase(s)	string
Pos(s1,s2)	integer
Trim(s)	string
TrimLeft(s)	string
TrimRight(s)	string
UpperCase(s)	string

Formation Embedded Lists

Below are the predefined embedded lists for PDS Formation Office.

Code	Description
<List: All Pledges>	Total of all families for all pledges for selected funds
<List: Comm>	Communion data for each member
<List: Confirm>	Confirmation data for each member
<List: Coupon Bottom>	Prints a return coupon at the bottom of the letter
<List: Coupon Top>	Prints a return coupon at the top of the letter
<List: Delinq>	Family's delinquency information
<List: Fam Sched>	Class schedule for all students in the family
<List: Fund Month>	Fund name and total paid each month
<List: Gifts>	Additional gifts totals
<List: Grp List>	Fund groups with Due, Paid, "Balance for stmt"
<List: Grp List 2>	Fund groups with Due, "Total Tax deduct.", balance
<List: Grp List 3>	Fund groups showing Amt Paid only
<List: Hist List>	Activity history for the selected fund
<List: Hour List>	Hours due, completed, balance (similar to Grp List)
<List: IRS Note>	Prints a special tax note at the end of tax statements
<List: Mem Amt>	Member of family and amount each has paid
<List: Receipt History With Desc>	Activity history with date and description
<List: Running Total>	Activity history with cumulative due/paid totals
<List: Simple Totals>	Total Pledged, Paid, Balance for all selected funds

UDR Options

Under **[Default]** in the UDR file, you can use the following code to control your report.

For Use in Fund Reports

Code	Description
FundLimit = xxx	The number of funds the report can print.
IncludeFundIncludeHist=1	
UseSepRecapDate=1	
RelaxDates=1	
FuncType=1	<p>Tells the program what functions are available on the Fund Selection window.</p> <ul style="list-style-type: none"> • 0-4 items: Don't include, use in month, group totals only, itemize • 1-2 items: Don't include, include group • 2-3 items: Don't include, include group, itemize • other-2 items: Don't include, include group, and in months

For Use in PDS Church Office and Formation Office Reports

Code	Description
FamType=YNNYYN	Y - Active, N - Inactive, N - Loose, Y - Fund, Y - Student, N - Teacher
MemType=YN	Y - Active, N - Inactive
CanUseEMail=1	Tells the program to create email options in the Sortation section on the Selection Information tab.
CanSkipEMail=1	
CanLog=1	
CanUseCC=1	
CanUseBulk=1	
DisableScaling=1	Tells the program not to reposition items to take up available paper size.
UseBap=1	
ForceMailTbl=1	Tells the program to build the mail table.
OrderMailTbl=1	
UseBap=1	Tells the program to use the place of baptism as the address, city/state, and ZIP code.

For Use in PDS Church Office Reports

Code	Description
CanUseSepStmnt=1	Tells the program to treat members with separate stmnt selected differently.
CanUseSepMem=1	

For Use in PDS Ledger/Payroll Reports

Code	Description
CRasNeg=1	Make credits as negative debits.
TrnType=1NNNNNNNNNN	Indicates which transaction types are included by default on the Transaction Selections tab.
ForceTrnType=1xxxxNxxxx	
CoaType=YYNNNNYNYYY	Indicates which COA types are included by default on the Chart of Accounts selections tab.
ForceCoaType=YYYxxxxNxxxx	
IsBalanceSheet=1	Indicates the report is a balance sheet.
IsIncomeStmnt=1	Indicates the report is an income statement.
IsGeneralLedger=1	Indicates the report is a general ledger report.
IsCashFlow=1	Indicates the report is a cash flow report.
JustBeginningBalances=1	Indicates to use only the Beginning Balance values.

Exercises and Example Project

We've included some exercises and a sample project so you can practice writing advanced reports.

- [Exercise 1](#) - Create a new advanced report
- [Exercise 2](#) - Create a family report with member sub-report
- [Exercise 3](#) - Create a region with labels and data fields
- [Exercise 4](#) - Create an embedded list report
- [Project](#) - Build a Family Directory report

Exercise 1

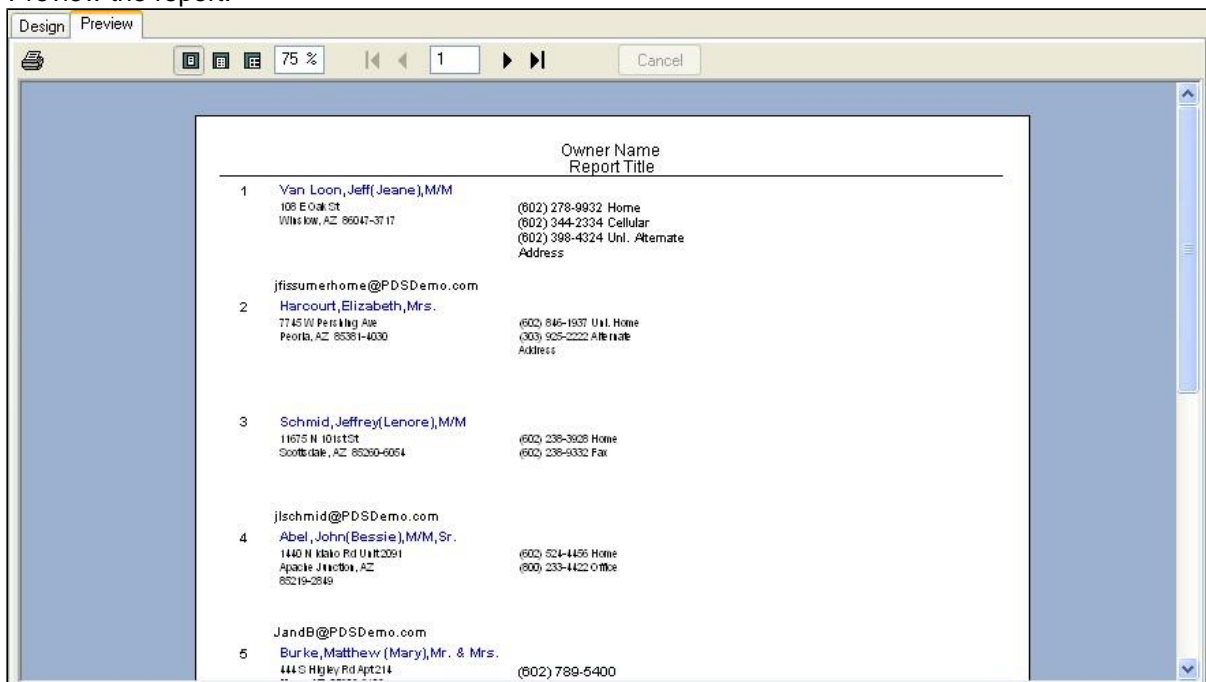
Create a new advanced report using the PDS Header/Footer template. Include the following fields and options:

- Family ID, in bold black text
- Family Name, next to the ID, in bold blue text
- Family Address Block, beneath the Family Name, in regular text (hint: use DBMemo)
- Family Phone List, next to Family Address Block, in regular text (hint: use DBMemo)
- Family Home Email, beneath Family Address Block, and ShiftWithParent

In the Summary band, add a count of families.

Here's how to do it!

1. On the ribbon, click **Reports > Family Reports**.
2. In the Select Report window, click **Add**.
3. In the Add Report window, click the **Advanced Reports** tab, and click **Report**.
4. Click the **Templates** menu, and select **PDS Header/Footer**.
5. In the top left corner of the Detail band, insert a **DBText** object. In the drop-down lists, select **Fam** and **Fam ID/Env Number**. Bold the text.
6. Next to the Family ID, insert a **DBText** object. Select **Fam** and **Fam Name**. Right-click the object and select **AutoSize**. Bold the text and change the font color to blue.
7. Directly beneath the Family Name, insert a **DBMemo** object. Select **Fam** and **Fam Address Block**. Right-click the object and select **Stretch**.
8. Next to the Address Block, insert a **DBMemo** object. Select **Fam** and **Fam Phone List**. Right-click the object and select **Stretch**.
9. Directly beneath the Address Block, insert a **DBText** object. Select **Fam** and **Fam Email Home**. Right-click the object and select **AutoSize**.
10. Click the **Report** menu, and select **Summary**.
11. In the Summary band, insert a **DBCcalc** object. Right-click it, select **Calculations**, select **Count**, and click **OK**. Select **Fam** and **Fam Unique ID**. Right-click the object and click **AutoSize**.
12. Preview the report.

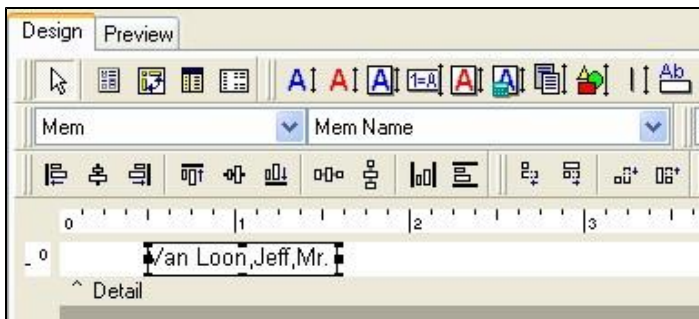


Exercise 2

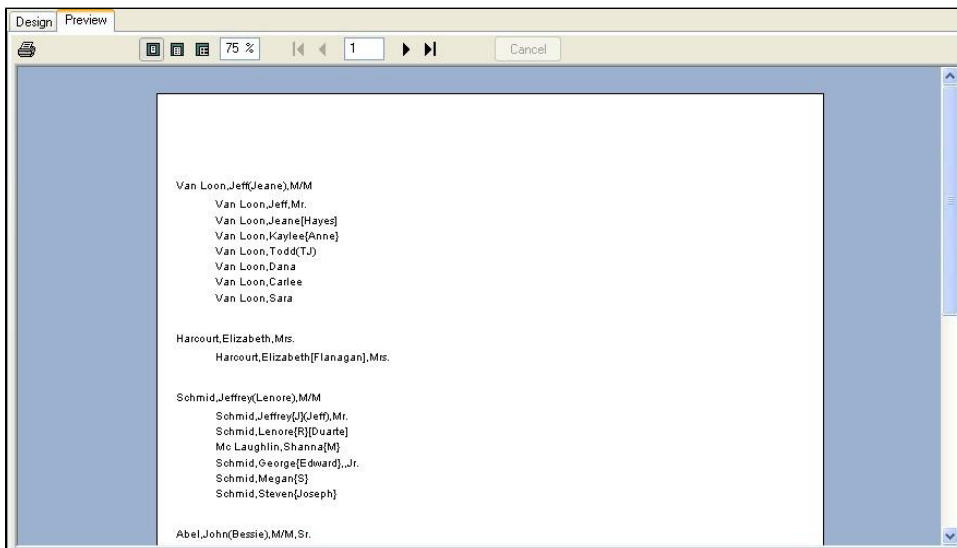
Create a family report with a sub-report of member names.

Here's how to do it!

1. Create a new ARW report.
2. In the Detail band of the main report, insert a **DBText** object, and select **Fam** and **Fam Name**. Right-click the object and select **AutoSize**.
3. In the Detail band, insert a **SubReport** object beneath the Fam Name.
4. Select the sub-report, and select **Mem** as the data pipeline name.
5. Click the **PDS SubReport1:Mem** tab at the bottom of the workspace.
6. To turn off the Title and Summary bands, click **Report > Title**, then click **Report > Summary**.
7. In the Detail band of the sub-report, insert a **DBText** object, and select **Mem** and **Mem Name**. Right-click the object and select **AutoSize**. Using the horizontal ruler, place the object at 0.5 in, then manually resize the Detail band to the height of the object.



8. Preview the report.



Exercise 3

Using the report you created in Exercise 1, create a region next to the Family Phone List, and include the following objects:

- A label and data field for **Date Registered**.
- A label and data field for **Geographic Area**.
- A label and data field for **Family Status**.
- Place a line object under each data field. This creates a "fill-in-the-blanks" area. If the data has already been entered in the program, it displays on these lines. If the data is not in the program, the lines are blank and ready to be completed.

Here's how to do it!

1. Either copy or modify the report from the first exercise.
2. On the Listing Layout window, click **Modify the Report**.
3. In the Detail band, insert a **Region** object beside the Family Phone List.
4. Inside the Region object, insert 3 **Label** objects and enter the appropriate text for each.
5. Beside the labels, still inside the Region object, insert **DBText** objects, and select **Fam Date Registered**, **Fam Geog. Area**, and **Fam Family Status**.
6. Insert a **Line** object under each DBText object. Resize as needed.
7. Preview the report.

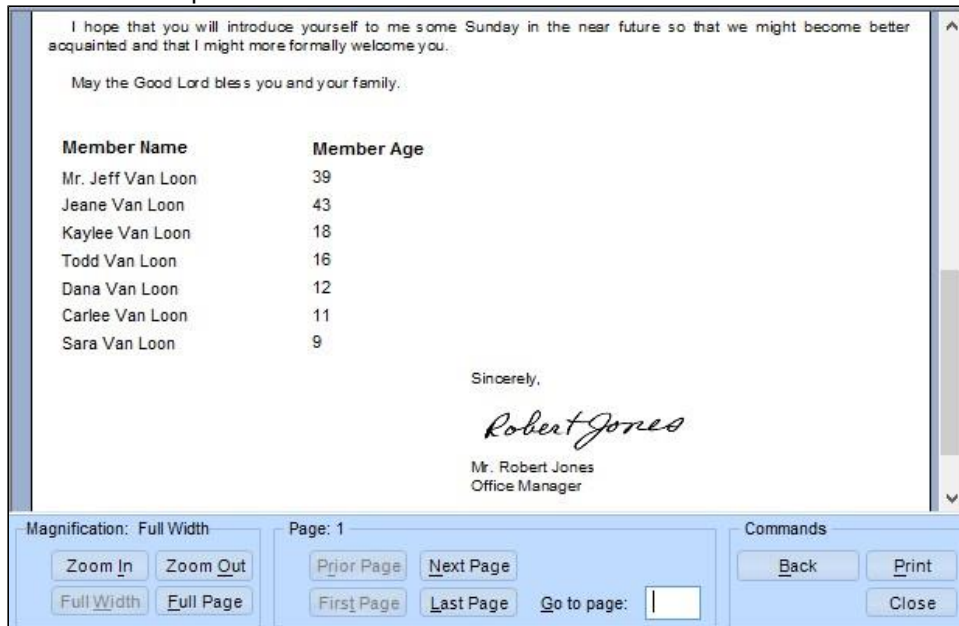
	Owner Name Report Title		
1	Van Loan, Jeff (Jeans), M/M P.O. Box 322 Peoria, AZ 85336 jvanloon@yahoo.com	(602) 278-9932 Home (602) 344-2334 Cell (602) 398-4324 Uni. Alternate	Date Registered 10/05/1990 Geog. Area 21-33 Family Status Couple
2	Harcourt, Elizabeth, Mrs. 8745 N Forest Grove Ln Aspen, CO 81611	(602) 846-1937 Uni. Home (303) 925-2222 Alternate Addr.	Date Registered 06/01/1970 Geog. Area 4-20 Family Status Individual
3	Schmid, Jeffrey (Lenore), M/M 5827 W. Missouri Ave. Peoria, AZ 85329-2904	(602) 238-3928 Home (602) 238-9332 Fax	Date Registered 11/01/1987 Geog. Area 20-49 Family Status Extended Family
13	Ramirez, Tomas (Louise), M/M P.O. Box 234 Glendale, AZ 85219-8548	(602) 466-2134 Home	Date Registered 03/01/1993 Geog. Area 26-25 Family Status Two Parent Family
17	Garinger, Ronald (Lucero Garinger, Jenny, Ms.), Mr. 4128 N. Emerald Cove Phoenix, AZ 85037	(602) 772-2434 Home (602) 318-9191 Pager	Date Registered 03/01/1996

Exercise 4

Create an embedded list report of members and their ages, then insert it into a family letter report.

Here's how to do it!

1. In Family Reports, expand **Letters/Statements** and select **Family Welcome Letter**.
2. In the Letter Layout window, click **Modify the Body of the Letter**.
3. Click **File > Edit List Selections**. Enter **List-MemList.rtm**, and click **Open**.
4. In the ARW editor, remove the header and footer bands, and add the Title band.
5. In the top-left corner of the Detail band, insert a **DBText** object. Select **Mem** and **Mem Name Format 1**. Right-click the object and select **AutoSize**.
6. In the Detail band, next to the Member Name, insert another **DBText** object. Right-click it, select **Position**, and enter "2.5" for Left. Click **Apply > OK**. Select **Mem** and **Mem Age**. Right-click it and select **AutoSize**.
7. Click and drag the Detail band so there is no white space below the Member Name and Age.
8. In the top-left of the Title band, insert a **Label** object. Enter "Member Name" and bold the text.
9. In the Title band, next to Member Name, insert another **Label** object. Right-click it, select **Position**, and enter "2.5" for Left. Click **Apply > OK**. Enter "Member Age" and bold the text.
10. Click and drag the Title band so there is no white space below Member Name and Age.
11. Save your work, and close the editor.
12. In the letter editor, insert a few blank lines at the bottom, and enter **<List: MemList>**.
13. Click **OK**, then complete the remaining steps in the report wizard.
14. Preview the report.



Project - Building the Family Directory

In this project, you will create a Family Directory report.

There are three parts to this project:

- Part 1 - Add a new report, set the header, add details, add a sub-report, and add a group.
- Part 2 - Create and initialize global variables, add embedded code, and run the report.
- Part 3 - Add screen routines, declare objects, create a new screen, save and load screen information, and communicate with the report.

Part 1 - Start the Report

To start the project, we'll add a new advanced report and set up the Header and Detail bands with family information. We'll also add a sub-report of members, and group each family together.

Use what you learned in the [Report Editor](#), [Sub-Reports](#), [Groups](#), and [Regions](#) sections to complete these steps.

Steps

Start the Report

1. In Family Reports, click **Add**.
2. Click the Advanced Reports tab, then click **Report**.
3. Click **File > Close**, then **Yes** to save the report.

Useful Information

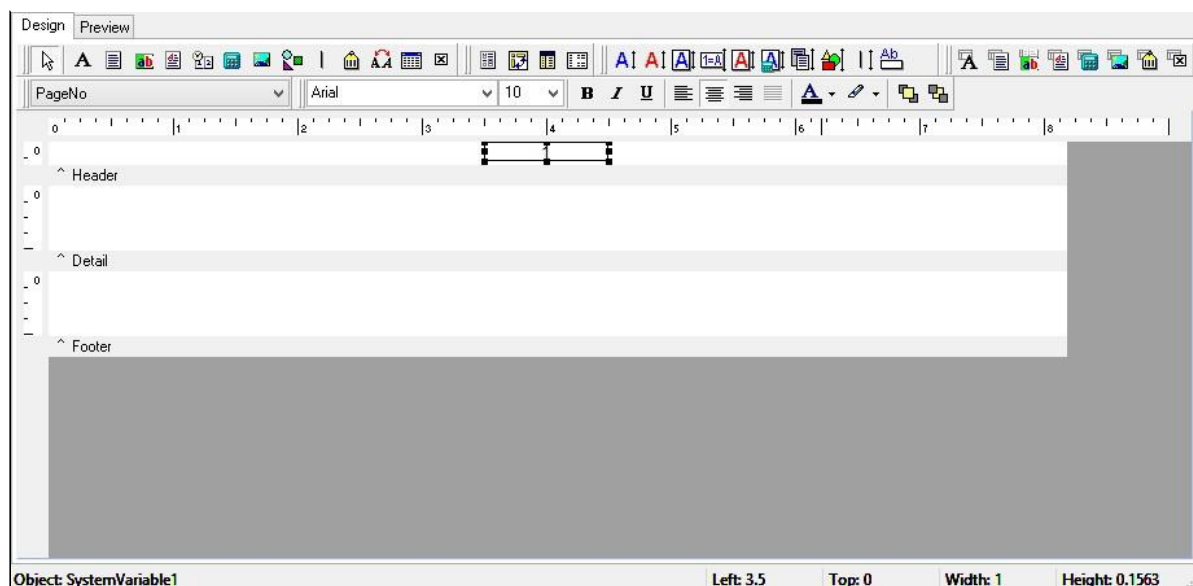
For this project, we want to name the report before we start working on it. However, you can name a report at any point during the creative process.

4. In the Overview window, change the report name to "Family Directory Tutorial", and change the description to "Family Directory built from scratch".
5. Click **Next**. Select your printer settings, then click **Next**.
6. Click **Modify the Report**. Now that the report is saved, we can start creating it.

The Header

1. In the editor, click the **Report** menu and click **Data**. Select **Fam**, and click **OK**.
2. In the Header band, insert a **System Variable** object. Center the value inside of the object. In the drop-down list at the top, select **PageNo**.
3. Right-click the object and select **Position**. Enter "3.5" for Left, "0" for Top, and "1" for Width. Click **Apply > OK**.
4. Right-click the Header band and select **Position**. Enter "0.1875" for Height. Click **Apply > OK**.

After completing these steps, your report should look like this:



The Detail

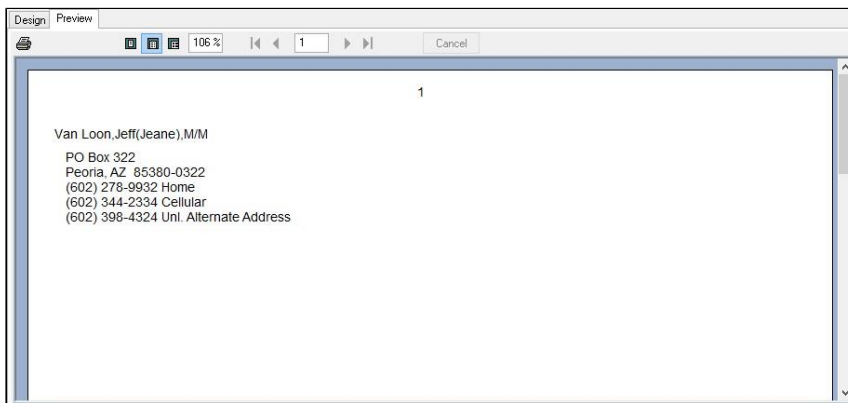
Next, we will add the Family Name, Address Block, and Phone List to the Detail band. We'll leave space for a column heading letter, taken from the first letter of the last name, which we'll add later.

1. In the Detail band, insert a **DBText** object. Left-align the text. In the drop-down list at the top, select **Fam** and **Fam Name**.
2. Right-click the object and select **Position**. Enter "0.125" for Left, "0.25" for Top, and "3" for Width. Click **Apply > OK**.
3. In the Detail band, beneath the Fam Name, insert a **DBMemo** object. In the drop-down list, select **Fam** and **Fam Address Block**.
4. Right-click the object and select **Position**. Enter "0.25" for Left, "0.5" for Top, "3" for Width, and "0.2" for Height. Click **Apply > OK**.
5. Right-click it again and select **Stretch**.
6. In the Detail band, beneath the Fam Address Block, insert a **DBMemo** object. In the drop-down list, select **Fam** and **Fam Phone List**.
7. Right-click the object and select **Position**. Enter "0.25" for Left, "0.6875" for Top, "3" for Width, and "0.2" for Height. Click **Apply > OK**.
8. Right-click it again and select **Stretch**.
9. Right-click it again and select **ShiftRelativeTo**, select **DBMemo1**, and click **OK**.

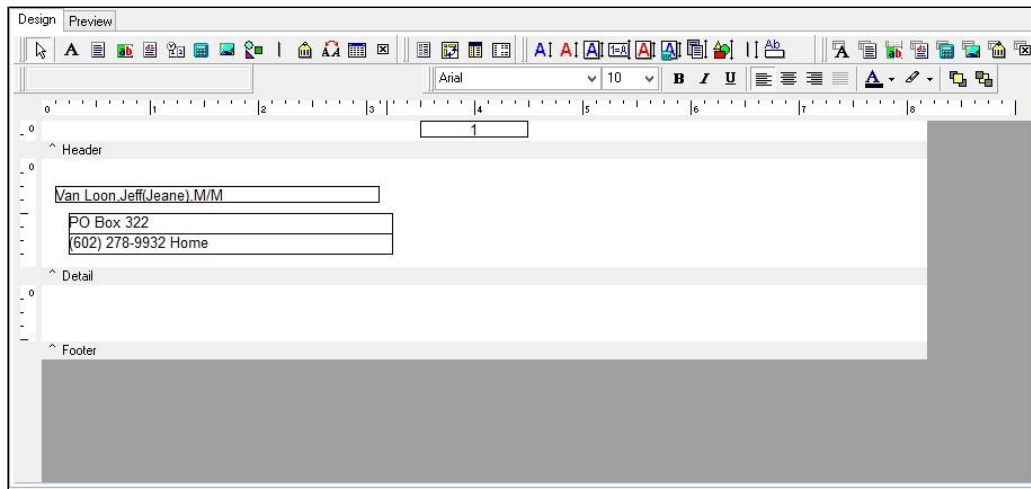
Useful Information

ShiftRelativeTo positions the object based on another object. In this case, we want to keep the Phone List below the Address Block, no matter how many lines the address has. This way, the objects won't print on top of each other.

10. Preview the report.



After completing these steps, your report should look like this:

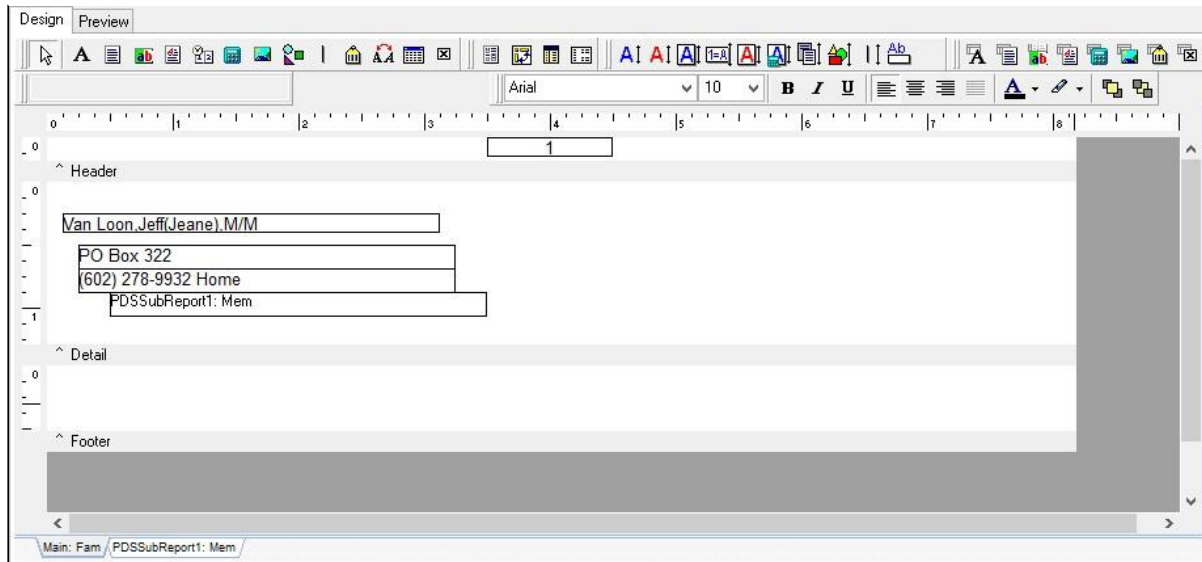


Add the Sub-Report

Next, we'll add a member sub-report to the Detail band.

1. In the Detail band, beneath the Fam Phone List, insert a **SubReport**. In the drop-down list, select **Mem**.
2. Right-click the sub-report and select **ParentWidth** to turn off the automatic horizontal stretching.
3. Right-click it again and select **Position**. Enter "0.5" for Left, "0.875" for Top, and "3" for Width. Click **Apply > OK**.
4. Right-click it again and select **ShiftRelativeTo**, select **DBMemo2**, and click **OK**.

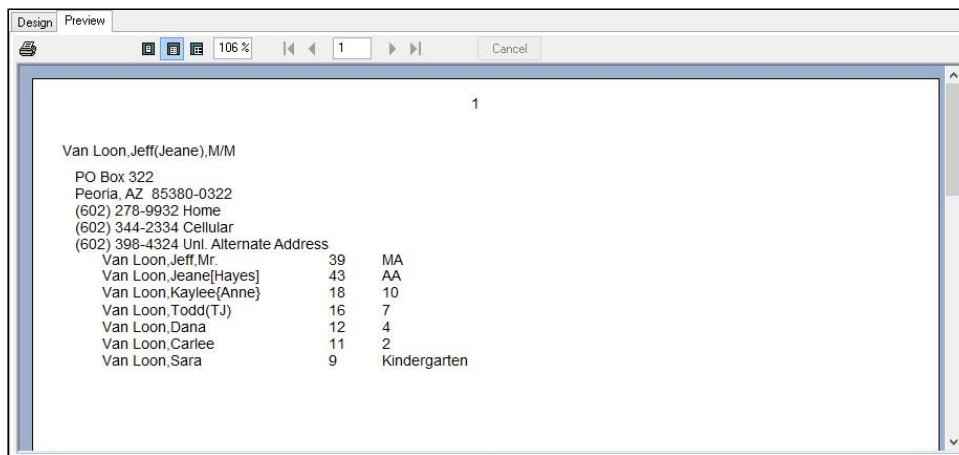
After completing these steps, your report should look like this:



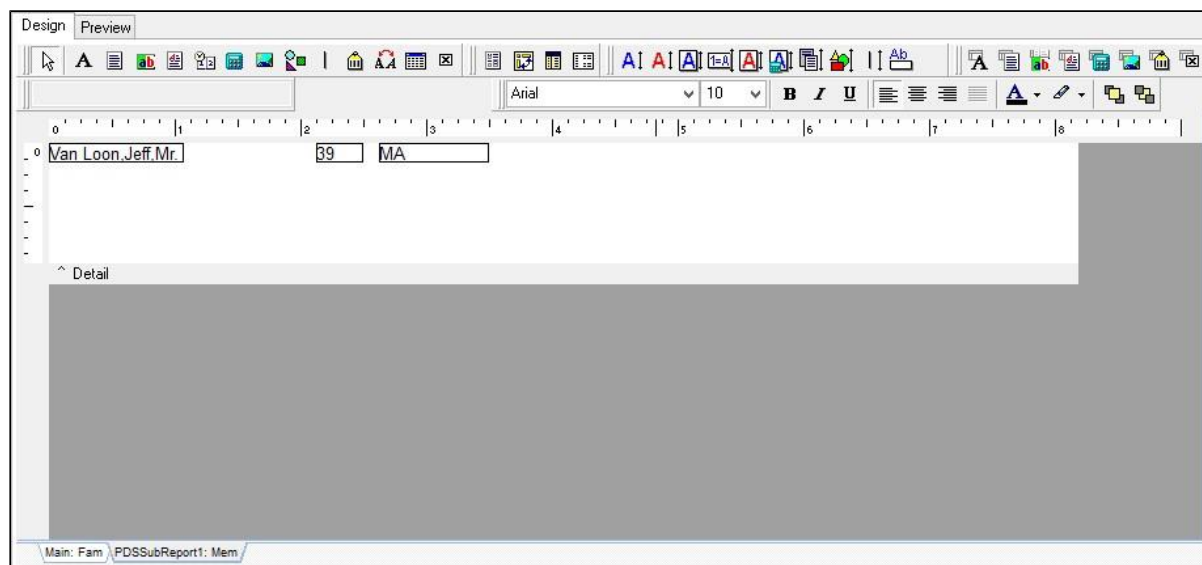
Set Up the Sub-Report

Now that we have a sub-report, we can add the appropriate objects to it.

1. Click the **PDSSubReport1: Mem** tab at the bottom.
2. Turn off the Title and Summary bands from the Report menu.
3. Right-click the Detail band and select **DynamicHeight** to move the band vertically to match the objects inside.
4. In the Detail band, insert a **DBText** object. In the drop-down lists, select **Mem** and **Mem Name**.
5. Right-click the object and select **Position**. Enter **0** for Left and Top, and **2** for Width. Click **Apply > OK**.
6. Right-click it again and select **AutoSize**.
7. Next to Mem Name, insert another **DBText** object. In the drop-down lists, select **Mem** and **Mem Age**.
8. Right-click the object and select **Position**. Enter "2.125" for Left, "0" for Top, and "0.375" for Width. Click **Apply > OK**.
9. Next to Mem Age, insert another **DBText** object. In the drop-down lists, select **Mem** and **Mem Grade**.
10. Right-click the object and select **Position**. Enter "2.625" for Left, "0" for Top, and "0.875" for Width. Click **Apply > OK**.
11. Preview the report.



After completing these steps, your report should look like this:



Add the Group

When we print this directory, we don't want to split a family across a page. To avoid this, we can create a group.

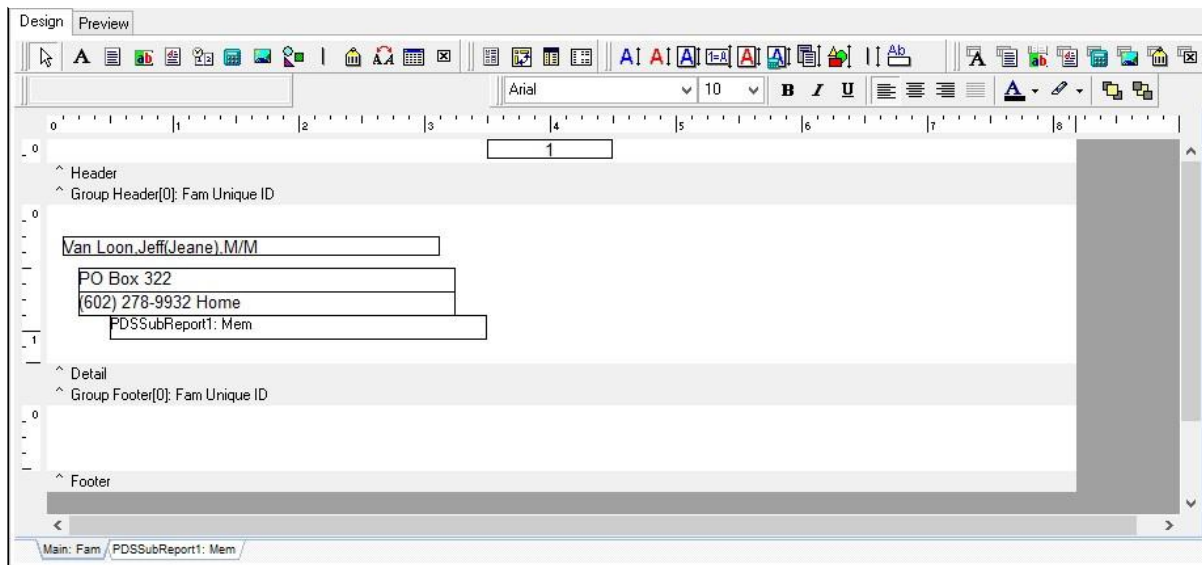
1. Click the **Main: Fam** tab at the bottom.
2. Click the **Report** menu and click **Groups**. Select **Fam.Fam Unique ID**, and click **Add**.

Useful Information

Fam.Fam Unique ID identifies each family in a unique way. If you select Fam Name instead, the group can't differentiate families with the same name.

3. By default, the **Keep group together** option is selected. Click **OK**.
4. Click **File > Save**. Complete the steps in the wizard to see the finished report.

After completing these steps, your report should look like this:



Finished Product

We have finished creating the basic Family Directory report. When you print, the final report should look like this:

1		
Van Loon, Jeff(Jeane), M/M		
PO Box 322		
Peoria, AZ 85380-0322		
(602) 278-9932 Home		
(602) 344-2334 Cellular		
(602) 398-4324 Uni. Alternate Address		
Van Loon, Jeff, Mr.	39	MA
Van Loon, Jeane[Hayes]	43	AA
Van Loon, Kaylee[Anne]	18	10
Van Loon, Todd(TJ)	18	7
Van Loon, Dana	12	4
Van Loon, Carlee	11	2
Van Loon, Sara	9	Kindergarten
Harcourt, Elizabeth, Mrs.		
10194 E Paradise Dr		
Scottsdale, AZ 85260-5915		
(602) 846-1937 Uni. Home		
(303) 925-2222 Alternate Address		
Harcourt, Elizabeth[Flanagan], Mrs.	78	MA
Schmid, Jeffrey(Lenore), M/M		
11675 N 101st St		
Scottsdale, AZ 85260-6054		
(602) 238-3928 Home		
(602) 238-9332 Fax		
Schmid, Jeffrey(J)(Jeff), Mr.	41	14
Schmid, Lenore(R)[Duarte]	41	14
Mc Laughlin, Shanna(M)	20	11
Schmid, George(Edward), Jr.	18	5
Schmid, Megan(S)	14	3
Schmid, Steven(Joseph)	11	1
Abel, John(Bessie), M/M, Sr.		
1440 N Idaho Rd Unit 2091		
Apache Junction, AZ 85119-2849		

Now, we can take a look at adding to the report in Part 2.

Part 2 - Add Embedded Code

In Part 1 of this project, we created the Family Directory report from scratch. In this part, we'll add a label to the Detail band above the Family Name that contains the first letter of the family's last name. This letter displays at the top of the page or when the letter changes.

To do this, you will create and initialize global variables, and add embedded code to events. Use what you learned in the [Embedding Code and Data](#) section to complete the steps.

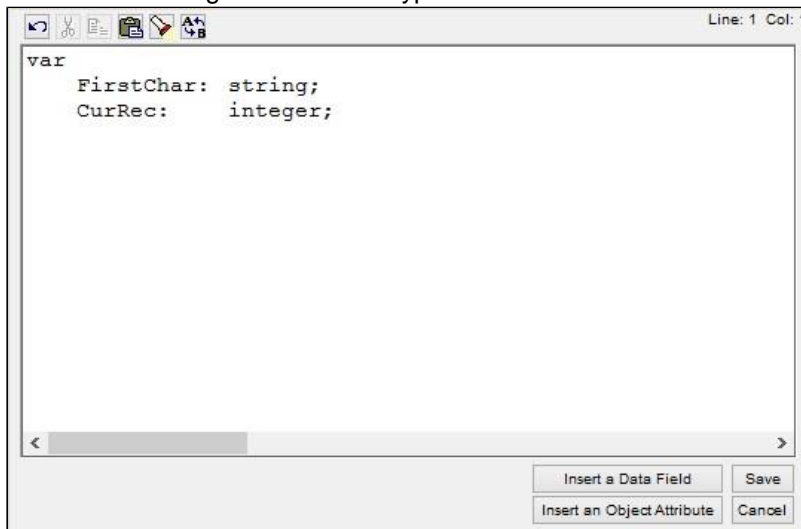
Steps

Open the Report

In Family reports, locate the Family Directory Tutorial report you created in Part 1, and click **Next**. On the layout window, click **Modify the Report**.

Create Global Variables

1. Click the **Embedded Code** menu, and select **Define Global Variables**.
2. Enter the following variables and types:



```
var
  FirstChar: string;
  CurRec: integer;
```

i Useful Information

FirstChar represents the first letter of each family name as it's encountered by the report. CurRec represents the current family record the report focuses on.

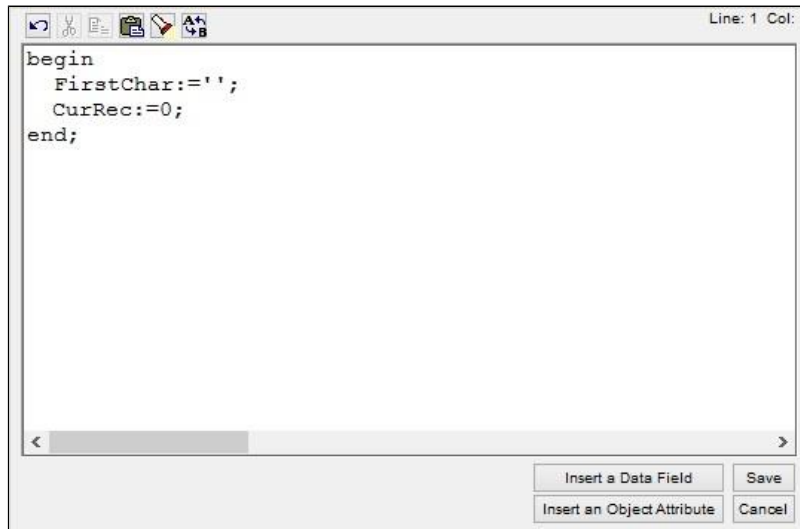
3. Click **Save**.

Initialize Global Variables

1. Click the **Embedded Code** menu, and select **Initialize Global Variables**.
2. Between the BEGIN and END statements, do the following:
 - Set FirstChar equal to a null character, which is represented by two single quotes with no space in between. This is not a double quote.
 - Set CurRec equal to zero.

 **Tip**

Use colon-equals (:=) in assignment statements, instead of regular equals (=).



```
begin
  FirstChar:='';
  CurRec:=0;
end;
```

The screenshot shows a code editor window with a toolbar at the top and a status bar at the bottom right indicating "Line: 1 Col: 1". The code editor contains the following text:

3. Click **Save**.

Inactive Families

We can add embedded code so that the word "Inactive" prints next to inactive families.

1. In the editor, select the DBText1 object for Family Name.
2. Right-click it and select **Get Text**.
3. Above the BEGIN statement, create a local variable "s", which is a string type to represent whether the family is inactive.
4. Inside the BEGIN-END statement, set "s" equal to the Fam Fam Inactive field. You can enter this manually or click **Insert a Data Field** to select the field from the Fam table.
5. Add an IF-THEN-ELSE statement that says, "if 's' is equal to 'yes', then the text of the field is equal to itself plus the word 'inactive', else the text is equal to itself."

```

var
  s: string;
begin
  s:=Fam['Fam Inactive'];
  if s='Yes'
    then Text:=Text+' (inactive)'
    else Text:=Text;
end;

```

6. Click **Save**.

Note the following about this block of code:

- The equals (=) used to compare "s" to "Yes" is not preceded by a colon (:). You don't use colons in comparison statements.
- The IF and THEN statements are not followed by a semicolon (;). You only use a semicolon after the last statement of a block.
- The ELSE statement assigns Text equal to itself. This is the text of DBText1. You must provide direction for the code when the IF statement is false. Otherwise, the text will be blank.

Create the New Label

1. In the Detail band, above Family Name, insert a **Label** object.
2. Bold the label text.
3. Right-click the label and select **Do Before Print**.
4. Enter the following code:

```

begin
  Label1.Caption:=Copy(Fam['Fam Name'],1,1);
  if Fam ['Fam Unique ID']<>CurRec then
    if Label1.Caption<>FirstChar
      then Label1.Visible:=True
      else Label1.Visible:=False;
  CurRec:=Fam['Fam Unique ID'];
  FirstChar:=Label1.Caption;
end;

```



Tip

Try using **Insert Data Field** and **Insert an Object Attribute** instead of manually entering the code.

5. Click **Save**.

Here's what the code is doing:

- Assigns the caption of Label1 equal to the first letter of Fam Fam Name. The Copy(s,i,j) command takes string "s" and copies "j" letters, starting at position "i".
- If the family's unique ID number is not equal to the current record, then:
 - If the caption of Label1 (the first letter of the current family's last name) is not already equal to the variable FirstChar:
 - Then turn on the visibility of Label1 because it has found a new letter.
 - Else turn off the visibility of Label1 because this family has the same initial letter as the previous family.
- Regardless of the results of the IF statement, set CurRec equal to the current family's ID number. Set FirstChar equal to the current value of Label1.
- In other words, if the last initial of the current family is the same as the previous family, the label is not printed. If it's different, the label prints, then sets this family's ID and initial as the new test values, then uses those to test the next family.

Run the Report

1. In the editor, click **File > Save**, then **File > Close**.
2. Click **Next**.
3. In the Select Families window, select **Name** as the sort order.
4. Preview the report.
5. When you're ready, print the report.

Finished Product

After completing these steps, when you print the report, it should look like this:

A		
Aaron,Carl(Penelope),M/M		
Aaron,Carl,Mr.		
Aaron,Penelope(Penny)[Pickle],Mrs.		
Aaron,Benjamin(Ben),Mr.		
Avillon,Laura(Garison,Lisa,Mrs.),Ms.		
17026 E Monterey Dr		
Fountain Hills, AZ 85268-6211		
(802) 827-5310 Home		
Avillon,Laura,Ms.	62	13
Garison,Lisa[Avillon],Mrs.	42	RN
B		
Babbett,Joseph(Agnes),M/M		
PO Box 9493		
Phoenix, AZ 85068-9493		
(602) 765-3131 Home		
(719) 753-9393 Intl. Alternate Address		
Babbett,Joseph[James],Mr.	60	14
Babbett,Agnes[June],Mrs.	50	14
Babbett,Matthew[T],Mr.	28	LPN
Babbett,Melissa,Miss	23	12
Bacote,Richard(Deborah),M/M		
4241 E Sheena Dr		
Phoenix, AZ 85032-5829		
(843) 865-8599 Home		
Bacote,Richard[L],Mr.	42	AA
Bacote,Deborah[Lewis],Mrs.	38	AA
Bacote,Ron[M],Mr.	8	Pre-School

Now, we can learn to add a layout window to further customize the report in Part 3.

Part 3 - Add a Layout Window

In Part 1 and Part 2, we created the Family Directory report and added last name initials at the beginning of each group of families.

In this last part, we'll add a layout window to the report wizard.

Steps

Add the Screen Routines

The program has a library of [common routines](#) you can insert into your script.

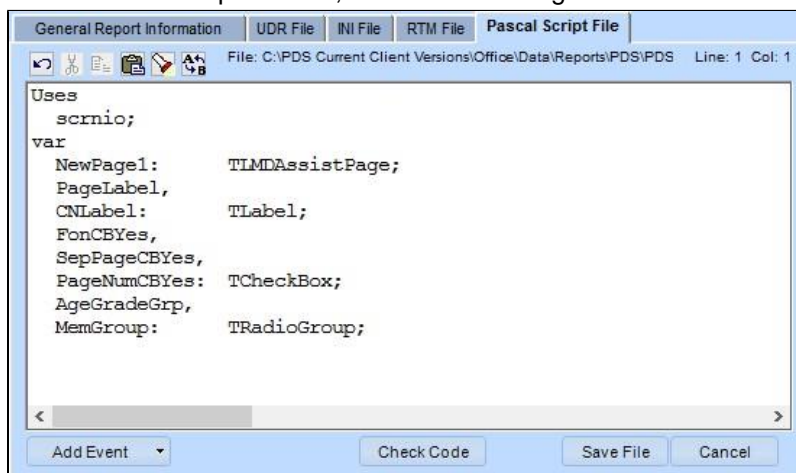
1. In the Select Report window, select the Family Directory Tutorial report we created.
2. Click **Adv. Script** at the bottom of the window.
3. Click the **Pascal Script File** tab.
4. Enter the following code in the text box:



5. Click **Save File**.

Declare the Objects

1. On the Pascal Script File tab, enter the following code below the routine:



2. Click **Save File**.

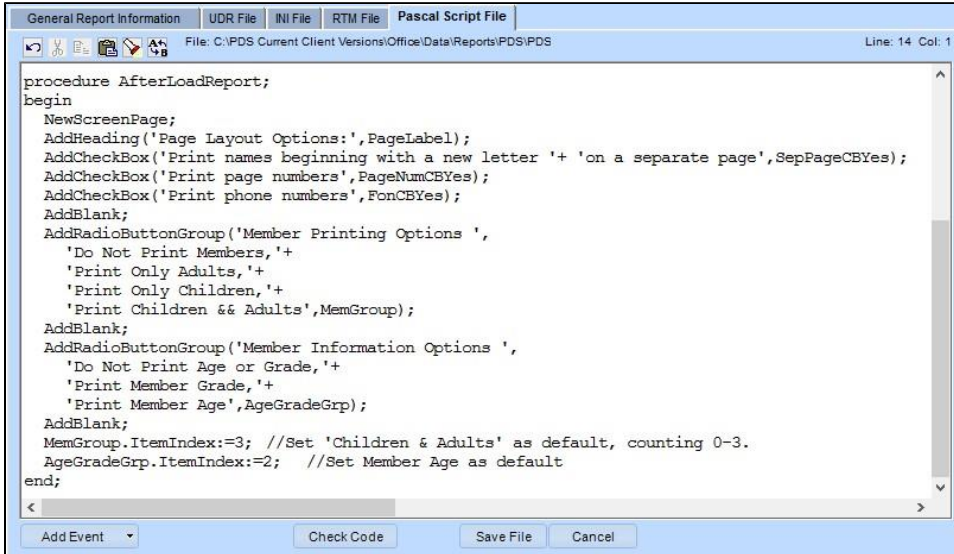
Create the New Window

1. On the Pascal Script File tab, place the cursor below the existing code.
2. Click **Add Event** and select **AfterLoadReport**.

Useful Information

The AfterLoadReport event executes before you get to the window but after the report is created.

3. Between the BEGIN and END statements, enter the following:



```
procedure AfterLoadReport;
begin
  NewScreenPage;
  AddHeading('Page Layout Options:',PageLabel);
  AddCheckBox('Print names beginning with a new letter '+' on a separate page',SepPageCByes);
  AddCheckBox('Print page numbers',PageNumCByes);
  AddCheckBox('Print phone numbers',FonCByes);
  AddBlank;
  AddRadioButtonGroup('Member Printing Options ',
    'Do Not Print Members, '+
    'Print Only Adults, '+
    'Print Only Children, '+
    'Print Children && Adults',MemGroup);
  AddBlank;
  AddRadioButtonGroup('Member Information Options ',
    'Do Not Print Age or Grade, '+
    'Print Member Grade, '+
    'Print Member Age',AgeGradeGrp);
  AddBlank;
  MemGroup.ItemIndex:=3; //Set 'Children & Adults' as default, counting 0-3.
  AgeGradeGrp.ItemIndex:=2; //Set Member Age as default
end;
```

4. Click **Save File**.
5. Click **Close**.
6. In the report wizard, click **Next** until you see the new Additional Layout window we created.



Page Layout Options:

- Print names beginning with a new letter on a separate page
- Print page numbers
- Print phone numbers

Member Printing Options

- Do Not Print Members
- Print Only Adults
- Print Only Children
- Print Children & Adults

Member Information Options

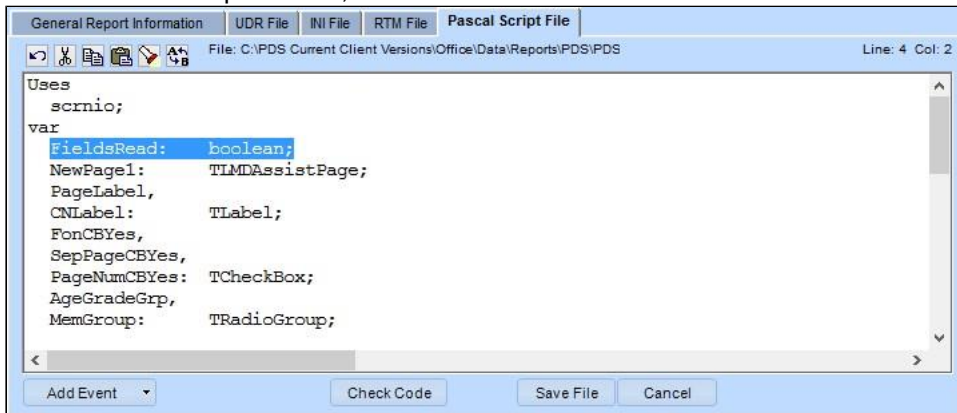
- Do Not Print Age or Grade
- Print Member Grade
- Print Member Age

Save and Load Window Information

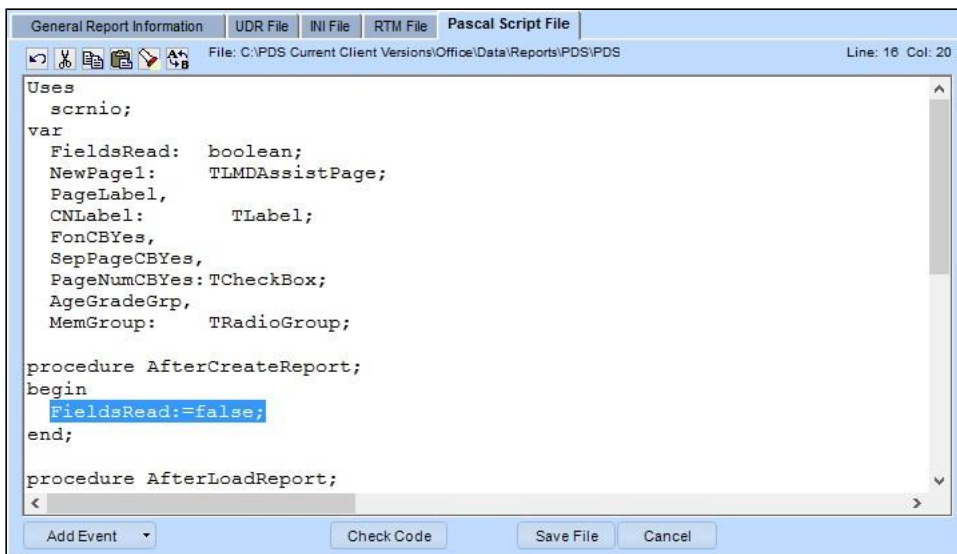
Now we want to save the information so we have the same values next time we run the report.

The program saves report values right before the report is finished in the BeforeDestroyReport in the INI file. If the program stops due to an error before it read the old values, we don't want to save the values because the good values will be overwritten with blank values.

1. Go back to the Select Report window, select the report, and click **Adv. Script** at the bottom.
2. On the Pascal Script File tab, add a variable called "FieldsRead".



3. Below the variables code, click **Add Event** and select **AfterCreateReport**.
4. Between the BEGIN and END statements, initialize FieldsRead to "False".



5. Below the AfterCreateReport event, click **Add Event** and select **BeforeDestroyReport**.
6. In the procedure, enter the following:

```

procedure BeforeDestroyReport;
var
  IniFile:TIniFile;
begin
  if FieldsRead then
  begin
    // ACCESS INI FILE save the values
    IniFile:=TIniFile.Create(trim(DMReport.MainRepName)+'.ini');

    IniFile.WriteInteger('LastInfo', 'MemGroup', MemGroup.ItemIndex);
    IniFile.WriteInteger('LastInfo', 'AgeGradeGrp', AgeGradeGrp.ItemIndex);
    IniFile.WriteBool('LastInfo', 'ChkPgNum', PageNumCByes.Checked);
    IniFile.WriteBool('LastInfo', 'ChkSepPg', SepPageCByes.Checked);
    IniFile.WriteBool('LastInfo', 'FonYes', FonCByes.Checked);
    IniFile.Free;
    IniFile:=nil;
  end;
end;

```

7. Click **Save File**.

8. Now we need to tell the program to read the INI file after the AfterLoadReport event.

Useful Information

It's important to place the code directly after the AfterLoadReport event because the object must already exist. That way, the program doesn't need to keep temporary variables around to hold the items stored in the INI file. It can directly load them into the screen objects.

On the Pascal Script File tab, enter the following code at the end of (but still within) the AfterLoadReport event:

```

AddBlank;
MemGroup.ItemIndex:=3; //Set 'Children & Adults' as default, counting 0-3.
AgeGradeGrp.ItemIndex:=2; //Set Member Age as default

// READ INI FILE set the initial values
try
  IniFile:=TIniFile.Create(trim(DMReport.MainRepName)+'.ini');

  MemGroup.ItemIndex:=IniFile.ReadInteger('LastInfo', 'MemGroup', 0);
  AgeGradeGrp.ItemIndex:=IniFile.ReadInteger('LastInfo', 'AgeGradeGrp', 0);
  PageNumCByes.Checked:=IniFile.ReadBool('LastInfo', 'ChkPgNum', true);
  SepPageCByes.Checked:=IniFile.ReadBool('LastInfo', 'ChkSepPg', true);
  FonCByes.Checked:=IniFile.ReadBool('LastInfo', 'FonYes', true);
  FieldsRead:=true;
finally
  IniFile.Free;
  IniFile:=nil;
end;
end;

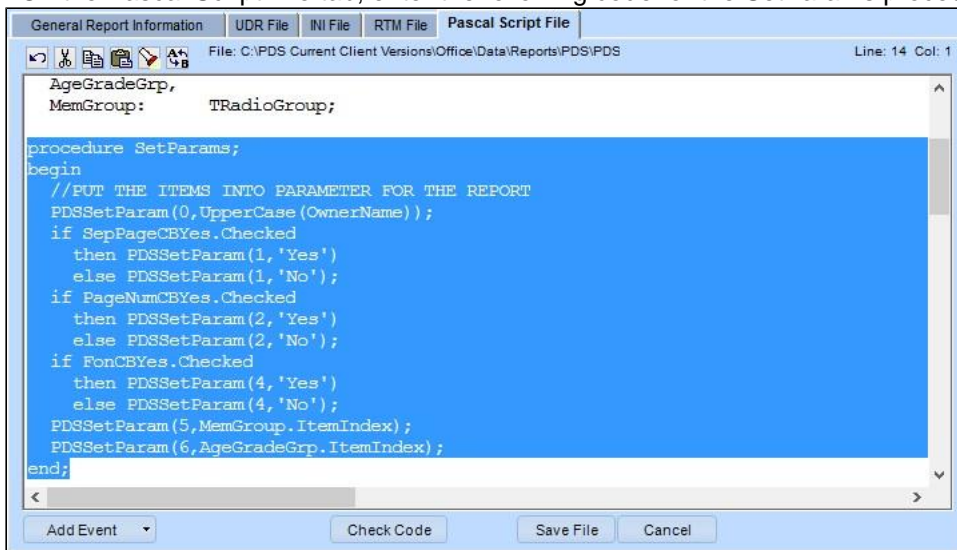
```

9. Click **Save File**.

Communicate with the Report

Now that we have a new window where a user can enter information, we must have a way to relay that information to the report.

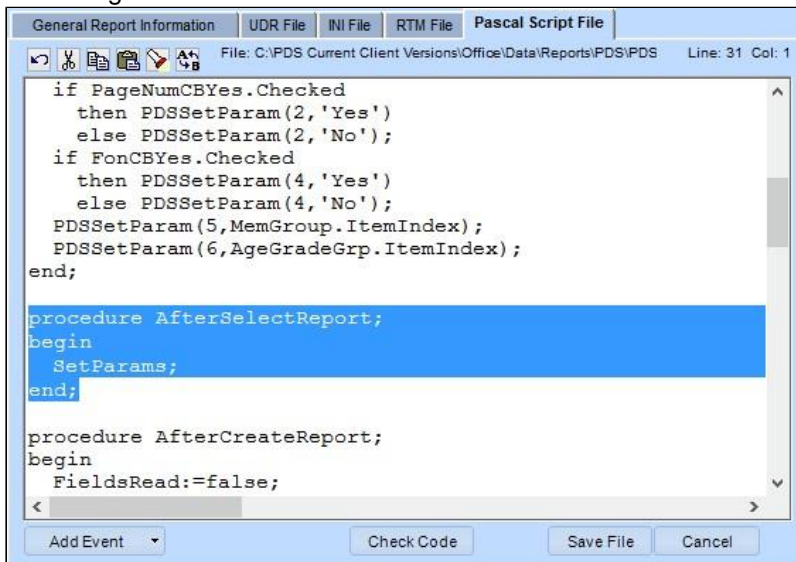
1. On the Pascal Script File tab, enter the following code for the SetParams procedure below the variables:



The screenshot shows a Pascal Script File editor window with the following code:

```
AgeGradeGrp,  
MemGroup: TRadioGroup;  
  
procedure SetParams;  
begin  
  //PUT THE ITEMS INTO PARAMETER FOR THE REPORT  
  PDSSetParam(0,UpperCase(OwnerName));  
  if SepPageCByes.Checked  
    then PDSSetParam(1,'Yes')  
    else PDSSetParam(1,'No');  
  if PageNumCByes.Checked  
    then PDSSetParam(2,'Yes')  
    else PDSSetParam(2,'No');  
  if FonCByes.Checked  
    then PDSSetParam(4,'Yes')  
    else PDSSetParam(4,'No');  
  PDSSetParam(5,MemGroup.ItemIndex);  
  PDSSetParam(6,AgeGradeGrp.ItemIndex);  
end;
```

2. Directly below the SetParams event, click **Add Report** and select **AfterSelectReport**. Enter the following code:



The screenshot shows a Pascal Script File editor window with the following code:

```
if PageNumCByes.Checked  
  then PDSSetParam(2,'Yes')  
  else PDSSetParam(2,'No');  
if FonCByes.Checked  
  then PDSSetParam(4,'Yes')  
  else PDSSetParam(4,'No');  
PDSSetParam(5,MemGroup.ItemIndex);  
PDSSetParam(6,AgeGradeGrp.ItemIndex);  
end;  
  
procedure AfterSelectReport;  
begin  
  SetParams;  
end;  
  
procedure AfterCreateReport;  
begin  
  FieldsRead:=false;
```

3. In the AfterLoadReport event, at the end of the try statement, add a call to the new SetParams routine:

```

// READ INI FILE set the initial values
try
  IniFile:=TIniFile.Create(trim(DMReport.MainRepName)+'.ini');

  MemGroup.ItemIndex:=IniFile.ReadInteger('LastInfo','MemGroup',0);
  AgeGradeGrp.ItemIndex:=IniFile.ReadInteger('LastInfo','AgeGradeGrp',0);
  PageNumCByes.Checked:=IniFile.ReadBool('LastInfo','ChkPgNum',true);
  SepPageCByes.Checked:=IniFile.ReadBool('LastInfo','ChkSepPg',true);
  FonCByes.Checked:=IniFile.ReadBool('LastInfo','FonYes',true);
  FieldsRead:=true;
  SetParams;
finally
  IniFile.Free;
  IniFile:=nil;
end;
end;

```

4. Click **Save File**, then **Close**.
5. Click **Next** until you reach the layout window, then click **Modify the Report**.
6. Right-click the PageNo object and select **Do Before Print**. Enter the following code, then click **Save**:

```

var
  s: string;
begin
  s:=PDSGetParam(2);
  if s='Yes'
  then SystemVariable1.Visible:=True
  else SystemVariable1.Visible:=False;
end;

```

7. Right-click the Fam Phone List object and select **Do Before Print**. Enter the following code, then click **Save**:

```

var
  s: string;
begin
  s:=PDSGetParam(4);
  if s='Yes'
  then DBMemo2.Visible:=True
  else DBMemo2.Visible:=False;
end;

```

8. At the bottom, click the **PDSSubReport1: Mem** tab.
9. Right-click the Mem Age object and select **Do Before Print**. Enter the following code, then click **Save**.

```

var
  i: integer;
begin
  i:=StrToInt(PDSGetParam(6));
  if i=2
    then DBText3.Visible:=True
    else DBText3.Visible:=False;
end;

```

10. Right-click the Mem Grade object and select **Do Before Print**. Enter the following code, then click **Save**.

```

var
  i: integer;
begin
  i:=StrToInt(PDSGetParam(6));
  if i=1
    then DBText4.Visible:=True
    else DBText4.Visible:=False;
end;

```

11. Right-click the Detail band and select **Do Before Print**. Enter the following code, then click **Save**.

```

var
  i: integer;
  i2: integer;
begin
  i:=StrToInt(PDSGetParam(5));
  i2:=Mem['Mem Type Number'];
  if i=0
    then Detail.Visible:=False
    else
      begin
        if ((i=1) and (i2=4)) or ((i=2) and (i2<>4))
          then Detail.Visible:=False
          else Detail.Visible:=True;
        end;
      end;
end;

```

After completing these steps, the highlighted options below on the Additional Layout window will work.

Page Layout Options:

Print names beginning with a new letter on a separate page

Print page numbers

Print phone numbers

Member Printing Options

Do Not Print Members

Print Only Adults

Print Only Children

Print Children & Adults

Member Information Options

Do Not Print Age or Grade

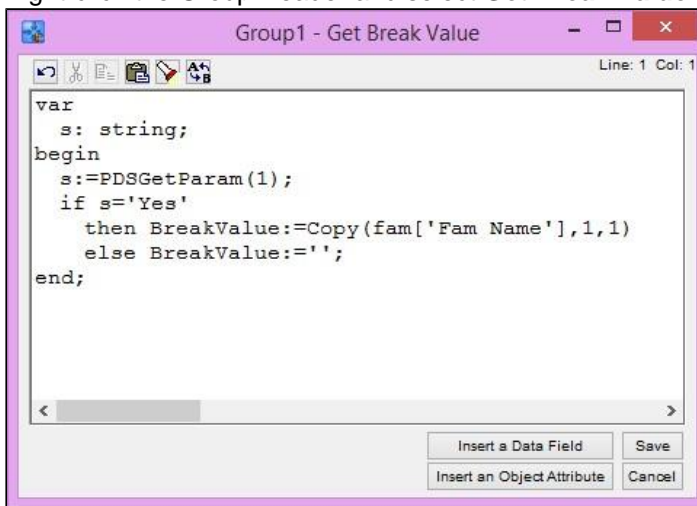
Print Member Grade

Print Member Age

Begin Each Letter on a New Page

To start each letter on a new page, we want to treat each letter as a group.

1. In the Advanced Report Writer, click the **Main: Fam** tab.
2. Click the **Report** menu and select **Groups**.
3. Select to break on **Custom Field**, and in the Groups drop-down list, select **Label1: Label1**.
4. Select **Start new page**, and click **OK**.
5. Right-click the Group Header and select **Get Break Value**. Enter the following code, then click **Save**:



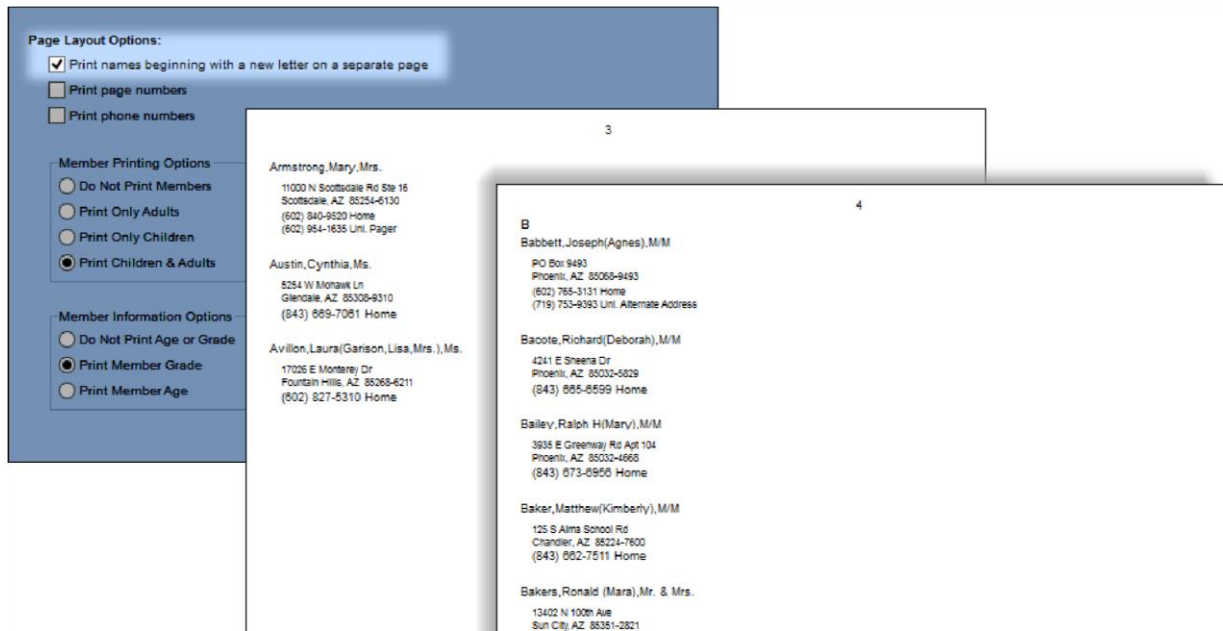
```

var
  s: string;
begin
  s:=PDSGetParam(1);
  if s='Yes'
    then BreakValue:=Copy(fam['Fam Name'],1,1)
    else BreakValue:='';
end;

```

6. Click **File > Save**, then **File > Close**.

After completing these steps, you can select to print new letters on a separate page.



Finished Product

Congratulations, you've completed the project! Not only do we have a complete Family Directory report, but now users have a few more options to customize it. Use the skills you learned to build and customize other reports.

Final code for the Pascal Script File

```
// NOTE: This does not include the code for the Do Before Print events for the objects

Uses
  scrnio;
var
  FieldsRead:          boolean;
  NewPage1:            TLMDAssistPage;
  PageLabel,
  CNLabel:             TLabel;
  FonCByes,
  SepPageCByes,
  PageNumCByes:       TCheckBox;
  AgeGradeGrp,
  MemGroup:           TRadioGroup;

procedure SetParams;
begin
  //PUT THE ITEMS INTO PARAMETER FOR THE REPORT
  PDSSetParam(0, UpperCase(OwnerName));
  if SepPageCByes.Checked
    then PDSSetParam(1, 'Yes')
    else PDSSetParam(1, 'No');
  if PageNumCByes.Checked
    then PDSSetParam(2, 'Yes')
    else PDSSetParam(2, 'No');
  if FonCByes.Checked
```

```

        then PDSSetParam(4,'Yes')
        else PDSSetParam(4,'No');
    PDSSetParam(5,MemGroup.ItemIndex);
    PDSSetParam(6,AgeGradeGrp.ItemIndex);
end;

procedure AfterSelectReport;
begin
    SetParams;
end;

procedure AfterCreateReport;
begin
    FieldsRead:=false;
end;

procedure BeforeDestroyReport;
var
    IniFile:TIniFile;
begin
    if FieldsRead then
        begin
            // ACCESS INI FILE save the values
            IniFile:=TIniFile.Create(trim(DMReport.MainRepName)+'.ini');
            IniFile.WriteInteger('LastInfo','MemGroup',MemGroup.ItemIndex);
            IniFile.WriteInteger('LastInfo','AgeGradeGrp',AgeGradeGrp.ItemIndex);
            IniFile.WriteBool('LastInfo','ChkPgNum',PageNumCByes.Checked);
            IniFile.WriteBool('LastInfo','ChkSepPg',SepPageCByes.Checked);
            IniFile.WriteBool('LastInfo','FonYes',FonCByes.Checked);
            IniFile.Free;
            IniFile:=nil;
        end;
end;

procedure AfterLoadReport;
begin
    NewScreenPage;
    AddHeading('Page Layout Options:',PageLabel);
    AddCheckBox('Print names beginning with a new letter '+' on a separate page',SepPageCByes);
    AddCheckBox('Print page numbers',PageNumCByes);
    AddCheckBox('Print phone numbers',FonCByes);
    AddBlank;
    AddRadioButtonGroup('Member Printing Options ',
        'Do Not Print Members, '+
        'Print Only Adults, '+
        'Print Only Children, '+
        'Print Children & Adults',MemGroup);
    AddBlank;
    AddRadioButtonGroup('Member Information Options ',
        'Do Not Print Age or Grade, '+
        'Print Member Grade, '+
        'Print Member Age',AgeGradeGrp);
    AddBlank;
    MemGroup.ItemIndex:=3; //Set 'Children & Adults' as default, counting 0-3.
    AgeGradeGrp.ItemIndex:=2; //Set Member Age as default
    // READ INI FILE set the initial values
    try
        IniFile:=TIniFile.Create(trim(DMReport.MainRepName)+'.ini');
        MemGroup.ItemIndex:=IniFile.ReadInteger('LastInfo','MemGroup',0);
        AgeGradeGrp.ItemIndex:=IniFile.ReadInteger('LastInfo','AgeGradeGrp',0);
        PageNumCByes.Checked:=IniFile.ReadBool('LastInfo','ChkPgNum',true);
        SepPageCByes.Checked:=IniFile.ReadBool('LastInfo','ChkSepPg',true);
        FonCByes.Checked:=IniFile.ReadBool('LastInfo','FonYes',true);
        FieldsRead:=true;
        SetParams;
    end;
end;

```

```
finally
  IniFile.Free;
  IniFile:=nil;
end;
end;
```