AppWizard

Developers Guide (v.4.02)

Last Updated: 17 May 2009

Table of Contents

1 The Template

1.1 Introduction

The Template is used to specify the layout of your AppWizard application. Within the template you specify where to find the database, where on your page the menu and content of your applications should be placed, and what the default formatting should be for all the functions used by the applications.

An application is merely a collection of functions performed in a specific sequence. The way these functions will be displayed on the site is determined by the settings within the template.

Some settings are compulsory and must always be present in a template, but most settings have a default value within AppWizard. You only need to specify these settings if you do not like the default settings.

1.2 Basic Template

The following code is the basic template (which is shipped with AppWizard). You can use this template as it is. The default database settings will work if you set up the ODBC settings as indicated in the installation guide. If you have moved the database or choose to use a different database, you must update the database setting in the template for your site to work.

Note that an ITEM tag must be completed on a single line. The | | characters below are only used to indicate that the line was broken due to space limitations and should not be included when creating the template.

```
<html><head><title>AppWizard</title></head>
<ITEM Database.ConnectionString=Provider=MSDASQL.1;Password=sa; || Persist
Security Info=True;User ID=sa;Data Source=AppWizard; ||
Initial Catalog=AppWizard>
<body bgcolor="white" link="#000000" alink="#000000" vlink="#000000"
<ITEM Body.Onload>
>
<table border="0" width="100%"><tr valign="bottom"><td><font size="+2"||
color="blue"><B><I>AppWizard</i></b></td>
<ITEM MenuItem.Start>
<td align="center">
<ITEM MenuItem.Text>
</td>
<ITEM MenuItem.End>
</tr></table>
<ITEM DropDown.BorderColor=black>
<ITEM DropDown.BorderSize=2px>
<ITEM DropDown.Margin=2px>
<ITEM DropDown.Color=white>
<ITEM Format.Folder=#e0e0e0>
<ITEM Format.TableColor=#e0e0e0>
<ITEM SubMenu.Indicator=...>
```

```
<ITEM Navigate.Text><B><ITEM Navigate.AppTitle></b>
<hr>
<ITEM Content.Text>
</body>
</html>
```

1.3 Including Settings in the Template

The template is a standard HTML page with extra <ITEM …> tags included. These Item tags hold the settings used by AppWizard.

You can add these tags anywhere in the template. AppWizard reads through the template and removes all the ITEM tags before returning the resulting screen. Some tags will give settings, while other tags will show AppWizard where certain items must be displayed (e.g. the Menu, Navigation bar and Content text).

The reference **(1.4)** will show you all the settings you can use and what their effect would be on your applications.

To see an example of how the Item tags are implemented, study the basic template given above.

IMPORTANT: Take note that all item tags are Case Sensitive. The tag ContentString is NOT the same as Contentstring.

An ITEM tag must be completed on a single line. Do not have line feeds or carriage return characters before the closing > character.

The format for all Item tags is as follows:
<ITEM Parameter.Variable=Value>
or
<ITEM Parameter.Variable>

The Parameter.Variable combination determines what setting will be addressed. The Value can be any text, excluding non-visible characters and the > closing sign. If you need to specify the >, use the > reference.

In some cases a Value is not required, in which case no value needs to be specified. If a value is specified for these tags, it will be ignored.

1.4 ITEM Tag Reference

1.4.1 Body.Onload

This parameter is necessary to set focus on an edit box when a page is opened. Note that this tag must be specified within the <body> tag of the html.

1.4.2 Content.Text

This parameter indicates where the application content must be displayed. No value is required for this parameter.

1.4.3 Database.ConnectionString

Use this parameter to specify the connection to your database. The default connection string is provided in the basic template. Note that you have to specify the ODBC name, the username, the password and the default catalogue (Database) used by AppWizard in the connection string.

- ***Data Source:*** This indicates the data source name that was specified in the ODBC setup during the installation of AppWizard.
- ***Provider:*** This must be set to **MSDASQL.1** to specify that the source database is a Microsoft SQL database.
- ***User ID:*** This is the SQL user which should be used to access the database. IN most cases this would be the **sa** user.
- ***Password:*** This is the password specified for the user given above. If you have followed a standard AppWizard installation, the sa user password will be **sa**.
- ***Persist Security Info:*** This indicates that the security info must be used for the duration of the session, and not prompted again. The value must be set to **True**.
- ***Initial Catalog:*** This indicates the default database which must be used for AppWizard. If a normal installation of AppWizard was done, the database name will be **AppWizard**.

1.4.4 DropDown

The drop down item indicates how the dropdown menus must be formatted.

- **DropDown.BorderColor** sets the color of the dropdown menu's border.
- **DropDown.BorderSize** sets the size (width) of the border.
- **DropDown.Color** sets the background color of the dropdown menu.
- **DropDown.Margin** sets the distance between the menu text and the dropdown menu borders.

1.4.5 Format

The format item allows you to set the default color of items displayed in AppWizard.

- **Format.Folder** gives the default background color of folder titles.
- **Format.TableColor** gives the default background color of table headings.
- **Format.TableBorderColor** gives the default border color of tables.
- **Format.TableRowColorEven** gives the default color for even numbered rows for interlaced table row colors.
- **Format.TableRowColorOdd** gives the default color for odd numbered rows for interlaced table row colors.

1.4.6 Link.Home

Use this item to place the source value of a link pointing to a home page. This item will typically be placed inside a ***<a href="*** tag.

1.4.7 MenuItem

The menu item indicates how your main menu will be displayed. The MenuItem parameter has 3 variables that must always be used together: Start, Text, and End. Each menu item displayed on your main menu will use the formatting given in the MenuItem parameter.

A menu item starts by giving the **MenuItem.Start** setting. The text of the menu item (as given by AppWizard) will be displayed where the **MenuItem.Text** setting is given. The menu item ends where the **MenuItem.End** setting is given. None of these settings has a value.

The HTML code between the Start and Text variables will be repeated before every menu item and the HTML code between the Text and End variables will be repeated after every menu item.

1.4.8 Navigate

Within AppWizard, the navigation back to previous applications is done using a navigation bar. The navigation bar is displayed in two parts. The first part provides links to the previous applications in the navigation, and the second part shows the title of the current application. The links to the previous applications is placed where the **Navigate.Text** item is found, and the application title is placed where the **Navigate.AppTitle** is found. Use the **Navigate.Separator** item to specify the separator to use between application links.

1.4.9 Shortcut

The **Shortcut** item indicates how a shortcut menu should be displayed. A shortcut menu displays a submenu as part of the screen, which doesn't disappear when a user moves through applications. The shortcut menu will only change or disappear when a new shortcut menu is selected. The shortcut area starts by using the **Shortcut.Title** item to indicate where the shortcut menu title must be displayed. Each menu item will repeat the code from the **Shortcut.Start** item, place the shortcut menu text where the **Shortcut.Text** item indicates, and end where the **Shortcut.End** item is reached.

1.4.10 SubMenu.Indicator

On the menu, when the menu item opens a submenu rather than an application, the menu item is followed by the characters specified by the **SubMenu.Indicator** string.

1.4.11 UserName

The current logged in user's name will be displayed where this item is found. This is typically shown as part f the page header and assists developers and support staff to identify which user is logged in.

2 Application Development

2.1 Introduction

Applications are developed by using a series of applications, each consisting of a sequence of functions, each with its own properties.

The AppBuilder application under the Utilities... menu can be used to develop the applications. AppBuilder is an application developed using AppWizard, so all the functionality you see within AppBuilder is also available for you to use. You can edit the AppBuilder applications if you really want to, but it is not recommended. (All the AppBuilder applications start with **dev**.)

Note that the AppBuilder application will also be updated in future upgrades, so any changes that you make to these applications might be overwritten.

2.2 Application Structure

If you would like to build an application, you will need to break it into logical applications. Each application will perform a set of actions, which can result in a web page being displayed, a series of database updates, or both.

The applications do not work on their own, but rely on a database structure to hold the required information, and sometimes on stored procedures to perform complex operations on the database. Take note that stored procedures will almost always perform better than direct database operations from AppWizard because the database optimizes their performance and all their operations are performed on the database itself.

The applications communicate with each other using parameters. You need to pass parameters from one application to the next to ensure that the next application knows what to do.

Pages can contain almost any HTML code, including forms, tables, and images. Forms will pass the user input as parameters to the next application.

2.3 Passing Parameters

Follow this example to see how to use parameters from one application to the next one, to ensure the proper flow between applications. This example holds a list of music titles. The user can add a new title and edit an existing title. (For a comprehensive example, see section 3).

It is assumed that there is a table on the database to hold the list of titles.

The first step is to break the program down into a list of logical applications.

- The **List** application will display a form which can be used to add a new title (which passes the user input to the **Add** application), and a list of existing titles with links to the **Edit** application (which passes the selected title identifier to the **Edit** application).

- The **Add** application reads the title parameter received from the **List** application, and adds the title to the database. Thereafter it redirects back to the **List** application without any parameters.
- The **Edit** application reads the title identifier passed from the **List** application and selects the specified title from the database. The selected title is displayed in a form allowing the user to edit it. The updated title is passed to the **Update** application.
- The **Update** application saves the updated title to the database. Thereafter it redirects back to the **List** application without any parameters.

3 Sample: Petty Cash

3.1 Application Definition

The Petty Cash application will need to track the issuing of petty cash vouchers and the reconciling of these vouchers once completed. The normal procedure for doing this is given below.

Firstly, the employee requests petty cash and is issued with the requested amount. The employee number and name is required as well as the purpose for the money. A petty cash voucher with a unique sequence number must be issued with the money.

Secondly, the employee returns the voucher, along with the change received and the applicable cash slip(s). The actual amount spent must be entered, indicating the amount paid in VAT (Value Added Tax). As a result the updated voucher must be displayed indicating the amount of change that should have been received.

Thirdly, the amount of change is confirmed and the voucher is finalized.

Finally, two reports must be available with one indicating a list of outstanding vouchers, and the other indicating the amount of money issued on completed vouchers.

3.2 Database Structure

The vouchers must be kept in a table in the database. For this application we will only need a single table. Note that other applications might require much more complex relational database structures.

We will create the following database structure on the AppWizard database.

Table Name: PettyCash

Field	Type	Special conditions
VoucherId	int	identity field
EmployeeCode	varchar(20)	
EmployeeName	varchar(100)	
Reason	varchar(200)	
InitialAmount	decimal(18,2)	
FinalAmount	decimal(18,2)	
FinalVAT	decimal(18,2)	
Status	int	

Note for MSDE users

If you are using MSDE, you will need to create the table using a Create Table query using the command line **osql** application. Go to Start, click on Run…, and type in cmd. Use the command line **osql -U sa -P sa**. (Note that there is no spaces between the - and U, and the - and P).

When inside the **osql** editor, type the following eleven lines of code and then close the command line window:

```
Use AppWizard
Create Table PettyCash (
VoucherId int identity(1,1),
EmployeeCode varchar(20),
EmployeeName varchar(100),
Reason varchar(200),
InitialAmount decimal(18,2),
FinalAmount decimal(18,2),
FinalVAT decimal(18,2),
Status int)
Go
```

3.3 Application Structure

The petty cash application will consist of various sub-applications. In order to group these sub-applications together, all of their names will be prefixed with pcash (in the same way all the sub-applications for the AppBuilder application starts with dev).

3.3.1 pcashVoucher sub-application

This sub-application displays a form for the user to enter the initial voucher details. These details will be passed to the pcashVoucherCreate sub-application to insert the details into the PettyCash table on the database.

Use AppBuilder to build the sub-applications. Set the filter to pcash as this will show only the pcash applications. Delete the - before entering pcash.

Add a new application: pcashVoucher

If the filter was set correctly, you should see the new application listed after adding it.

Click on the application name to edit its properties, and then click on the folder Update Application Properties.

Specify an application title (e.g. New Voucher) - which is what will be displayed on the navigation bar just underneath the logo, and give a brief description - which you will see in the application list. Update the properties. You should now be back at the application list, with your description being displayed. Click on the application name again to add the functions, and then click on Add a new Function.

Add the functions as described in the table below:

SeqNo	**Function**	**Other Properties**
10	Html	**HTML Code:** <B>Enter the new voucher details</B>
20	Form	**Application:** pcashVoucherCreate

30	EditBox	**Caption:** Employee Code **Width:** 10 **Name:** Code **1 for Focus:** 1
40	EditBox	**Caption:** Employee Name **Width:** 30 **Name:** Name
	EditBox	**Caption:** Reason **Width:** 70 **Name:** Reason
60	EditBox	**Caption:** Initial Amount **Width:** 15 **Name:** Amount **Fixed Value:** 0.00
70	Button	**Button Text:** Create Voucher
80	FormClose	

What we have done now, is to create a form which requests the employee code and name, as well as the reason for the petty cash and the initial amount. The values entered into the form will be passed to the pcashVoucherCreate application.

3.3.2 Adding the application to the Menu

To test the application, we need to add it to the menu. Use the navigation bar (which should be underneath the AppWizard 2 logo if you are using the standard template) to return to the AppBuilder application. You will notice that the last filter you specified is still in tact. If you used the menu to open the AppBuilder application again, you will have the - filter.

Change the filter to be awMenu. This will show you only the menu application. This is a special application which determines your menu structure. Any applications which should be added to the menu must be added in this application. Security access to applications will also be set in this application, but we will look at that later.

Open the menu application to see its functions. After studying it for a while, you will notice that the first menu items are the one's displayed at the top menu until the MenuMainClose function is reached. Initially this will only be the Programs... and Utilities... sub menus, as well as the Help menu item.

Further down you will notice that the sub menus is build by starting with a Menu function and ending with a MenuClose function. Whatever is between these two functions will be displayed on the sub menu.

Add a new function as follows:
-- We will place the Petty Cash applications in the Programs... sub menu
Sequence Number: 110
Function: MenuItem

Display Text: New Petty Cash Voucher
Application: pcashVoucher

After adding the line, you should see the menu item in the Programs… sub menu. Click on it to see your first application. If you fill in the form and click on the button, you will receive an error message stating that the application pcashVoucherCreate cannot be found. This is because we have not created it yet.

3.3.3 pcashVoucherCreate sub-application

Open the AppBuilder application again (you will have to use the menu as the navigation has been broken when the new application was loaded). Change the filter to pcash again to see your application.

Now add the pcashVoucherCreate application with a title of Created and a brief description. Add the functions given in the table below:

SeqNo	Function	Other Properties
10	DataAdd	**Table Name:** PettyCash
20	DataSetValue	**Field Name:** EmployeeCode **Parameter:** Code
30	DataSetValue	**Field Name:** EmployeeName **Parameter:** Name
40	DataSetValue	**Field Name:** Reason **Parameter:** Reason
50	DataSetValue	**Field Name:** InitialAmount **Parameter:** Amount
60	DataSetValue	**Field Name:** Status **Fixed Value:** 0
70	DataCommit	
80	SQL	**SQL Statement:** select max(VoucherId) as VoucherId from PettyCash
90	Html	**HTML Code:** <B>Voucher Number [Space after last word] **Query Field:** VoucherId
100	Html	**HTML Code:** was created for [Space before first word and after last word] **Parameter:** Name
110	Html	**HTML Code:** to the amount of R[Space before first word] **Parameter:** Amount
120	Html	**HTML Code:** </B>
130	DataClose	

This application adds a new record to the PettyCash table. It places the values into the data fields which were passed as parameters from the previous application. After adding the values to the newly created record, the record is saved to the database using the DataCommit function.

The VoucherId field is an identity field, which means it will automatically increment its value for every record added. We use this knowledge to get the newly created VoucherId by querying the maximum number in the table.

Finally, we build up an Html string by placing text, with either the queried data field holding the new VoucherId or the parameters passed from the previous application.

We do not need to place this application on the menu, as it will be called by the pcashVoucher application.

Note that it is always good practice to close a dataset after completing the application as it will improve the performance of your applications. If you do not close it, AppWizard will close it eventually, but the database session might remain in memory for a while before it is closed.

Run the New Petty Cash Voucher application again and fill in the form. This time, after submitting the form, you should receive a message indicating that the new voucher was created. Adding more vouchers will give you the next voucher number in the sequence.

3.3.4 pcashReturn sub-application

This application will allow the user to select the applicable petty cash voucher which the employee has returned along with the change and cash slip(s) of the transaction.

This application will need to be run from the menu again, and will read all the vouchers in the table which has a status of 0.

The selected voucher id will be passed to the pcashReturnEdit application.

Create the application pcashReturn with a title of Voucher Returns and an applicable description.

The application is given in the table below:

SeqNo	Function	Other Properties
10	Html	**HTML Code:** <B>Select a voucher to edit</B>
20	SQL	**SQL Statement:** select * from PettyCash **Where name:** Status **Fixed Value:** 0 **Order By:** EmployeeName
30	Table	
40	TableField	**Display Field:** VoucherId **Application:** pcashReturnEdit **Name:** VoucherId **Parameter Field:** VoucherId **Caption:** #
50	TableField	**Display Field:** EmployeeName

		Caption: Employee
60	TableShow	
70	DataClose	

Add this application to your awMenu application just underneath the New Petty Cash Voucher with Display Text as Petty Cash Returns and Application as pcashReturn.

Run the application and see what it looks like. Note that the voucher number is a link to the next application. If you click on a link it will again tell you that the destination application is missing.

3.3.5 pcashReturnEdit sub-application

This application will use the VoucherId that was just passed to it as a parameter, and show the full details of that voucher. It will also show a form in which the user can enter the final amount and VAT amount of the transaction. Just for fun we will allow the user to edit the reason.

After completion this application will pass the required parameters to the pcashReturnChange sub-application for processing.

Note that the VoucherId parameter does not automatically get passed to the next application, so we need to specify it as part of our form. A space for a single parameter is provided in the Form function. If you would like to pass more than one parameter, you should use the Hidden function for the rest.

Add the new application pcashReturnEdit with a title of Editor and an applicable description.

The table below gives the list of functions:

SeqNo	**Function**	**Other Properties**
10	Html	**HTML Code:** <B>Edit the selected Voucher</B>
20	SQL	**SQL Statement:** select * from PettyCash **Where name:** VoucherId **Parameter:** VoucherId
30	Form	**Application:** pcashReturnChange **Name:** VoucherId **Parameter:** VoucherId
40	Html	**Query Field:** EmployeeCode **HTML Code:** <B>Employee Code: </B>
50	Html	**Query Field:** EmployeeName **HTML Code:** <B>Employee Name: </B>
60	Html	**Query Field:** InitialAmount **HTML Code:** <B>Initial Amount: </B>R
70	EditBox	**Caption:** Reason **Width:** 70 **Name:** Reason **Query Field:** Reason

		1 for Focus: 1
80	EditBox	**Caption:** Final Amount **Width:** 15 **Name:** FinalAmount **Query Field:** FinalAmount
90	EditBox	**Caption:** Final VAT **Width:** 15 **Name:** FinalVAT **Query Field:** FinalVAT
100	Button	**Button Text:** Edit Voucher
110	FormClose	
120	DataClose	

3.3.6 pcashReturnChange sub-application

This application will again use the VoucherId that was just passed to it as a parameter, and show the updated details for the voucher after the Reason, Initial Amount and Final VAT have been updated. Note that the fields will be displayed in a table format. It will also show the amount of change to be received.

We will provide two buttons to confirm that the amount is of change is correct, Yes and No. We will pass the required parameters to the pcashReturnConfirm sub-application for processing.

Add the new application pcashReturnChange with a title of Change and an applicable description.

The table below gives the list of functions:

SeqNo	**Function**	**Other Properties**
10	DataUpdate	**Table Name:** PettyCash **Key Field:** VoucherId **Parameter:** VoucherId
20	DataSetValue	**Field Name:** Reason **Parameter:** Reason
30	DataSetValue	**Field Name:** InitialAmount **Parameter:** InitialAmount
40	DataSetValue	**Field Name:** FinalVAT **Parameter:** FinalVAT
50	DataCommit	
60	Html	**HTML Code:** <B>Updated Voucher</B><P>
70	SQL	**SQL Statement:** select * from PettyCash **Where name:** VoucherId **Parameter:** VoucherId
80	Html	**Query Field:** EmployeeCode **HTML Code:** <table border="0" cellspacing="0"><tr><td><B>Employee Code: </B>

90	Html	**Query Field:** EmployeeName **HTML Code:** </td></tr><tr><td><B>Employee Name: </B>
100	Html	**Query Field:** Reason **HTML Code:** </td></tr><tr><td><B>Reason: </B>
110	Html	**Query Field:** InitialAmount **HTML Code:** </td></tr><tr><td><B>Initial Amount: </B>R
120	Html	**Query Field:** FinalAmount **HTML Code:** </td></tr><tr><td><B> FinalAmount: </B>R
130	Html	**Query Field:** FinalVAT **HTML Code:** </td></tr><tr><td><B> FinalVAT: </B>R
140	Html	**HTML Code:** </td></tr>
150	DataClose	
160	SQL	**SQL Statement:** select InitialAmount-FinalAmount as Change from PettyCash **Where name:** VoucherId **Parameter:** VoucherId
170	Form	**Application:** pcashReturnConfirm **Name:** VoucherId **Parameter:** VoucherId
180	Html	**Query Field:** Change **HTML Code:** <P><B>Change to be received: </B>R
190	Html	**HTML Code:** </table><P><B>I confirm that the amount of change is correct</B>
200	Button	**Button Text:** Yes **Name:** Yes
210	Button	**Button Text:** No **Name:** No
220	FormClose	
230	DataClose	

3.3.7 pcashReturnConfirm sub-application

This application will check whether the user clicked the Yes or No button with the help of IfGoto functions. If the user selected the Yes button the Status of the voucher will be set to 1, which means that the Change received was confirmed. If the user selected the No button the pcashReturnEdit sub-application will be displayed again for the user to edit the Final Amount and/or Final VAT.

The pcashReturnConfirm sub-application will not be seen by the user, but the sub-application redirected too.

Add the new application pcashReturnConfirm with a title of Change and an applicable description. Set Nav Back as 3 – transfer must go back through pcashReturnConfirm, pcashReturnChange and pcashReturnEdit. This is so that the navigation at the top of the screen will not be duplicated.

The table below gives the list of functions:

SeqNo	Function	Other Properties
10	IfGoto	**Test Value:** Yes **Condition:** = **GotoNo:** 40 **Parameter:** Yes
20	IfGoto	**Test Value:** No **Condition:** = **GotoNo:** 30 **Parameter:** No
30	Redirect	**Application:** pcashReturnEdit **Name:** VoucherId **Parameter:** VoucherId
40	DataUpdate	**Table Name:** PettyCash **Key Field:** VoucherId **Parameter:** VoucherId
50	DataSetValue	**Field Name:** Status **Fixed Value:** 1
60	DataCommit	
70	Redirect	**Application:** pcashReturnFilter

3.3.8 pcashReturnFilter sub-application

This application is only for the redirection to the pcashReturn sub-application if the user selected the Yes button to confirm the Change. We need to put this application in because the Nav Back to the two different applications from pcashReturnConfirm is not the same. The Nav Back of pcashReturnFilter must be 2. Because pcashReturnConfirm already went back 3, pcashReturnFilter must just go back through itself and pcashReturnEdit.

Add the new application pcashReturnFilter with a title of Filter and an applicable description. Set Nav Back as 2.

The table below gives the list of functions:

10	Redirect	**Application:** pcashReturn

3.3.9 pcashReportOutstanding sub-application

This application will need to be run from the menu again, and will display a list of all the outstanding vouchers. We will display the fields in a table.

Add the new application pcashReportOutstanding with a title of Outstanding Vouchers and an applicable description.

The table below gives the list of functions:

SeqNo	Function	Other Properties
10	SQL	**SQL Statement:** select * from PettyCash **Where name:** Status **Fixed Value:** 0 **Order By:** VoucherId
20	Table	**Border Type:** 1
30	TableField	**Display Field:** EmployeeCode **Caption:** Employee Code **Alignment:** Center [note the spelling of the word Center]
40	TableField	**Display Field:** EmployeeName **Caption:** Name
50	TableField	**Display Field:** Reason **Caption:** Reason
60	TableField	**Display Field:** InitialAmount **1=Number Format/2=Integer:** 1 **Caption:** Initial Amount **Alignment:** Right
70	TableShow	
80	DataClose	

3.3.10 pcashReportIssued sub-application

This application will need to be run from the menu again, and will display the total amount of money issued on completed vouchers. Again we will display the fields in a table. Note that we enter SUM in the GROUP or Summary Type field. This is because we want the total of all the fields at the end of the table. We also set the Colspan of Totals field to 3. This is because there is a total of four columns and we only want to display the last column's total.

Add the new application pcashReportIssued with a title of Amount Issued On Completed Vouchers and an applicable description.

The table below gives the list of functions:

SeqNo	Function	Other Properties
10	SQL	**SQL Statement:** select * from PettyCash **Where name:** Status **Fixed Value:** 1 **Order By:** VoucherId
20	Table	**Border Type:** 1 **1 to Show Totals:** 1 **Colspan of Totals:** 3 **Text of Totals:** <B>Total Amount Issued On Completed Vouchers</B>
30	TableField	**Display Field:** EmployeeCode **Caption:** Employee Code

		Alignment: Center [note the spelling of the word Center]
40	TableField	**Display Field:** EmployeeName **Caption:** Employee
50	TableField	**Display Field:** Reason **Caption:** Reason
60	TableField	**Display Field:** FinalAmount **1=Number Format/2=Integer:** 1 **GROUP or Summary Type:** SUM **Caption:** Final Amount **Alignment:** Right
70	TableShow	
80	DataClose	

4 Function Reference

4.1 – NEW FUNCTION –

When a new function is started, this is the default function used. AppWizard will ignore this function when it is found. This function can also be used as a placeholder to 'disable' a function during a time of debugging.

Because all the parameters for this function will be ignored, they can be used to place comments within the applications where needed.

4.2 Button

This function creates a Submit button on a form and is used between the ***Form*** and ***FormClose*** functions.

Fields used:
Button Text: Specifies the text on the button. If the button has a name, this text will be the value of a parameter with this name.
Name: Specifies the Name of the button. This is optional. If a name is given, the text of the button will be returned as the value for the parameter with the specified name.

4.3 CheckBox

This function displays a check box on a form and is used between the ***Form*** and ***FormClose*** functions. The checkbox returns a 1 if it is selected and a 0 if it is not selected.

Fields used:
Caption: The Caption to be displayed next to the check box.
Name: The name of the parameter which will contain the value of the check box.
Parameter, Query Field, Fixed Value: These fields give the value to be used as the initial value of the check box. See the ***GetValue()*** function for further details.

4.4 ComboBox

This function shows a drop down box in a form and is used between the ***Form*** and ***FormClose*** functions. The entries in the drop down list can be loaded using a SQL statement, values, or an SQL statement built using the ***ComboSQL*** function.

Using an SQL statement, the "Text" field returned by the SQL statement is used as the caption for the item and the "Value" field is used to set the value of the item, e.g. select *ColomnA* as Text, *ColumnB* as Value from *TableC*. ("Text" and "Value" are aliases for the caption and the value of the item).

If the source is set for Values, the value list in the *Value List* field is used instead to populate the combo box. The value list is a comma delimited list containing a sequence of Text and Value entries. As an example **Jan,1,Feb,2,Mar,3** will produce a combo box containing 3 entries displaying the captions Jan, Feb and Mar, with a resulting value of 1,2 and 3 respectively.

If the source is set to use the SQL prepared in the ***ComboSQL*** function, the internal SQL statement and value list will be ignored.

Fields used:
Caption: The Caption to be displayed next to the combo box.
Source: This field specifies how the combo box will be populated, either the an SQL statement, a list of values, or from the ***ComboSQL*** function.
Name: The name of the parameter which will contain the Value (from the SQL statement) returned by the combo box.
2-10 for Multiple Selection: Specifying a value from 2 to 10 in this field will produce a list box with the specified number of rows displayed, rather than a combo box. The user can select multiple options from the list by holding down either the Ctrl or Shift buttons while clicking. The selected values will be returned in a comma delimited list.
Parameter, Query Field, Fixed Value: These fields give the value to be used as the initial selected value of the combo box. See the ***GetValue()*** function for further details.
SQL Statement: This field contains the SQL statement that must return a "Text" field and a "Value" field. The items in the combo box will be displayed in the order in which the query returns it.
Value List: This field contains the value list (Text1,Value1,Text2,Value2,…) to be used instead of the SQL statement if a 1 is specified in *1 for Values*. (This is when the user wants, for example, a list in which to choose from Jan, Feb and Mar, each with a resulting value of 1,2 and 3 respectively, or from the options Yes, No, and Uncertain. In this instance *Text1* might be *Yes* and *Value1* might be *1*, *Text2* might be *No* and *Value2* might be *-1*, and *Text3* might be Uncertain and *Value3* might be *0*). A number from 2 to 9 allows multiple selections in the combo box and displays the number of lines specified at a time.
Top Value: This field contains the value to be returned if the text in *Top Text* is specified. This is the first value to be used irrespective of the other values in the list. This field can be left blank when a top text is specified, which will return a blank value if selected.
Top Text: This field contains the first item to be displayed in the list, if specified.

4.5 ComboSQL

This function builds a database query to be used within the next ComboBox function called. An SQL statement is generated as follows: *SQL Statement* **where** *Where name= Parameter / Query Field / Fixed Value* **order by** *Order By*. If *Where name* has no value, the "where"-clause will not be included. If *Order By* has no value the "order by"-clause will not be included. The "where" condition is the logic used to compare the field with the parameter.

Fields used:

Where name: This is a parameter that will be used to generate the "where"-clause of the query. If the SQL statement starts with an EXEC to run a stored procedure, the word *where* will not be included in the final query.
Parameter, Query Field, Fixed Value: These fields give the value of the parameter. See the ***GetValue()*** function for further details.
SQL Statement: The SQL query statement
Order By: This value will be used to generate an "order by"-clause.

4.6 Comment

This function is ignored by AppWizard and is purely used by the developers to show a programming comment.

Fields used:
Comment: Place your comment in this field.

4.7 CSVExport

This function produces a .CSV comma delimited file with the content specified by the ***CSVField*** function, into a file specified by the ***CSVFile*** function.

Fields used:
Column Headings: Indicates if the first row exported must contain field names as specified by the ***CSVField*** function.

4.8 CSVField

This function specifies the fields that must be exported or imported using the ***CSVExport*** or ***CSVImport*** functions, after the file was specified by the ***CSVFile*** function.

Fields used:
Heading: Provides a heading name for the field. This is used in the ***CSVExport*** function when a heading row is required.
Format: Indicates if the general format or string format must be used during import or export. The string format expects the field value to be enclosed by double quotes (" ").
@Name for Import: This is the stored procedure parameter name, preceded by the @ character, which must process the field when it is imported.
Parameter, Query Field, Fixed Value: These fields give the value to be exported. See the ***GetValue()*** function for further details.

4.9 CSVFile

This function specifies the file to be imported or exported using the ***CSVExport*** or ***CSVImport*** functions. After this function is called, the ***CSVField*** function must be called for each field that must be imported or exported.

Fields used:
Parameter, Query Field, Fixed Value: These fields give the filename of the file. See the ***GetValue()*** function for further details.

4.10 CSVImport

This function imports values from a .CSV comma delimited file specified using the ***CSVFile*** functions. The fields imported must be specified by the ***CSVField*** function for each field to be imported.

A store procedure receives a call for every record being imported to perform the required action actions with the record.

Fields used:
Headings: This field indicates if the file contains a heading row. The heading row is ignored if it exists.
Stored Proc Name: This field specifies the stored procedure which must process each imported record.

4.11 DataAdd

This function inserts a blank record into a table in the database, allowing the ***DataSetValue*** function to update the values of the fields in this table. Use the ***DataCommit*** function to post the changes to the table.

Fields used:
Table Name: Specifies the full table name of the table into which the record will be added.

4.12 DataClose

This function is used to close a query opened using the ***SQL*** function. It must be called before a new query is opened. No fields are used.

4.13 DataCommit

This function will save any changes made to a table using the ***DataSetValue*** function to the database. The table is closed for further updating. If further changes are to be made to the table it must be opened again using the ***DataUpdate*** or ***DataAdd*** functions. No fields are used.

4.14 DataDelete

This function deletes the records in a table where the key field is the given value.

Fields used:

Table Name: Specifies the full table name of the table with the records to be deleted.
Key Field: Specifies the name of the key field in the table. This key field will be used to determine which record(s) in the table should be deleted.
Parameter, Query Field, Fixed Value: These fields give the value of the key field. See the ***GetValue()*** function for further details.

4.15 DataNoRecords

This function performs a specified action if the stored procedure or SQL called immediately before this function returns no records. It can perform one of three actions:

- **Continue** will display the HTML message and continue with the next function.
- **Stop** will display the HTML message and then stop the application
- **Goto** will display the HTML message and then continue processing from the specified sequence number.

Fields used:
Action: This field specifies if the function should continue, stop, or go to a specified function.
Goto SeqNo: This field specifies the sequence number to go to if the Goto action is selected.
Parameter, Query Field, Fixed Value: These fields give the value that will be displayed after the HTML code. See the ***GetValue()*** function for further details.
HTML Code: This field contains some HTML formatted text to be displayed.

4.16 DataSetValue

This function alters the value of a field in the table opened by using either the ***DataAdd*** or ***DataUpdate*** functions. After all the changes were made to the fields, the ***DataCommit*** function must be called to post the changes to the database. When you use DataSetValue to update a date, it will be in the format yyyy/mm/dd. If you want the date to be captured in another format, you must make use of a stored procedure to apply the format.

Fields used:
Field Name: Specifies the field name to be updated. If this field does not exist within the table, the application will give an error.
Parameter, Query Field, Fixed Value: These fields give the value to be entered into the field. See the ***GetValue()*** function for more details.

4.17 DataUpdate

This function opens a specific record in a table to be updated using the ***DataSetValue*** function. After the changes were made to the fields, the ***DataCommit*** function must be used to post the changes to the database.

Fields used:
Table Name: Specifies the full table name of the table to open.
Key Field: Specifies the name of the key field in the table. This key field will be used to determine which record in the table should be opened.

Parameter, Query Field, Fixed Value: These fields give the value of the key field. See the ***GetValue()*** function for further details.

4.18 DynamicList

This function displays a drop down box, similar to the ***ComboBox*** function, which dynamically changes selected values on the page, based on the selected value in the dropdown box. The values that can be dynamically changed are the captions of all form functions, e.g. ***EditBox, ComboBox*** and ***TextBox***. Additionally the contents in the ***DynamicComboBox*** will also be dynamically loaded with the values contained in this function.

In addition to the "Text" and "Value" fields used to load the drop down box values, each record in the SQL query must also contain the "TextList" and "ValueList" fields. These fields will contain a comma delimited list of the Text and Value entries that should become active when the respective item is selected.

The entire list will be loaded into the ***DynamicComboBox***, with the respective Text and Value entries. In the case of changing the captions of the form functions, the caption entry must start with an @ symbol, and the rest of the caption must match a Text value in the "Text" list. If the caption is contained in the "Text" list, the caption will be replaced by the respective "Value" list entry.

Fields used:
Caption: The Caption to be displayed next to the dynamic list box.
Name: The name of the parameter which will contain the value returned by the dynamic list box.
Parameter, Query Field, Fixed Value: These fields give the value to be used as the initial selected value of the dynamic list box. See the ***GetValue()*** function for further details. The dynamic combo box and caption values will be initialized to the selected item in the dynamic combo box.
SQL Statement: This field contains the SQL statement that must return "Text", "Value", "TextList" and "ValueList" fields. The items in the dynamic list box will be displayed in the order in which the query returns it.

4.19 EditBox

This function displays an edit box on a form and is used between the ***Form*** and ***FormClose*** functions.

Fields used:
Caption: The Caption to be displayed next to the edit box.
Width: Indicates the width of the edit box. If the width is not specified, a width of 20 is used. This width might be interpreted differently by different browsers.
Name: The name of the parameter which will contain the value entered into the edit box.
Parameter, Query Field, Fixed Value: These fields give the value to be used as the initial value of the edit box. See the ***GetValue()*** function for further details.

1 for Focus: A value of 1 (one) will set the focus on this edit box when the application is run.

4.20 FileName

This function allows the user to browse for a picture. The path to the picture will be displayed in an entry box next to the browse button.

Fields used:
Caption: The Caption to be displayed above the entry box.
Width: Indicates the width of the entry box. If the width is not specified, a width of 20 is used. This width might be interpreted differently by different browsers.
Name: The name of the parameter which will contain the value which the user selected for the entry box.
Parameter, Query Field, Fixed Value: These fields give the value of the key field. See the ***GetValue()*** function for further details.

4.21 FileSave

This function saves the path to the picture which the user selected in the ***FileName*** function.

Fields used:
File Parameter: This is the parameter name used in the form which contains the uploaded file to be saved.
Parameter, Query Field, Fixed Value: These fields give the filename where the file must be saved. See the ***GetValue()*** function for further details.

4.22 Folder

A folder is an area on the page that can be dynamically opened or closed. The content of the folder will be everything between the ***Folder*** and ***FolderClose*** functions. When a folder is closed, only the caption line will be visible. Folders can be contained within other folders.

Fields used:
Background Color: This background color can be set to override the default background color of the folder caption.
Indent: The folder can be indented to show it as a sub-folder. The indent shows the depth of the indent (number 1, 2, 3 …).
Caption: The caption on the folder line next to the icon used to open or close the folder.

4.23 FolderClose

This function is used to close the folder area. The folder area is the area between the ***Folder*** and ***FolderClose*** functions. No fields are used.

4.24 Form

This function will open a new form. A form cannot be contained within another form. No error message will be shown if this is done, but the results of the application might not be desirable. A form can contain any of the form functions, like ***EditBox*** or ***Button***, and is closed by using the ***FormClose*** function.

Fields used:
1 for NO formatting: By default a form is formatted to be displayed in a two column table, with the captions on the left and the form elements on the right, with buttons centered across the columns. If you would like to use your own formatting, place a 1 in this field.
Application: This field contains the name of the application to which the form will pass the parameters.
Name: An optional parameter name can be specified to be passed with the form.
Parameter, Query Field, Fixed Value: These fields give the value of the specified parameter. See the ***GetValue()*** function for further details.

4.25 FormClose

This function closes a form that was opened by using the ***Form*** function. A form must be closed before another ***Form*** function is called to ensure that the application functions properly. No fields are used.

4.26 ForSQL

This function is used to create a loop that will perform all the functions between the ***ForSQL*** and ***NextSQL*** functions for every record in the query generated by the ***SQL*** function. No fields are used.

4.27 GetValue() Function – *You cannot call this function directly*

The GetValue() function is not used as a function on its own, but forms part of many other functions. It loads a value from either a parameter which was passed from a previous application, a database field, or a specified fixed value. This value is returned to be used by the function.

Fields used:
Parameter: Specifies a parameter name which contains the value. If the parameter doesn't exist, a blank value will be returned.
Query Field: Specifies a field name in the query which contains the value. The query is opened using the ***SQL*** function.
Fixed Value: If neither a parameter name nor a field name was given, the value of this field will be used as the value.

4.28 GraphLine

More details to follow in the next version of the document.

4.29 GraphSQL

More details to follow in the next version of the document.

4.30 Hidden

A hidden field is a way to pass a parameter within a form without showing the value on the screen. If only a single parameter is needed to be passed with the form, use the parameter option in the ***Form*** function to specify the parameter. Use the ***Hidden*** function to specify additional parameters. Use the ***Hidden*** function between the ***Form***- and ***FormClose*** functions.

Fields used:
Name: The parameter name which will contain the value.
Parameter, Query Field, Fixed Value: These fields give the value of the parameter. See the ***GetValue()*** function for further details.

4.31 Html

This function allows you to add HTML text into the application. A value can be displayed after the HTML code.

Fields used:
Parameter, Query Field, Fixed Value: These fields give the value that must be displayed after the HTML code. See the ***GetValue()*** function for further details.
HTML Code: The HTML code

4.32 IfGoto

This function allows the user to transfer the flow of control in an application. When the If-condition results to true, flow continues to the application line specified. When the If-condition results to false, execution of the application carries on as normal. Remember to alter the GotoNo if you have renumbered the sequence of your application.

Fields used:
Test Value: The value with which the parameter value will be compared.
Condition: The condition with which to evaluate the parameter value and the test value. The conditions =, <>, <, <=, > and >= can be used.
GotoNo: The line number to which the flow of control must transfer if the condition results to true.
Parameter, Query Field, Fixed Value: These fields give the value of the key field. See the ***GetValue()*** function for further details.

4.33 IfStop

This function allows the user to specify a specific point where the application will end.

Fields used:
Fixed Value: The value with which the parameter value will be compared.
Condition: The condition with which to evaluate the parameter value and the fixed value. The conditions =, <>, <, <=, > and >= can be used.
Parameter, Query Field, Fixed Value: These fields give the value of the key field. See the ***GetValue()*** function for further details.

4.34 Link

This function creates a hyperlink to another application. You can also pass a parameter to the application by adding a parameter in the fields. To add additional parameters you must use the ***Parameter*** function before the ***Link*** function for each of the additional parameters.

Fields used:
Application: Is the application to which the link points.
Name: An optional parameter name to pass to the application.
Parameter, Query Field, Fixed Value: These fields give the value of the parameter. See the ***GetValue()*** function for further details.
Link Text: This is the text that is displayed in the link.

4.35 Menu

This function indicates the start of a drop down sub menu or shortcut menu within a menu application. The setting indicated by the parameters will determine how the menu will be displayed. If a shortcut menu is specified and a shortcut menu area is defined in the template, the sub menu will be displayed in the shortcut menu area. If either of these conditions is not true, a drop down sub menu will be displayed. The sub menu is ended with the ***MenuClose*** function.

Fields used:
Shortcut Title: Shortcut menu title. This value is ignored in a drop down sub menu.
Submenu Name: Sub menu name. This name is used to call the submenu using the ***MenuSubMenu*** function.
Width: The width of the menu in pixels.
1 for Shortcut: A **1** (one) in this field indicates that this is a shortcut menu.

4.36 MenuClose

This function indicates the end of a drop down sub menu or shortcut menu that was started using the ***Menu*** command. No fields are used.

4.37 MenuItem

This function indicates a menu item on the main menu, a sub menu or a shortcut menu. The menu item calls an application and can optionally start that application with a fixed set of parameters. Another URL can be specified if the resulting page is not part of AppWizard.

Fields used:
Display Text: Text displayed on the menu.
Application: Application to be called. If empty, the URL in Memo1 will be used.
Name: Name of the parameter to be passed to the application.
Parameter, Query Field, Fixed Value: These fields give the value of the parameter. See the ***GetValue()*** function for further details.

4.38 MenuMainClose

This function is used in a menu application to indicate that the main menu is completed. Any menu item after this function will be regarded as being part of a submenu. There is no main menu start function because the menu starts as a main menu. No fields are used.

4.39 MenuSubMenu

This function calls a submenu to be displayed. The submenu can either be displayed as a drop down menu or as a shortcut menu. The submenu must be defined with a ***Menu*** function, followed by optional ***MenuItem*** and more ***MenuSubMenu*** functions, and closed using the ***MenuClose*** function.

The submenu level indicates the menu depth of the submenu. The first submenu's from the main menu has a level of 0. The sub-submenus called from these submenus will have a level of 1. Increment the submenu level every time the menu grows deeper. If the submenu levels are not set correctly, the showing and removing of submenus will not work correctly. The level is ignored if the submenu is set as a shortcut menu. Ensure that the resulting ***Menu*** function is set up as a shortcut menu as well.

Fields used:
Display Text: Text displayed on the menu.
Submenu Name: Submenu name. This must be same as the name given in the ***Menu*** function.
Menu Level: Submenu depth level.
1 for Shortcut Menu: If this field is set to **1** (one), the submenu will be a shortcut menu.

4.40 NextSQL

This function returns the loop to the ***ForSQL*** function until all the records in the query generated by the ***SQL*** function were evaluated. No fields are used.

4.41 Parameter

This function adds extra parameters for the various functions that need to pass parameters to the next application, including the ***Link***, ***TableField***, and ***Redirect*** functions. After a function is called using the parameters, the parameters are cleared.

Fields used:
Name: The parameter name to be included.
Parameter, Query Field, Fixed Value: These fields give the value of the parameter. See the ***GetValue()*** function for further details.

4.42 Password

This function allows the user to enter a password. The information entered will be displayed as round bullets.

Fields used:
Caption: The Caption to be displayed next to the password box.
Width: Indicates the width of the password box. If the width is not specified, a width of 20 is used. This width might be interpreted differently by different browsers.
Name: The name of the parameter which will contain the value entered into the password box.
Parameter, Query Field, Fixed Value: These fields give the value of the key field. See the ***GetValue()*** function for further details.

4.43 Redirect

This function will pass the control to another application. The current application is ended and the browser passes to the other application as if a hyperlink was clicked on that page. Any parameters that are required by the new application must be passed to it. The ***Redirect*** function can pass one parameter, but if more parameters must be passed the ***Parameter*** function must be used to specify each of them. Use the ***Parameter*** function just before the ***Redirect*** function in your application.

Fields used:
Application: The name of the application to which transfer must flow.
Name: An optional parameter name to pass to the application.
Parameter, Query Field, Fixed Value: These fields give the value of the parameter. See the ***GetValue()*** function for further details.

4.44 SetVariable

This function is used to set the value of a variable within an application and to do basic calculations on a variable. The result of the ***GetValue()*** function will be applied to the variable, depending on what the specified calculation is, as follows:

Calculation	Result	Type

none (blank)	Variable = GetValue()	Text and Numeric
= (equal to)	Variable = GetValue()	Text and Numeric
+ (add)	Variable = Variable + GetValue()	Numeric
- (subtract)	Variable = Variable – GetValue()	Numeric
* (multiply)	Variable = Variable * GetValue()	Numeric
/ (divide)	Variable = Variable / GetValue()	Numeric
& (concatenate)	Variable = Variable & GetValue()	Text

IMPORTANT: Any numeric calculation performed on a text value or a division by zero will result in an error.

Fields used:
Name: The variable name.
Calculation: This is the calculation that should be performed on the variable.
Parameter, Query Field, Fixed Value: These fields give the value that should be applied to the variable. See the ***GetValue()*** function for further details.

4.45 SQL

This function opens a database query. The result fields of the query can be used as values in the ***GetValue()*** function or as fields in a table. If the fields are used as values, the first record's values will be used. If the fields are used as values in a table, a table row will be generated for every record returned by the query. The query must be closed using the ***DataClose*** before another ***SQL*** function is called. An SQL statement is generated as follows: *SQL Statement* **where** *Where name=Parameter / Query Field / Fixed Value* **order by** *Order By*. If *Where name* has no value, the "where"-clause will not be included. If *Order By* has no value, the "order by"-clause will not be included.

The "where" condition is the logic used to compare the field with the parameter. If not given, the field must be equal to the parameter. Other possible conditions are >, >=, <, <=, like.

Fields used:
Where Name: This is a parameter that will be used to generate the "where"-clause of the query. If the SQL statement starts with an EXEC to run a stored procedure, the word w*here* will not be included in the final query.
Parameter, Query Field, Fixed Value: These fields give the value of the parameter. See the ***GetValue()*** function for further details.
SQL Statement: The SQL query statement
Order By: This value will be used to generate an "order by"-clause.

4.46 StoredProc

This function calls a stored procedure.

Fields used:
Proc Name: The name of the stored procedure to be called.

1 to Execute: A value of **1** (one) will execute the stored procedure.
@Name: The name of the parameter passed to the stored procedure. The parameter name must begin with @.
Parameter, Query Field, Fixed Value: These fields give the value of the key field. See the ***GetValue()*** function for further details.

4.47 StoredProcParam

This function allows the user to pass additional parameters to the stored procedure.

Fields used:
1 to Execute: A value of **1** (one) will execute the stored procedure.
@Name: The name of the parameter passed to the stored procedure. The parameter name must begin with @.
Parameter, Query Field, Fixed Value: These fields give the value of the key field. See the ***GetValue()*** function for further details.

4.48 Table

Use this function to initialize a table. The table properties are set using this function. Use the ***TableField*** function to specify each column that will be displayed in the table. The table rows will be generated using the data given in a query opened using the ***SQL*** function. After all the columns are specified, use the ***TableShow*** function to generate the table.

Fields Used:
Border Type: This field specifies the table border type. A value of 1 will show a table with a border. A value of 2 will show a compact table without a border or spacing between cells (this allows for continuous images to be built in the table). Any other value will show a table without a border and with normal spacing between cells.
Border Color: This field specifies the border color of the table. Use HTML-color codes.
Caption Color: This field specifies the background color of the row containing the column headers. Use HTML-color codes. If this value is not given, the default background color will be used.
Width: This field contains the width of the table. If the width must be a percentage of the screen, include the % sign when the width is specified.
1 to Show Totals: A value of **1** (one) will add a totals row at the end of the table with the sum of the column values.
Colspan of Totals: The value specified in **Text of Totals:** will span over the amount of columns entered in this field.
Text of Totals: The name of the row displaying the totals of the columns.

4.49 TableCheckBox

This function displays a check box in a table.

Fields used:

Field name: Specifies the field name containing the value to be displayed.
Alignment: Displays the data left aligned (left), right aligned (right), or centered (center).
Parameter Name: The name of the parameter to be passed to the application.
Caption: This field contains the caption to be displayed in the first row as the column header.

4.50 TableField

This function specifies each text column of a table after the ***Table*** function was called. The text of a field in the query opened using the ***SQL*** function will be displayed in the column. The column can also be used to link to another application by passing a fixed value or a field value in the query as parameter. Any further parameters specified using the ***Parameter*** function can also be passed to the application.

Fields Used:
Display Field: Specifies the field name containing the value to be displayed.
Application: This is the application name of the application to which the link should point. If this field is empty, the column will not be displayed as a link.
Name: The name of the parameter to be passed to the application.
Parameter Field: This field gives the value of the parameter. See the ***GetValue()*** function for further details.
1=Number Format/2=Integer: A value of **1** (one) will display the value as an integer, and a value of **2** will display the value with the thousand-separator and two decimals.
GROUP or Summary Type: GROUP will act the same as the Group By-clause in a SQL statement, and SUM will add all the values together and display it in the Totals row if it is specified in the ***Table*** function.
Caption: This field contains the caption to be displayed in the first row as the column header.
Alignment: Displays the data left aligned (left), right aligned (right), or centered (center).

4.51 TableHidden

More details to follow in the next version of the document.

4.52 TableShow

This function displays the entire table. Each record returned by the query will show a separate row in the table. No fields are used.

4.53 TextBox

This function shows a text box as part of a form and should be used between the ***Form*** and ***FormClose*** functions. A text box is like an edit box, but it has more than one row and a vertical scroll bar, enabling free typing.

Fields Used:
Caption: The Caption to be displayed next to the text box.

Width: Indicates the width of the text box. If the width is not specified, a width of 20 is used. This width might be interpreted differently by different browsers.
Name: The name of the parameter which will contain the value entered into the text box.
Parameter, Query Field, Fixed Value: These fields give the value to be used as the initial value of the text box. See the ***GetValue()*** function for further details.
Rows: Indicates the number of rows in the text box. If the rows are not specified, a default of 2 rows will be used.

4.54 TreeView

More details to follow in the next version of the document.
Builds a TreeView of the content of the SQL.
SQL must have the following fields:
Depth – How deep the item must be displayed, 1 being root level on the left
Key – A Unique key field value which will identify the node
Text – the text that must be displayed in the tree
Application – the next application to be called when clicking on the text (if this is empty, the txt will not be a link)
Parameter – the paremeter names and values that must accompany the link, in the format Par=Val&Par=Val etc.