OPENGATE SMALL BUSINESS SOFTWARE

VERSION 2.1

Developer's Guide

Version 1.31.2011

TABLE OF CONTENTS

1 Welcome

OpenGate Small Business Software is an extensive package of feature from Customer Relationship Management (CRM) to Sales Order Management to Financial Management. This guide is intended to help developers understand how to customize the OpenGate Small Business product to your specific needs.

2 Customization Best Practices

This section is intended to serve as a guide for customization best practices to ensure you are able to easily modify OpenGate Small Business, and help keep your maintenance time and costs low when new updates are available. The section is broken out into the three distinct areas where customization practices are different.

2.1 *TABLE CUSTOMIZATION*

Adding new fields to your database tables is fairly easy to accomplish. Simply open your data file, open the table in design mode, and add the necessary fields. You will need to add those fields to any forms, queries, or reports where the information is relevant.

2.2 *QUERY CUSTOMIZATION*

You can add new queries to OpenGate Small Business without any concern for impacts to other features. If you are adding new fields to pre-delivered queries, or changing a query that is delivered with OpenGate Small Business, it is important to understand the primary function(s) of several delivered queries and potential issues.

Query	**Purpose**	**Notes**
qryAccountJoin_Crosstab	This query displays when a user clicks the "Annual Summary" button in the Income/Expense Entry screen.	Adding a field to this query is not recommended.
qryOrderLineItems	This query is used by many functions to inspect and total the individual line items of a sales order.	
qryUniqueCustomerActivities		
qryOrderReportSource	This query is used in order reports and dashboards.	
qryProspect		
qryServiceLineItems	This query is used by many functions to inspect and total the individual line items of a sales order.	
qryOrderPowerSearch	This query is used by the PowerSearch feature.	
qryLicensedProducts	This query displays the licensed products for each customer.	
qryContactCustJoin		
qryMasterCustomerCare	This query summarizes customers and the products and services they have purchased. It is used by the Customer Care Manager to select customers that match specific criteria.	If you add any customer or order table fields, you will want to add them to this query.

Query	Purpose	Notes
qryMasterCustomerCare_product	This is a source query that ultimately feeds the qryMasterCustomerCare query.	If you add any fields to your products table, you will want to add them here.
qryMasterCustomerCare_service	This is a source query that ultimately feeds the qryMasterCustomerCare query.	If you add any fields to your services table, you will want to add them here.
qryProductMailMerge		
qryPOSummary		
qryOrder_Dashboard_Source	This query is used in the order dashboards.	
qrytmpMissingFinancialImport	This temporary query is used to export any records that were not imported by the financial import feature.	
qryCampaignPerformance	This query serves as the source for campaign performance metrics on dashboards.	
qryOrderConfirmation_simple	This is a mail merge query that is	
qryPurchaseOrderSummary		
qryOnOrderCalculation		
qryGoogleEarthMapSource		
qryItemInventory		
qryAccountTransactionExport		
qryAccountJoin		
qrytmpMissingCustomerImport		
qrytmpCustomerList		
qryMasterCustomerCare_unique		
qryServiceRenewalReport_source		
qryBillingSource		
qryObjectsNotIntblUpgrade		
qryCampaignTargetMerge		
qryBillingSource		
qryObjectsNotIntblUpgrade		
qryCampaignTargetMerge		

2.3 FORM AND REPORT CUSTOMIZATION

When customizing OpenGate Small Business, there are two primary modes for customization:

A. Modify the form or report delivered by OpenGate. This method will make it difficult to apply updates to your software at a later date and keep your customizations. If you decide to upgrade, you will need to add your customizations into the new version of the form or report.

B. Use the built-in customization tool to create copies of the delivered forms and reports. The cusomization tool is available from the LaunchPad form, or by opening frmAdminCustomObjects. The customization tool will create a copy of the selected form or report, adding "_custom" to the end of the object's name. When you upgrade OpenGate Small Business, your custom objects are not changed. You may need to ensure your custom object is compatible with any changes to VB function calls in the new version.

3 Database Architecture

3.1 DATABASE FILES

OpenGate Small Business is split into a front end database which serves as the user interface, and contains the business logic for the product. The front end is effectively similar to an executable file, only it relies on Microsoft Access to serve as the application engine. The back end data file contains the entire database and data to operate OpenGate Small Business.

Typically customers place their back end data file on a shared file server on a local area network so that multiple front end database clients on different machines can use the database simulaneously.

3.2 DATABASE SCHEMA

The full database schema is available by opening the back end data file (password provided upon request) and opening the Relationships tool. Because the number of tables and relationships is significant, they are not detailed in this guide.

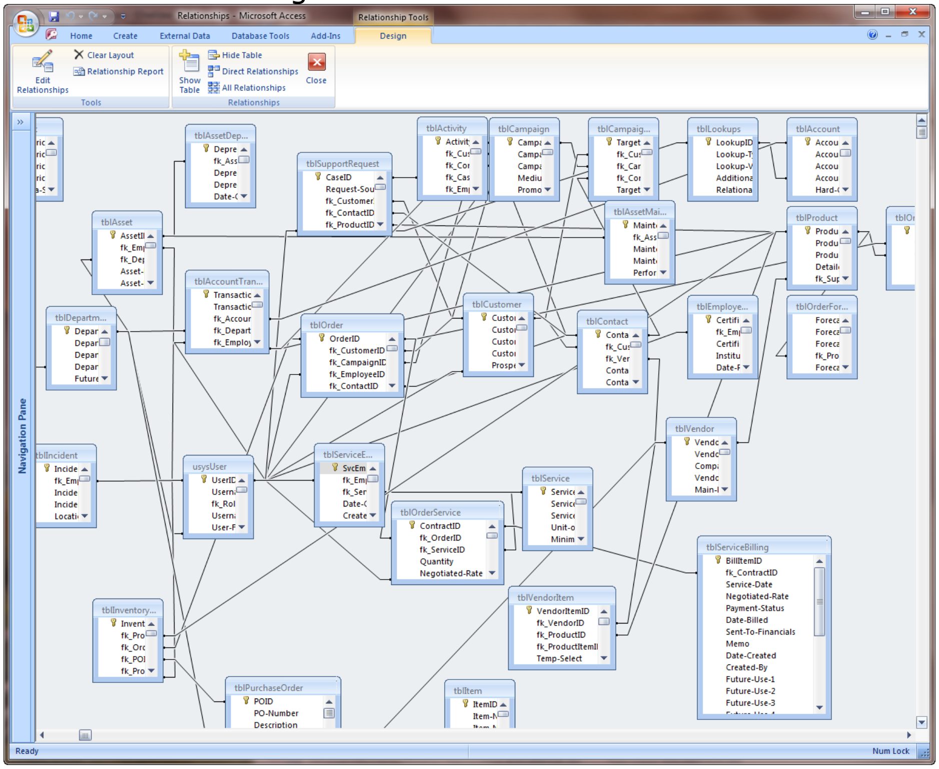


Figure 1: Database Schema and Relationships

You may notice some tables in the database that are not presently used. OpenGate Software has architected the database to support many future features that are not currently active. Should you decide to develop using these tables, please note that your changes and data entered may not be compatible with our future feature designed.

3.3 LOOKUP FIELD DESIGN

To simplify the large number of lookup fields in the database, we chose to design OpenGate Small Business with a single Lookup table (tblLookups) to contain the majority of lookup fields. In most cases, the lookup values are not relational (that is, if the lookup value is changed, the related records in other tables will not reflect that change). This design choice was made to simplify lookup value manage by end users, and allow users to maintain "point in time" lookup values. This does mean that queries must be run to update changed lookup values to reflect new values.

4 Form and Report Design

4.1 TEMPLATES

OpenGate Small Business includes several template forms and reports to help you create forms and reports that are consistent with the other screens in the product. Following is the list of templates:

• frmTemplate	Summary form template
• subfrmTemplate	Subform template
• rptTemplate	Report template

The templates include the base VB function calls to set up the forms and reports when loaded, including changing the logos on reports, and header label color schemes on the forms.

We recommend copying the template and assigning your own form/report name. Then proceed with changing the recordsource of the object, and adding the fields and features you need for that object.

5 VBA Functions

5.1 APPLICATION FUNCTIONS

5.1.1 PURCHASE ORDERS

5.1.1.1 fChangePOPaymentStatus

Description

This function processes a change to the PO Payment Status field. If the payment status field is changed to "Paid in Full," the function will validate that all required fields are filled out, and update the table tblAccountTransaction with an expense record (using fPOToLedger) if configured to do so.

Function Name

fChangePOPaymentStatus

Parameters and Use

objForm – The form object from which the function was called. Use "Me" in most cases.

5.1.1.2 fChangePOStatus

Description

This function processes a change to the PO Status field. If the PO Status is "Complete" and the payment status field is not equal to "Paid in Full," the function will prompt the user to ask if they want to set the PO Payment Status = "Paid In Full." If the user answers with "Yes," then the function fChangePOPaymentStatus is called.

Function Name

fChangePOPaymentStatus

Parameters and Use

objForm – The form object from which the function was called. Use "Me" in most cases.

5.1.1.3 fPOToLedger

Description

This function updates the account transaction table (tblAccountTransaction) with the expense amounts related to the purchase order.

Function Name

fPOToLedger

Parameters and Use

lngPOID - The purchase order ID key.

lngVendorID - The vendor ID key associated with the PO.

strPayee - Optional. If passed, the Payee field in the Income/Expense record will reflect the value provided.

blnReverse - Optional. If set to True, a new record for the PO expense amount will be created in tblAccountTransaction in a positive amount.

lngDepartmentID - Optional. The Department ID on the tblAccountTransaction record will reflect this value, if provided.

lngAccountID - Optional. The Department ID on the tblAccountTransaction record will reflect this value, if provided.

5.1.2 Sales Orders

5.1.2.1 fChangeOrderStatus

Description

This function processes a change to the Order Status field. If the Status is "Complete" and the payment status field is not equal to "Paid in Full," the function will prompt the user to ask if they want to set the Payment Status = "Paid In Full." If the user answers with "Yes," then the function fChangeOrderPaymentStatus is called.

Function Name

fChangeOrderStatus

Parameters and Use

objForm – The form object from which the function was called. Use "Me" in most cases.

5.1.2.2 fChangeOrderPaymentStatus

Description

This function processes a change to the Order Payment Status field. If the payment status field is changed to "Paid in Full," the function will validate that all required fields are filled out, and update the table tblAccountTransaction with a revenue and tax record (using fOrderToLedger) if configured to do so.

Function Name

fChangeOrderPaymentStatus

Parameters and Use

objForm – The form object from which the function was called. Use “Me” in most cases.

5.1.2.3 fOrderToLedger

Description

This function updates the account transaction table (tblAccountTransaction) with the revenue and tax amounts related to the sales order. The Account for revenue defaults to "Income" and the default account for the sales tax record is " Sales Tax Collected." The default accounts can be configured in the Application Preferences screen (frmAdminFoundation) in the "Financials" tab.

Function Name

fOrderToLedger

Parameters and Use

lngOrderID - The order ID key.

lngCustomerID- The Customer ID key associated with the Order.

strPayee - Optional. If passed, the Payee field in the Income/Expense record will reflect the value provided.

blnReverse - Optional. If set to True, new records for the Order revenue and tax amounts will be created in tblAccountTransaction in a negative amount.

5.1.2.4 fReverseOrder

Description

This function serves as the primary means to reverse a sales order. The function will update the screen to display the order status and payment status passed in to the function. The function will call fOrderToLedger to create a reversing financial transaction, and if you have configured OpenGate Small Business to maintain product inventories, will prompt the user to indicate if they want to create a reversing inventory entry.[1]

Function Name

fReverseOrder

Parameters and Use

objForm – The form object from which the function was called. Use “Me” in most cases.

[1] The user is prompted because in some cases a returned order cannot be returned to inventory.

strOrderStatus– The order status that the order will be changed to on the screen.

strPaymentStatus– The order payment status that the order will be changed to on the screen.

5.1.2.5 fUpdateInventoryItems

Description

This function updates the inventory transaction table, tblInventoryItem, for a specific OrderID.

Function Name

fUpdateInventoryItems

Parameters and Use

lngOrderID– The primary key of the order (OrderID)

dtTransactionDate– Optional. The date you wish to appear on the inventory item transactions for this order.

blnReverse– Optional. If set to "True," a new set of inventory records will be created with a negative quantity in the "Quantity-Sold" column.

5.1.2.6 fGetProductOrderTotal

5.1.2.7 fGetServiceOrderTotal

5.1.2.8 fGetTotalOrderRevenue

5.1.2.9 fGetTotalOrderTax

5.1.2.10 fDeleteOrder

Description

This deletes an order record, associated tblOrderProduct and tblOrderService records, and if desired, all financial and inventory records as well.

Function Name

fDeleteOrder

Parameters and Use

lngOrderID– The primary key of the order (OrderID)

blnConfirm– Optional. Set to "True" if you wish to prompt the user to confirm the order deletion first.

blnDeleteCompleteTrail– Optional. If set to "True," the system will delete all related financial and inventory transactions associated with the order.

5.1.2.11 fApplyPaymentFee

5.1.3 Prospects and Customers

5.1.3.1 fDeleteCustomer

Description

This deletes a customer record and all associated contact, activity, support request, and campaign records.

Function Name

fDeleteCustomer

Parameters and Use

lngCustomerID– The primary key of the customer (CustomerID)

blnConfirm– Optional. Set to "True" if you wish to prompt the user to confirm the order deletion first.

5.1.3.2 fUpdateCustomerSince

Description

This function updates the "Customer-Since" field on a customer record with the specified date if the "Customer-Since" field is null. This function is called by the function fChangeOrderPaymentStatus.

Function Name

fUpdateCustomerSince

Parameters and Use

lngCustomerID– The primary key of the customer (CustomerID)

dtDate– The date this customer first ordered a product or service.

5.1.3.3 fUpdateProspectStatus

Description

This function updates the "Prospect-Status" field on a customer record with the specified status. This function is called by the function fChangeOrderPaymentStatus.

Function Name

fUpdateProspectStatus

Parameters and Use

lngCustomerID– The primary key of the customer (CustomerID)

strStatus– Optional. The current status of the prospect. If left empty, the status is set to "Closed."

5.1.4 *Reminders*

5.1.5 *Mapping with Google Earth*

OpenGate Small Business generates a map of customers and prospects when the function fMapCustomers() is called. This function is set up to call the primary mapping function, fOutputKML_Generic(). If you choose to create your own custom mapping function, we recommend looking at fMapCustomers() as an example of how to call fOutputKML_Generic().

Function Name

fOutputKML_Generic

Parameters and Use

strFileName - The filename and path to save the Google Earth KML file to.

strSQL - The full SQL SELECT statement to use to collect the records for mapping.

strGroup1Field - The first field to group records by in the Google Earth folders.

strGroup2Field - The second field to group records by in the Google Earth folders. The second grouping field will appear as child folders under the first grouping field (*strGroup1Field*).

strMapType - The type of map to display. At this time the only supported mapping is "Customers." The primary customer fields are used to create the hover-over labels for Google Earth (customer name, status, mailing address, city, state, country)

blnOpenGoogle - If set to "True," Google Earth will be opened automatically once the KML file is created.

lnShowLabels - Not currently used.

strSearchName - The name of the Google Earth parent folder.

5.2 SYSTEM FUNCTIONS

5.2.1 FORM APPEARANCE FUNCTIONS

5.2.1.1 Future Use Fields

5.2.1.2 Form Color Schemes

The function sApplyFormColorScheme is a global subroutine used by all forms to apply the same color scheme to a form's header that is used by the menu buttons. To use this routine in your own forms, you must have one or more labels in the header section named "HeaderLabel#" and a box control named "HeaderBox."

Function Name
sApplyFormColorScheme

Parameters and Use

objForm– The form object that is calling the subroutine. Use "Me".

intHeaderLabels– The number of header labels on the form.

5.2.2 INFORMATIONAL FUNCTIONS

5.2.2.1 fMyCompanyAddress

The fMyCompanyAddress function returns a formatted string with the designated company address from tblMyCompany.

5.2.3 RE-LINKING TO, MOVING, AND BACKING UP THE DATA FILE

OpenGate Small Business will handle broken table links automatically. If you need to relink tables, use the function fRefreshLinks(). You can optionally pass in the path to the data file, or leave blank to prompt the user for the location of the back end data file.

Use the function fMoveDataFile() to move the back end data file. You may pass in the new path, or leave blank to prompt the user for the new location.

To back up the data file (even if users are active in the database), use the function fBackUp().

5.2.4 *Event Logging*

The Event Log function in UI Builder provides a way for you to log important events to a local or remote table, or to an XML output file in the location you specify. There are three functions that are used for event logging, one to initialize the log, the second to write to the log, and the third to close the log when finished. To optimize performance, if you are using a table to log events, the table will remain open for the duration of the user's session, or until you intentionally close the log with the fCloseLog() function.

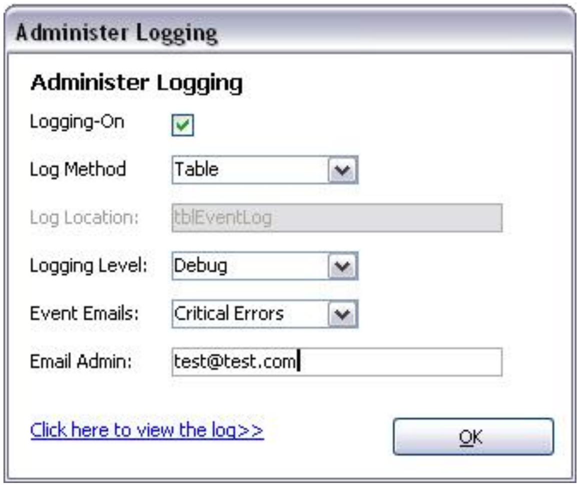
If you choose to write to the log table ("tblEventLog"), you can locate the table locally on the PC where your database application is used, or copy the table to a file server and then link the table back to your user's database.

Similarly, you can choose to store the XML event log on the local PC where the database is being used, or to a central location such as a file server.

Important!
Each copy of your database application can write to the same remote log table (linked) or central file. Note that each copy of your database application will have it's own settings and if you want to change them universally, you will need to do so for each copy.

5.2.4.1 Configuring the log

To configure the log, open the Administration form/subform, or open the form "frmLoggingAdmin" from the database window.



Administer Logging

Administer Logging

Logging-On ☑

Log Method: Table

Log Location: tblEventLog

Logging Level: Debug

Event Emails: Critical Errors

Email Admin: test@test.com

Logging Administration Form

You can choose to turn logging on or off for the current Access database, specify if log events should be written to the event log table, or a file location you specify. Note that if you specify a file name, you will need to provide a fully qualified path name and file name in the "Log Location" field. The file format will be XML (without header/footer), and will not depend upon the file name extension you supply in the "Log Location" field.

The logging level will dictate whether certain events are written to the log or not. When you define a log event you can indicate if it is a Normal or Debug event. If you have the Logging Level set to "Debug," all events will be written to the log. If set to "Normal," only normal events will be written to the log, Debug events will be ignored.

Finally, you can send an email to a designated administrator when certain events occur. There are three values for the "Event Emails" setting:

Never – Events will never be emailed to the Administrator.

Defined Events – Only events where the parameter "blnEmailAdmin" is set to True in the fLogEvent() function.

Critical Errors – Any event where the parameter "intEventType" is set to "auiCriticalError" will be emailed to the designated administrator.

5.2.4.2 Initializing the Log

Description

The function fInitializeLog() obtains logging settings from the table tblAppInfo, such as where log events should be written to (table or file), what level of log events will be written, and what email address should be used to email log events, if any. The function is automatically called whenever the main form (frmMain) is opened. If you want to log specific events that might happen before the main form is ever opened, you will need to call fInitializeLog, otherwise it is done automatically for you.

Function Name

fInitializeLog

Parameters and Use

No parameters.

5.2.4.3 Writing to the Log

Description

Writing to the log can occur at any time after the log is initialized.

Function Name

fLogEvent

Parameters and Use

strDescription - Description of the event which will be passed to the user and/or table/file.

intEventLevel - Determines if the event will be written to the table/file based on the application's logging level (normal/debug).

blnAlarmUser - If set to True, the user will see a popup alert with the text of the event.

strSource - The source of the event.

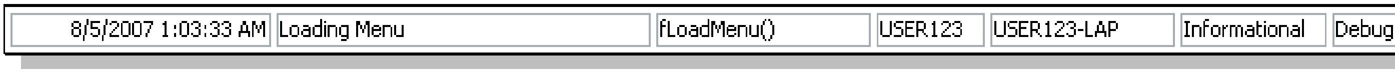
intEventType - Category of the event to help you understand if it is informational, an error, or critical error.

blnEmailAdmin - Determines if the event is emailed to an administrator. There are three modes, the default being only those events where they blnEmailAdmin is explicitly set to True. Alternatively, you can set the system to email no events, or critical errors, in which case it doesn't matter whether blnEmailAdmin is True or False when passed in. Note that blank = Never email events.

Example 1

```
fLogEvent "Loading Menu", auiDebug, False, "fLoadMenu()", auiInformation
```

In this example, we are logging an event with description “Loading Menu” that will only be captured if the Logging Level = Debug, we won’t alert the user, and the origin is “fLoadMenu()”. This is an informational event. No email will be sent to the administrator. If the Logging Level = Debug, this is how the event will appear in the log table:

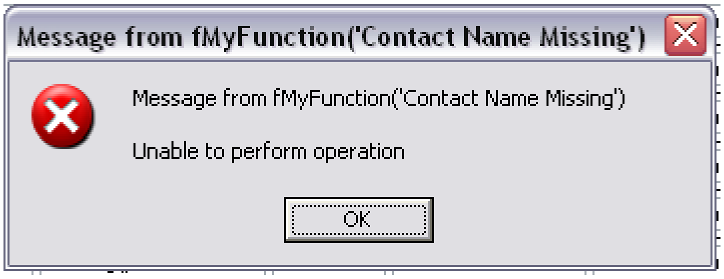


8/5/2007 1:03:33 AM	Loading Menu	fLoadMenu()	USER123	USER123-LAP	Informational	Debug

Example 2

fLogEvent err.Description, auiNormal, True, "fMyFunction(" & strMyVariable & ")”, auiCriticalError, True

In this example, we are logging an event with description of the Access error that occurs (generally when you are using error handling in a function). It will only be captured regardless of the Logging Level, and we will alert the user. The origin is “fMyFunction” and we will also log the information that was passed into the function through strMyVariable. This is a critical error.



Message from fMyFunction('Contact Name Missing')

Message from fMyFunction('Contact Name Missing')

Unable to perform operation

User Alert Dialog

5.2.4.4 Closing the Log

Description

The function fCloseLog() simply closes down the log function and, if necessary, the log table recordset as well. The function is automatically called whenever the main form (frmMain) is closed.

Function Name

fCloseLog

Parameters and Use

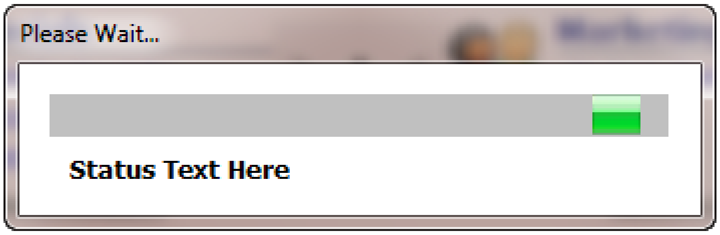
No parameters.

5.3 GENERAL VBA FUNCTIONS

5.3.1 Progress Bar

Description

The progress bar is used to notify the user of the present state of the application. The progress bar is more visible than the status bar (acSysCmdSetStatus) function, and does not require an ActiveX control, which can sometimes be problematic in a multi-user environment where each machine has different versions of the Microsoft Common Controls ActiveX file. Note that the progress bar allows you to specify the text to display to the user, but does not provide a % complete visualization, just an indication that the system is still working.



Status Text Here

Figure 2: Progress Bar Example

Function Name

fProgressUpdate

Parameters and Use

strUpdateText – String Variable. Populates the text shown on the progress bar. Up to two lines of text can be displayed as shown in the example above.

iPercent – A whole number that represents the percent of the progress to represent on the bar.

Note that you must open and close the form in our code using the Docmd.OpenForm and Docmd.CloseForm commands.

Depending upon how intense your processing is, you may need to also insert the "DoEvents" command into your code to let Access refresh the form occasionally.

Example

```
Docmd.OpenForm "frmProgress"

fProgressUpdate "This is the text that would be displayed" &
vbCrlf & "on two different lines in the progress bar"
        (your own code doing something here)
fProgressUpdate "Next status update you want to provide"
```

```
DoEvents
        (your own code doing something here)
fProgressUpdate “Next status update you want to provide”
DoEvents
(your own code doing something here)

        Docmd.CloseForm “frmProgress”
```

5.3.2 *Table Record Count*

Description

The function fTableRecordCount enables you to quickly obtain a count of records in a specified table using an SQL string you supply. You can call this function from a form field, or from another VBA function. This function provides a simple alternative to the Access DCount() function, with embedded error handling and event logging.

For example, you may have a field on a form “Customers” that you want to show the user the number of customers currently in your database. You could insert this function into the read-only field to display that information to them. Of you may have a function where your processing depends upon whether there are records in a specific table, or a specific number of records in that table meeting a given criteria.

Function Name

fTableRecordCount

Parameters and Use

strTable –Provide a valid table name (linked or local) for the current database

strSQL – Provide a valid SQL string beginning with “WHERE…” If you want to count all records in the table, simply pass “” in the variable.

blnDisplayErrors– Optional. Set to False if you do not want this function to display an error to the user if something goes wrong. For example, if you have this function called multiple times, you may not want to alert the user every time an error occurs. If left blank, UI Builder will display any error messages.

Example 1

```
lngRecords = fTableRecordCount(“tblMyTable”,””)
```

In this example, a value will be returned to lngRecords for the number of records contained in the table "tblMyTable." No SQL criteria will be applied.

Example 2

lngRecords = fTableRecordCount("tblTableX","WHERE [CustomerID] = " & me.customerid.value)

In this example, a value will be returned to lngRecords for the number of records contained in the table "tblTableX" where the CustomerID field is equal to a value passed from the current form for the field "customerid."

5.3.3 TABLE RECORD SUM

Description

The function fTableRecordSum enables you to quickly obtain a sum of records in a specified table field using an SQL string you supply. You can call this function from a form field, or from another VBA function. This function provides a simple alternative to the Access DSum() function, with embedded error handling and event logging.

For example, you may have a field on a form "Revenue" that you want to show the user the amount of revenue this month. You could insert this function into the read-only field to display that information to them. Or you may have a function where your processing depends upon whether you have generated a specific amount of revenue from orders.

Function Name

fTableRecordSum

Parameters and Use

strTable –Provide a valid table name (linked or local) for the current database

strSumField – Designate the name of the field in *strTable* that contains the data you want to sum together.

strSQL – Provide a valid SQL string beginning with "WHERE…" If you want to count all records in the table, simply pass "" in the variable.

blnDisplayErrors– Optional. Set to False if you do not want this function to display an error to the user if something goes wrong. For example, if you have this function called multiple times, you may not want to alert the user every time an error occurs. If left blank, UI Builder will display any error messages.

Example 1

lngRecords = fTableRecordSum("tblMyTable","MyField","")

In this example, a value will be returned to lngRecords for the sum of values contained in the field "MyField" in table "tblMyTable." No SQL criteria will be applied.

Example 2

lngRecords = fTableRecordSum ("tblTableX","MyField", "WHERE [Order-Date] > #1/1/2007#")

In this example, a value will be returned to lngRecords for the sum of values contained in the field "MyField" in table "tblTableX" where the Order-Date field is greater than 1/1/2007.

5.3.4 *TABLE RECORD VALUE*

Description

The function fTableRecordValue enables you to quickly obtain a value from a field in a specified table field using an SQL string you supply. You can call this function from a form field, or from another VBA function. This function provides a simple alternative to the Access DLookup() function, with embedded error handling and event logging.

For example, you may have want to fetch the current value of a field that is in a table other than the one your form uses as the data source. You could insert this function into the read-only field to display that information to them. UI Builder uses this function, as an example, to change the color scheme on some forms to conform to the color scheme you choose. The form load event obtains your currently active color scheme's menu button color to paint the menu administration form buttons the same color.

Important!
This function will return the first value found in the data source for the field you specify. If you happen to have multiple records in the data source that match your SQL expression, UI Builder will only return the first record found.

Function Name

fTableRecordValue

Parameters and Use

strTable –Provide a valid table name (linked or local) for the current database

strField – Designate the name of the field in *strTable* that contains the data you want to fetch.

strSQL – Provide a valid SQL string beginning with "WHERE…" If you want to count all records in the table, simply pass "" in the variable.

blnDisplayErrors– Optional. Set to False if you do not want this function to display an error to the user if something goes wrong. For example, if you have this function called multiple times, you may not want to alert the user every time an error occurs. If left blank, UI Builder will display any error messages.

Example 1

```
lngButtonColor = fTableRecordValue("tblColorScheme",
"MenuButtons", "WHERE [ActiveScheme] = -1", False)
```

In this example, a value will be returned to lngButtonColor for the first value contained in the field "MenuButtons" in table "tblColorScheme" where the scheme is currently active. Errors will not be displayed to the user if they occur.

5.3.5 File Check

Description

The function fFileExists allows you to validate whether a file exists or not. When you call fFileExists with a filename, it will respond with a True/False to your function to indicate if the file exists or not.

Function Name

fFileExists

Parameters and Use

strFileName – Supply the name of the file you want to determine if it exists or not.

Example

```
blnMyVariable = fFileExists("C:\Win.ini")
```

The function will return a True/False value to your calling function for the file "C:\Win.ini"

5.3.6 *OPERATING SYSTEM NAME*

Description

The code for this function is courtesy of Dev Ashish. Call the function within any other function to retrieve the name of the operating system upon which your Access database application is running.

Function Name

fOSName

Parameters and Use

No Parameters.

5.3.7 *NETWORK USERNAME*

Description

The code for this function is courtesy of Dev Ashish. Call the function within any other function to retrieve the network username for the machine on which your Access database application is running.

Function Name

fOSUserName

Parameters and Use

No Parameters.

5.3.8 *MACHINE NAME*

Description

The code for this function is courtesy of Dev Ashish. Call the function within any other function to retrieve the name of the machine on which your Access database application is running.

Function Name

fOSMachineName

Parameters and Use

No Parameters.

5.3.9 *CREATE OUTLOOK TASK – DIRECT FUNCTION CALL*

Description

UI Builder helps users create Outlook tasks quickly and efficiently, without ever leaving your database application. Simply configure a menu button to use the "Create Outlook Task" command, and UI Builder takes care of the rest. When clicked, the user is prompted to add details about the task, click "Create Task>>", and they continue working on what they were doing without the normal multi-step process.

To call the create task function directly, use the fAddOutlookTask function. For example, you might add an event that automatically creates an Outlook task to confirm an order shipped up five days in the future when a new order is entered.

Function Name

fAddOutlookTask

Parameters and Use

strSubject – The subject of the Outlook Task.

strBody – The details for the task created.

dtDueDate – Optional due date. If not supplied, the current date will be used.

Return Value – Yes/No value indicating if the merge was successful or failed.

Example

See the form frmExamples for an example of the methods used to create Outlook tasks.

5.3.10 Mail Merge API – Direct Function call

Description

UI Builder provides the ability to let users click a menu or submenu button and generate a mail merge letter or email based on a predefined Mail Merge Profile. If you want full control over how the mail merge will function, use the fMerge() function call directly.

Function Name

fMerge

Parameters and Use

strMergePath – Provide the full filename for the mail merge template document, or “Prompt User” to have the user select a template when the mail merge is run.

strDataSource – Provide the text name of the table or query to be used in the mail merge. Be sure the data source contains all the data fields that the mail merge template also uses to prevent user error messages from Microsoft Word.

strWhereStatement - Provide a valid SQL statement, beginning with “WHERE “, or leave as “” to select all records from the specified data source.

strMergeOption - Specify the type of mail merge*. Valid values are*

mrgPrint – Print the merged document immediately
mrgPreview– Open the merged document for editing
mrgMail – Send the merged email immediately
mrgMailPreview– Open the merged email for editing

strSubject – Email subject line. If the merge option mrgMail or mrgMailPreview is selected, this variable must be populated.

strEmail - Email recipient(s). Use semi-colons for multiple recipients. If the merge option mrgMail or mrgMailPreview is selected, this variable must be populated.

blnKeepOpen - True/False value that indicates if the document should remain open if using the mrgPrint method.

Return Value – Yes/No value indicating if the merge was successful or failed.

5.3.11 NOTE EDITOR

Description

Note Editor gives you the ability to condense a memo or long text field on screen, saving valuable form real estate. You place a button next to the field, or create an OnDoubleClick event, that calls the fNoteEditor function to present a small form that allows users to view and/or edit the contents of the field. The results of the user's edits are returned in the function's return variable.

Function Name

fNoteEditor

Parameters and Use

strNote - The current text contained in the field, if any.

blnReadOnly- True/False. Specify if the Note Editor should allow the user to edit the notes, or just read them.

Return Value – Text supplied by the user, or original text if no changes were made.

5.3.12 TOAST POPUPS

Description

Toast popups allow you to display an informational message to the user for a short period of time similar to email message notifications now available in most email clients.

Function Name

fDisplayPopup

Parameters and Use

strHeader - The header/title text that will appear at the top of the popup message in bold. Similar to the title bar of a message box.

strText- The main text to display to the user.

iDisplaySeconds- Optional. Indicates how long the popup message should show. By default, the message will show for 5 seconds.

blnLegacyMode- Optional. Indicates that UI Builder should use the expanding popup mode which was present in UI Builder 3.0.

6 Indices

6.1 TABLE OF FIGURES