[1. DDE Conversations 2](#_Toc230162411)

[a. The Conversation Relationship: Clients & Servers 2](#_Toc230162412)

[b. Before the Conversation is Initiated 2](#_Toc230162413)

[c. Appraise-It ServiceName, TopicName, and ItemName 2](#_Toc230162414)

[d. Related Topics: 3](#_Toc230162415)

[1) Form Names 3](#_Toc230162416)

[2) Field Locations 3](#_Toc230162417)

[3) An Example of ServiceName, TopicName, and ItemName 3](#_Toc230162418)

[4) ServiceName, TopicName and ItemName for Other Applications 3](#_Toc230162419)

[5) Initiating and Terminating the Conversation 4](#_Toc230162420)

[6) DDEInitiate (ServiceName-string, TopicName-string) DDETerminate (expression) 4](#_Toc230162421)

[e. Conversational Exchanges 4](#_Toc230162422)

[1) DDE Request 5](#_Toc230162423)

[2) DDE Poke 6](#_Toc230162424)

[3) DDEPoke (expression, Server-ItemName-expression, data-expression) 6](#_Toc230162425)

[f. DDE Execute 6](#_Toc230162426)

[g. Related Topics: 7](#_Toc230162427)

[1) DDEExecute (expression, string) 7](#_Toc230162428)

[2) Appraise-It Execute Items 7](#_Toc230162429)

[h. Providing Appraise-It Data to Other Applications 7](#_Toc230162430)

[2. Writing the DDE Macro 7](#_Toc230162431)

[1) An Example of a DDE Global Macro for Word 7](#_Toc230162432)

[2) An Example of a DDE Global Macro for Excel 8](#_Toc230162433)

[3) An Example of a DDE Global Macro for WordPerfect® for Windows 9](#_Toc230162434)

[3. Response Database Macros 10](#_Toc230162435)

[a. Writing Response Database Macros 11](#_Toc230162436)

[b. To Write a Response Database Macro 11](#_Toc230162437)

[c. Macro Files 11](#_Toc230162438)

[d. Executing Macro Files 11](#_Toc230162439)

[e. Introduction 11](#_Toc230162440)

[4. Custom Commands 12](#_Toc230162441)

[a. Custom Commands in TRA.INI 12](#_Toc230162442)

[b. Before You Add the Custom Command 12](#_Toc230162443)

[c. Adding a New Custom Command to the Tools Menu 12](#_Toc230162444)

[d. Using Macros to Provide Custom Commands 12](#_Toc230162445)

[e. Related Topics: 13](#_Toc230162446)

[1) Suggested Uses for Macros 13](#_Toc230162447)

[2) Introduction 13](#_Toc230162448)

[5. Variables 13](#_Toc230162449)

[a. User Information 13](#_Toc230162450)

[b. Information from Registration Procedure 13](#_Toc230162451)

[c. Pre-defined Keywords 14](#_Toc230162452)

[1) Across 14](#_Toc230162453)

[2) Checkbox 14](#_Toc230162454)

[3) Keypress 14](#_Toc230162455)

[4) Leave 14](#_Toc230162456)

[5) Line 14](#_Toc230162457)

[6) Multi 14](#_Toc230162458)

[7) Multihandle 14](#_Toc230162459)

[8) Single 14](#_Toc230162460)

[9) Totallines 14](#_Toc230162461)

[6. Cell Representation 14](#_Toc230162462)

[a. Cell or Field 15](#_Toc230162463)

[b. Special Syntax 15](#_Toc230162464)

[c. Single and Double Slashes 15](#_Toc230162465)

[d. Operators 15](#_Toc230162466)

[1) Math Operators 15](#_Toc230162467)

[2) Comparison Operators 15](#_Toc230162468)

[3) String Operators 15](#_Toc230162469)

[4) Procedures, Functions, and Sub Procedures 16](#_Toc230162470)

[5) Pre-defined Functions 16](#_Toc230162471)

[7. Glossary of Terms 45](#_Toc230162472)

[a. client 45](#_Toc230162473)

[b. client/server 45](#_Toc230162474)

[c. conversation 45](#_Toc230162475)

[d. dynamic data exchange (DDE) 45](#_Toc230162476)

[e. functions 45](#_Toc230162477)

[f. procedures 45](#_Toc230162478)

[g. sequential file 45](#_Toc230162479)

[h. server 45](#_Toc230162480)

[i. sub procedures 45](#_Toc230162481)

[8. Notes 45](#_Toc230162482)

[9. Additional SendCommandMessages 46](#_Toc230162483)

[10. Miscellaneous Tibits 49](#_Toc230162484)

[a. ctrl +m then print to put the cell ID in each cell 49](#_Toc230162485)

[b. ctrl +m then formname to put the formname 49](#_Toc230162486)

[c. Conditional Statements 49](#_Toc230162487)

[d. Long text 50](#_Toc230162488)

# DDE Conversations

 Dynamic Data Exchange provides the medium through which data can be exchanged between application programs or processes. This means, for example, that information from an Appraise-It report may be linked to a letter in Microsoft® Word, a monthly statement in Microsoft® Excel, and then faxed to a client via Delrina WinFAX Pro.

## The Conversation Relationship: Clients & Servers

The relationship between the application programs that are sharing information is called a conversation7.L8OZ. Conversations occur between clients and servers. The client8.INLA is the requester and receiver of the shared information. The serverWXJ36R provides the information to the client. Appraise-It is an application program that may act as both a client and a server, in other words, both requesting and providing information.

## Before the Conversation is Initiated

To initiate a DDE conversation7.L8OZ, the client8.INLA and serverWXJ36R applications need to know where to locate the information to be exchanged, and how it is to retrieved. The directions to the information are provided to the server application program in Basic code instructions. There are three main components to these directions: ServiceName, TopicName, and ItemName. When using Appraise-It, you will typically provide the Basic instructions in the form of a macro written in one of Appraise-It's response databases.

## Appraise-It ServiceName, TopicName, and ItemName

|  |  |
| --- | --- |
| **Item** | **Description** |
| ServiceName | TRA |
| TopicName | Full path of report |
| ItemName | Form Name | Line # | Field # across  |

The ServiceName for Appraise-It is simply TRA. The TopicName is the full drive, directory, and file name for the report from which you wish to retrieve information. For example, if you have a report named Sample.RPT located in the TRADATA\DATA directory of your C drive, the TopicName is C:\TRADATA\DATA\SAMPLE.RPT.

The ItemName is determined by the form name and the field location as indicated by the line number and the location of the field across the line. For example the first field on line 4 is 4|1. The next field on line 4, reading across from left to right is 4|2.

## Related Topics:

### Form Names

A complete listing of the Form Name for Appraise-It reports and addenda is available in the TRA.INI file located in the Windows directory. Simply scroll down to the section under **[Descriptions]**.

### Field Locations

Appraise-It provides a macro that makes the determination of ItemName locations fast and simple.

1. Start Appraise-It.

2. Open a new report.

3. Strike **Ctrl+M** simultaneously. The TRA macro filename dialog box appears.

4. Type **print.bas**.

5. Choose **OK.**

6. Notice that the field location numbers appear in the fields of the report.

 **Note:** If the locations do **not** appear, check to make sure that Auto Cell Transfer (Options: Advanced menu) is checked.

### An Example of ServiceName, TopicName, and ItemName

To provide the Property Address from the URAR, which is located on line 4, field 1 across, from a report located at C:\TRADATA\DATA\SAMPLE.RPT, use this information:

ServiceName = TRA

TopicName = C:\TRADATA\DATA\SAMPLE.RPT

ItemName = "ua2" | 4 | 1

### ServiceName, TopicName and ItemName for Other Applications

It is not within the scope of this documentation to provide DDE information for other Windows applications. Refer to the documentation or technical support for the specific application, or Microsoft® Windows for further information.

When searching other documentation for ServiceName, TopicName and ItemName, it is possible that the other application may use different terms for this identification system. In general, the three terms refer to three levels within the application program. The ServiceName may be the application name or an abbreviation for the application name, but this is not an absolute requirement. The TopicName further defines the application. The ItemName identifies the details within the topic name which are to be exchanged.

### Initiating and Terminating the Conversation

Once you know the ServiceName and TopicName for the application with which you wish to exchange information, you are ready to initiate (and then terminate) the conversation7.L8OZ. See Providing Appraise-It Data to Other ApplicationsPIO1QG, for examples of macros written to exchange information between Appraise-It and other application programs. Included as part of the Appraise-It macro is a Shell command which checks to see if the other application is open. See Shell (filename of application)1G2J0F. for additional information on this command. However, the actual conversation is initiated through the DDEInitiate command and terminated through the DDETerminate command.

### DDEInitiate (ServiceName-string, TopicName-string) DDETerminate (expression)

The DDEInitiate command starts the conversation7.L8OZ between the two programs and provides a channel through which the conversation occurs. The ServiceName is unique for the particular application program. The TopicName identifies the individual drive and path information for your files on your system. The DDETerminate command ends the conversation specified by the expression for the channel.

**Word for Windows Example:**ChanNumber! = DDEInitiate("WinWord", "c:\\cpp\\doc\\testdde.doc")
. . .
DDETerminate(ChanNumber!)

**Microsoft® Excel Example:**ChanNumber! = DDEInitiate("Excel", "c:\\excel\\xlddetst.xls")
. . .
DDETerminate(ChanNumber!)

Here is a macro a customer created to copy and paste information from cells 6|1 and 59|1 into row 5 columns 11 and 12 respectfully in Excel.

ChanNumber! = DDEInitiate("excel", "newledger.xls")

DDEPoke( ChanNumber!, "R5C11", "duq"|6|1 )

DDEPoke( ChanNumber!, "R5C12", "duq"|59|1 )

DDETerminate( ChanNumber! )

That's all the macro language he used to poke info from AI to Excel (The shell command didn't work with XP's long names). Excel needs to be open to "newledger".

WordPerfect® for Windows Example:
ChanNumber! = DDEInitiate("WordPerfect", "Commands")
. . .
DDETerminate(ChanNumber!)

The examples above use a single variable for the channel number and specify the DDEInitiate command with ServiceName and TopicName. Notice that for Word and Excel, the path requires double back-slashes (\\) to separate components. Notice also that the TopicName for WordPerfect for Windows DDE conversations uses a slightly different syntax by specifying "Commands" as the TopicName. The DDETerminate command uses the variable expression for the channel number to terminate the conversation.

## Conversational Exchanges

After the conversation7.L8OZ is initiated, the data exchange can occur. There are several means by which data exchanges can happen.

### DDE Request

For situations in which the client8.INLA application requests data from the serverWXJ36R application, a DDERequest command may be used. The data requested may be specific information from an individual field, or information such as the number of currently open reports, the identification of the current user, or the addenda included in a current report.

#### DDERequest(expression,TopicName-string)

Requests information from the serverWXJ36R via the channel in the first parameter, from the location specified by the TopicName string in the second parameter.

WordBasic Example:
Sub MAIN
ChanNumber = DDEInitiate("TRA", "c:\cpp\tradata\data\pugh.rpt")
a$ = DDERequest$(ChanNumber, **"**ua2|4|1**"**)
b$ = DDERequest$(ChanNumber, "ua2|159|1")
Insert a$
Insert b$
DDEExecute ChanNumber, "[RunMacro( ScrollToLine(110) )]"
DDETerminate(ChanNumber)
End Sub

The example above requests the information from fields 4|1, and 159|1 in the URAR (the property address and market value conclusion) and assigns the data to the string variables a$ and b$ respectively. **Note the placement of the quotations on the second parameter**. This placement is in accordance with WordBasic syntax, since Word is the client8.INLA and Appraise-It is the server in this example.

#### DDERequest with TopicName as Query

Sends a Query request to the serverWXJ36R. Currently, the only available option for the Query TopicName is "ActiveReports".

WordBasic Example:
ChanNumber! = DDEInitiate( "TRA", "Query" )
DDERequest$( ChanNumber!, "ActiveReports" )
DDETerminate(ChanNumber!)

The WordBasic example above uses DDEInitiate with "Query" as TopicName to start the DDE conversation7.L8OZ. Then a DDERequest is sent to TRA (as the server) to provide information about the currently open reports (ActiveReports). The conversation is terminated with a DDETerminate command.

Paradox® Example:
method pushButton(var eventInfo Event)
var
 d1, d2 DDE
 active, active2 AnyType
endVar

d1.open( "tra", "Query" )
d1.setItem( "ActiveReports" )
active = d1
msgInfo( "ActiveReports = ", active )
d2.open( "tra", "c:\\cpp\\tradata\\skels\\ua2skel\\test.rpt", "ua2|3|1" )
active2 = d2
msgInfo( "By Position = ", active2 )
d1.close()
d2.close()
endmethod

The Paradox example above uses "Query" as the TopicName in an Open command, then provides a DDE Request (active = d1) with ActiveReports as the TopicName to return the currently open reports in Appraise-It, the server.

#### Request Items

|  |  |  |
| --- | --- | --- |
| **Request:** | **TopicName** | **Information Returned:** |
| ActiveReports | Query | A comma separated list of all open reports, with the active report as the first item in the list |
| FileNo | File Name | The file number of the active report |
| "formtype"|Line|Across | File Name | The contents of the cell on form of type "formtype", in the cell indicated by "Line|Across" |
| ListForms | File Name | A comma separated list of all addenda included in the active report |
| MajorForm | File Name | The form type of the major form of the active report, e.g. "ua2", "704", "eval" |
| UserID | File Name | The current user's ID for the active form |

### DDE Poke

Another situation occurs when the client8.INLA application provides data to the serverWXJ36R application. The client can send the data to the server for the first time, or update information already sent to the server. For this situation, use the DDEPoke command.

### DDEPoke (expression, Server-ItemName-expression, data-expression)

Sends data via the channel specified in the first parameter, from the client8.INLA application to the serverWXJ36R application. The second parameter is a string expression for the location in the server to which the data will be provided, which is the server ItemName. The third parameter is a cell expression for the location in the client from which the data is taken.

Example:
DDEPoke(ChanNumber!, "Address", **"**ua2**"**|4|1)

The example above provides the information from the URAR, field 4|1, which is the property address, to a location in the server application (which is Word for Windows™ in this example) marked with the bookmark, "Address". Note that the cell expression places quotes only around the form name, "ua2". This is according to the syntax for **Appraise-It's macro language**, which is the client in this example.

## DDE Execute

Another type of DDE exchange occurs when a client8.INLA asks the serverWXJ36R to execute a command, such as a macro. In this situation, the appropriate command for the client to send to the server is DDEExecute.

## Related Topics:

### DDEExecute (expression, string)

Sends a command to the serverWXJ36R to execute. The first parameter is the channel, the second parameter is a string expression for the command to be executed. The second parameter is the command to be executed and may have an additional command with its own parameters nested within the original command.

WordPerfect Example:
DDEExecute(ChanNumber!, "MacroPlay(MacroName:amerge.wcm)")

Paradox Example:
DDEExecute(ChanNumber1, "[RunMacro(SplitScreen(\"pdoxwin.exe\",\"tra.exe\"))]

The examples above use DDEExecute to send a macro command to be executed. WordPerfect syntax requires the command "MacroPlay(MacroName:*filename for macro to be executed*)".

### Appraise-It Execute Items

|  |  |  |
| --- | --- | --- |
| **Execute Syntax:** | **TopicName** | **Action** |
| [RunMacro(- - - )] | File Name | Runs macro commands indicated by - - -. |
| [RunMacro(- - -.bas)] | File Name | Executes the macro commands in the file - - -.bas. |

## Providing Appraise-It Data to Other Applications

Just as it is necessary to determine the location in Appraise-It from which you are retrieving information, it is necessary to determine in the other application where the information will be retrieved. The methods for this retrieval vary from application to application. In general, you will want to insert the Appraise-It field at the location you wish in your other application.

Again, you should refer to your application's manual for information on how this is done for your specific application.

# Writing the DDE Macro

Once you know where to locate the information you wish to retrieve from Appraise-It, and how to insert the information in your application, you are ready to write the DDE macro.

### An Example of a DDE Global Macro for Word

Suppose you want to provide the property address and market value from the URAR to a document in Microsoft Word. In your particular situation, you want to execute Word on the F drive, then open a document called TestDDE.Doc on the C drive, CPP directory, and DOC sub-directory. Additionally, you want the Word bookmarks for the property address and market value to be called "Addrss" and "MktVal," respectively. The actual macro for the Appraise-It global response database looks like this:

exe$ = "f:\\winword\\winword.exe c:\\cpp\\doc\\testdde.doc"

shell( exe$ )

ChanNumber! = DDEInitiate("winword", "c:\\cpp\\doc\\testdde.doc")

DDEPoke( ChanNumber!, "Addrss", "ua2"|4|1 )

DDEPoke( ChanNumber!, "MktVal", "ua2"|159|1 )

DDETerminate(ChanNumber!)

Each line of the macro breaks down as follows:

exe$ = "f:\\winword\\winword.exe c:\\cpp\\doc\\testdde.doc"

Provides the drive and full path for executing Word plus the drive and full path for the Word document you wish to open"

shell( exe$ )

Provides the full path as well as any other components to the command line. **Shell checks if the exe$ is currently running: if the exe is not running, Shell starts it up; if the exe is currently running, Shell switches the focus to the specified exe.**

ChanNumber! = DDEInitiate("winword", "c:\\cpp\\doc\\testdde.doc")

Provides a channel number for the conversation7.L8OZ defined by DDEInitiate("winword", "drive and full path for Word document you wish to open.")

DDEPoke( ChanNumber!, "Addrss", "ua2"|4|1 )

DDEPoke( ChanNumber!, "MktVal", "ua2"|159|1 )

Provides (or pokes) to Word, via the channel number, to the Word "Bookmark" for the property address, the information specified in the "form name for URAR"|field location for property address.

Pokes to Word, via the channel number, to the Word "Bookmark" for the Market Value, the information specified in the "form name for URAR"|field location for market value.

In Word, the document which retrieves the DDE information would have Word Bookmarks inserted for Addrss and MktVal at the exact location to which the Appraise-It information should be provided.

For example:

Lender
Lender Address

Re: (*Insert* ***Addrss*** *Bookmark here*)

Dear Lender:

As requested, the above property was duly inspected and found to have an estimated market value of $(*Insert* ***MktVal*** *Bookmark here*).

If you have any questions on the enclosed appraisal report, please do not hesitate to contact me.

Sincerely,

Each field of information from Appraise-It is transferred to Word via a DDEPoke command which includes the Appraise-It ItemName and a Word Bookmark name. The Appraise-It ItemName provides the Appraise-It form name and field location for the information. The Word Bookmark name is created to designate the location in the Word document for the Appraise-It information.

DDETerminate(ChanNumber!)

Finally, the DDE conversation is terminated using the DDETerminate command with the variable expression for the channel number.

### An Example of a DDE Global Macro for Excel

Suppose you want to provide the borrower name and market value from the URAR to a document in Microsoft Excel. In your particular situation, you want to execute Excel on the F drive, then open a worksheet called XLDDETST.XLS on the C drive, Excel directory. Additionally, you want the Excel locations for the borrower's name and market value to be "R11C1" and "R11C5," respectively. The actual macro for the Appraise-It global response database looks like this:

exe$ = "f:\\excel\\excel.exe c:\\excel\\xlddetst.xls"

shell( exe$ )

ChanNumber! = DDEInitiate("excel", "c:\\excel\\xlddetst.xls")

; addresses

DDEPoke( ChanNumber!, "R11C1", "ua2"|7|1 )

DDEPoke( ChanNumber!, "R11C5", "ua2"|160|1 )

DDETerminate( ChanNumber! )

Each line of the macro breaks down as follows:

exe$ = "f:\\excel\\excel.exe c:\\excel\\xlddetst.xls"

Provides the drive and full path for executing Excel plus the drive and full path for the Excel worksheet you wish to open"

shell( exe$ )

Provides the full path as well as any other components to the command line. Shell checks if the exe$ is currently running: if the exe is not running, Shell starts it up; if the exe **is** currently running, Shell switches the focus to the specified exe.

ChanNumber! = DDEInitiate("excel", "c:\\excel\\xlddetst.xls")

Provides a channel number for the conversation7.L8OZ defined by DDEInitiate("excel", "drive and full path for Excel document you wish to open.")

DDEPoke( ChanNumber!, "R11C1", "ua2"|7|1 )

DDEPoke( ChanNumber!, "R11C5", "ua2"|160|1 )

Provides (or pokes) to Excel, via the channel number, to the Excel "cell reference location for borrower's name", the information specified in the "form name for URAR"|field location for borrower's name.

Pokes to Excel, via the channel number, to the Excel "cell reference location for market value", the information specified in the "form name for URAR"|field location for market value.

In Excel, the worksheet which is to receive the DDE information looks like this:

{bmc bm0.BMP}

DDETerminate( ChanNumber! )

Finally, the DDE conversation is terminated using the DDETerminate command with the variable expression for the channel number.

### An Example of a DDE Global Macro for WordPerfect® for Windows

In WordPerfect, the macro procedure is a little different. One of the simplest methods to use with WordPerfect is to perform a file merge combining a WordPerfect primary merge file with a secondary merge file that contains the Appraise-It data.

To perform the merge, the Appraise-It data must first be written to a WordPerfect file. This file with the Appraise-It data becomes the WordPerfect secondary merge file. You will also need to create a primary merge file with field locations for the Appraise-It data. Then a WordPerfect macro is executed which merges the secondary and primary WordPerfect merge files. See your WordPerfect documentation for more information about creating merges.

Suppose you want to provide the street address and legal description from the URAR to a document in WordPerfect for Windows. The Global Macro which you write for Appraise-It requires an Open # and Print # command to write the secondary merge file. Next a DDEExecute command is sent to run the merge macro within WordPerfect.

The actual macro for the Appraise-It global response database which executes WordPerfect on your C drive, creates a secondary merge file called Amerge2, and then executes a WordPerfect macro called Amerge.WCM, looks like this:

shell("c:\\wpwin\\wpwin.exe")

Open "c:\\wpwin\\amerge2" for output as #1

Print #1, "Start Record"+Chr$(10)

Print #1, Char$(9)+Cell(4,1)+Chr$(10)

Print #1, Char$(9)+Cell(5,1)+Chr$(10)

Print #1, "End Record"

Close #1

ChanNumber! = DDEInitiate("WordPerfect", "Commands")

DDEExecute(ChanNumber!, "MacroPlay(MacroName:Amerge.WCM)")

DDETerminate(ChanNumber!)

The lines of the macro break down as follows:

shell("c:\\wpwin\\wpwin.exe")

Provides the drive and full path for executing WordPerfect. Shell checks if the exe$ is currently running: if the exe is not running, Shell starts it up; if the exe **is** currently running, Shell switches the focus to the specified exe.

Open "c:\\wpwin\\amerge2" for output as #1

Print #1, "Start Record"+Chr$(10)

Print #1, Char$(9)+Cell(4,1)+Chr$(10)

Print #1, Char$(9)+Cell(5,1)+Chr$(10)

Print #1, "End Record"

Close #1

Opens the WordPerfect secondary merge file and writes the Appraise-It data to it. "Start Record" and "End Record" are used as record delimiters in the secondary merge file. Char$(9) and Char$(10) are Tab and Carriage Return respectively, and are used as field delimiters in the secondary merge file. After the secondary merge file is written, the file is closed.

ChanNumber! = DDEInitiate("WordPerfect", "Commands")

Provides a channel number for the conversation7.L8OZ defined by DDEInitiate("WordPerfect", "Commands").

DDEExecute(ChanNumber!, "MacroPlay(MacroName:Amerge.WCM)")

Executes the WordPerfect macro, Amerge.WCM. This macro looks like:

{bmc bm1.BMP}

Note the embedded codes for the field delimiters in the Reveal Codes portion of the WordPerfect window. The WordPerfect primary merge file looks like this:

{bmc bm2.BMP}

# Response Database Macros

Appraise-It provides you with three avenues through which you may send Basic language instructions. You may provide Basic instructions through macros written in the Common Response Database and the Global Response Database, or through Basic language macro files which you direct Appraise-It to access. These macro files may be maintained in Appraise-It's TRA directory, or in another directory.

## Writing Response Database Macros

The general instructions for writing a macro are similar for both the Common Response Database and the Global Database.

## To Write a Response Database Macro

1. Have the Appraise-It report open.

2. Choose Database:

 For the Global Response Database, choose **Database**, then **Global.**

 For the Common Response Database, choose **Cell** then **Common Response.**

3. Select **Add**.

4. Enter Clipboard Name, then **OK**.

5. Enter macro.

6. Check Macro checkbox.

7. Select Hot Key, if desired.

8. Choose **OK**.

See *The User's Guide*, **Chapter 11**: Macros, for detailed instructions on writing a macro for Appraise-It. Additionally, Chapter 11 provides sample macros as well as a brief Basic command reference.

## Macro Files

You may feel more comfortable working with separate macro files, instead of using Appraise-It's Response Databases. If so, simply create macro files using your usual text editor. Save your Basic instructions in these files and Appraise-It provides a quick method to retrieve them.

## Executing Macro Files

Appraise-It allows you to access your macro files using **Ctrl + M**:

1. Have the Appraise-It report open.

2. Enter **Ctrl + M**.

3. Enter path/filename for macro file.

** If the macro is in the TRA directory, simply enter the filename.**

 If the macro is in a directory other than TRA, enter the path and filename.

4. Choose **OK.** Your macro is executed.

## Introduction

From the start, Appraise-It was designed to be a flexible component of an integrated forms processing system. You can expect to see more and more standard component software evolving over the next few years. This component software is easily integrated with other software applications. Appraise-It supports many existing forms of component technology today. Appraise-It's existing DDE capability, working together with an advanced macro language offers users complete data import and export capability to any DDE-enabled application. Additionally, Appraise-It's customizable menu system allows you to **add your company-specific macros directly to Appraise-It's menu**.

When a new product is released that integrates data with Appraise-It, a simple addition to the TRA.INI file quickly gives the user instant access to this new product through Appraise-It's **Tools** menu. The Tools menu is almost completely built from custom entries found **in our TRA.INI file**. These customizations provide the integrating interfaces between our Windows products via macro language and DDE. Tools like Microsoft Mail™ are enabled using our macro language's capability to register and call nearly any Dynamic Link Library.

# Custom Commands

## Custom Commands in TRA.INI

{bmc bm3.BMP}

The Tools menu as shown above is almost completely built from custom entries found in the Custom Command section of the TRA.INI file.

You may view and edit the existing custom commands in the TRA.INI file using your favorite text editor.

{bmc bm4.BMP}

The entries in the Custom Commands section of the TRA.INI file are in the form:

TextIdentifier=MenuName,MenuText,BasicScriptFile,AcceleratorKey(optional)

## Before You Add the Custom Command

Suppose you wanted to perform a type of cost analysis of your report data.

1. Write a cost analysis routine using, for example, Microsoft Access™ or Excel™.

2. Write an Appraise-It macro to pass the desired data from the report into the analysis routine via DDE.

3. Perform the analysis, and if you wish, Poke the data back into your report.

4. Decide whether you want to run the macro as a global or common response, or if you want to add the command to the Tools menu.

## Adding a New Custom Command to the Tools Menu

1. Save the macro script as a file, such as NEWTOOL.BAS.

2. Open the TRA.INI file in your favorite text editor.

3. Modify the Custom Commands section to include this line:

**NewTool=Tools,Cost Analysis,NewTool.bas**

 where NewTool is the custom command, Tools is the menu the command appears on, Cost Analysis is the menu item, and NewTool.bas is the macro script file.

4. Save the TRA.INI file.

**Marty=Tools,Marty Ctrl+Shift+R,Marty.bas,Ctrl+Shift+R**

**The first short cut is will be displayed, the second one will be executed.**

## Using Macros to Provide Custom Commands

Appraise-It's powerful macro language allows you to write effective scripts that will perform almost any type of task. Macros can be implemented as common responses, global responses, or stand alone scripts that can be executed **as menu items or by the Ctrl+M** key combination.

## Related Topics:

### Suggested Uses for Macros

 Use to register and call almost any Windows Dynamic Link Library, including any you write

 Use to manipulate report data via DDE and internal referencing schemes

 Use to add and retrieve any graphical images in a report

 Use to execute any menu command

 Use to write advanced scripts using FOR and WHILE loops

 Use to provide built-in WritePrivateProfileString and GetPrivateProfileString functionsDMM99R

 Use to provide Message Box function

### Introduction

The Appraise-It macro language is similar to other Basic languages with which you may be familiar. At the same time, it contains many unique commands and identifiers which apply only to the Appraise-It program. Once you are acquainted with these unique commands and identifiers, you will be well on your way to providing macros that accommodate your own specific circumstances.

# Variables

Different variable structures are used to represent different data types. The following are the Basic type representations used in this package:

|  |  |  |
| --- | --- | --- |
| **Structure** | **Type** | **Example** |
| *Identifier* | Real variable | ID |
| *identifier***$** | String variable | ID**$** |
| *identifier***%** | Integer variable | ID**%** |
| *identifier***&** | Long variable | ID**&** |
| *identifier***!** | Single variable | ID**!** |
| *identifier***#** | Double variable | ID**#** |

## User Information

## Information from Registration Procedure

Registering your Appraise-It product provides these variables:

|  |  |
| --- | --- |
| **Variable Name** | **Description of Information Provided During Registration Procedure** |
| Company$(Name) | Company Name |
| Company$(Address1) | Company Address Line 1 |
| Company$(Address2) | Company Address Line 2 |
| Company$(Phone) | Company Phone Number |
| Company$(TaxID) | Company Tax ID Number |
| Appraiser$(Name) | Appraiser's Name (current user) |
| Appraiser$(ID) | Appraiser's ID Number (current user) |
| Appraiser$(License) | Appraiser's License # (current user) |
| Appraiser$(State) | Appraiser's State (current user) |
| Appraiser$(Expire) | Appraiser's License Expiration Date (current user) |
| Appraiser$(Certification) | Appraiser's Certification # (current user) |

## Pre-defined Keywords

Some terms and variables are already defined in the language. Here are the variables that are pre-defined in the language.

### Across

Across represents the location of the current field, as numbered from left to right across the current line.

### Checkbox

Checkbox is a field type identifier used to identify a checkbox.

### Keypress

Keypress identifies when a key has been pressed on the keyboard.

### Leave

Leave identifies that the editor has left a field location and may be used as the second parameter to a Call function.

### Line

Line represents the vertical location of the current field, as determined by numbering the lines down from the top of the form.

### Multi

Multi is a field type identifier used to identify a multi field.

### Multihandle

Multihandle is a field type identifier used to identify any line but the first line of a multi-line field.

### Single

Single is a field type identifier used to identify a single field.

### Totallines

Totallines indicates the total number of lines in a report.

# Cell Representation

A cell can be represented in different forms:

|  |  |
| --- | --- |
| **Current Form** | **Example** |
| Cell ( *expression, expression* ) | Cell(14, 1) |
| Cell ( *string-type*, *expression, expression* ) | Cell("ua2", 14, 1) |
| *integer-type* | *integer-type* | 14|1 |
| *string-type* | *integer-type* | *integer-type* | "ua2"|14|1 |
| Cell ( *string-type | string-type )* | Cell( " offscrn ", " File# " ) |

## Cell or Field

Cell and field are used interchangeably to indicate an area in which information is stored or where calculations are done. A field may be represented by two keyword variables:

 Line indicates the line number of the field, or how far down from the top of the form the field is located.

 Across indicates the location of the field as numbered across from left to right on the current line, or how many fields over from the left of the form the field is located.

## Special Syntax

## Single and Double Slashes

The  **\** character is used to represent special of non-printable characters.

Use two slashes **\\**  to represent a single slash character.

Example:
 "C:\\AUTOEXEC.BAT"

## Operators

### Math Operators

Appraise-It supports these math operators:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| \* | Multiplies |
| / | Divides |
| + | Adds |
| - | Subtracts |
| ^ | Raises the number to the power of the exponent |

### Comparison Operators

Appraise-It supports these operators which are used to compare expressions:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| <> | Not equal to |

### String Operators

Appraise-It supports these operators which are used in string concatenations, equivalencies, and comparisons.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Concatenates  |
| = | Equal to (not case sensitive) |
| <> | Not equal to (not case sensitive) |

### Procedures, Functions, and Sub Procedures

In general terms, proceduresE786T7 are portions of a program that perform tasks. FunctionsDMM99R are procedures that result in a value. Sub procedures2SHPXO do not result in the return of a value.

### Pre-defined Functions

Appraise-It has certain functionsDMM99R which have already been defined.

#### Abs (number-expression)

Returns the absolute value of the number indicated as the parameter.

Example:
 If Abs(Val(2|1))>=
 (Val(2|4)/100) Then
 Call(2|6, Calc)
 End If

The example above uses Abs to calculate the absolute value of 2|1. The If-Then statement uses the Abs function to calculate whether the absolute value of 2|1 is greater than or equal to the value of 2|4 divided by 100. If so, then 2|6 is called for the Calc selection.

#### ActivateApp()

Returns focus and control to Appraise-It.

#### ActiveField (cell-identifier)

Returns true if the parameter is the current field.

Example:
If ActiveField(14|6) Then
 A$="N"
ElseIf ActiveField(14|7) Then
 A$="L"
ElseIf ActiveField(15|9) Then
 A$="I"
End If

The example above uses If-then statements to cover situations in which this code will be Run from other field locations. These other situations are handled by specifying A$ variable identities according to the ActiveField location, which is the location from which the code is Run.

#### AddAttachment%(identifier-string, filename-string, description-string)

Attaches a file to the report which is embedded with the file at save time and is transmitted with the report via e-mail, EDI, etc. AddAttachment% returns a 1 if the attachment is successful, or a 0 for an unsuccessful attachment.

Example:
AddAttachment%( , , )

The example above uses AddAttachment% to attach the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ to the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. The third parameter is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. AddAttachment% returns a 1 for a successful attachment and a 0 for an unsuccessful attachment.

#### Call (cell-identifer, selection-expression)

Performs the code contained in the cell indicated by the first parameter, using the expression given in the second parameter as the called cell's Selection value. **Call** performs this code using the called cell's location, not the current cell's location.

Example:
Fields 36|1, 36|2, and 36|3:
Select Case Selection
 Case Leave
 Call(Line|4, Calc)
End Select

36|4:
Select Case Selection
 Case Calc
 = Val(Line|1)\*Val(Line|2)\*Val(Line|3)
 Call(18|1, Calc)
End Select

The example above Calls the calculation case of 36|4 from all three fields, 36|1, 2, and 3. Field 36|4 calculates a multiplication formula for the values from the first three fields, and then Calls still another field, 18|1, whose value depends on 36|4. Call is the appropriate function choice for the first three fields, because the calculation needs to be performed and provided in field 36|4 using 36|4's Calc selection (as opposed to being Run using the current cell's selection). If fields 36|1 through 36|3 contained the Run function instead of Call, the multiplication calculation would replace the values in each of the first three fields.

#### Call (filename)

Performs the code contained in the file indicated by the parameter.

Example:
**Call("script.bas")**
Call("script.bxe")

The example above uses Call("script.bas") to perform the source code, and Call("script.bxe") to perform the compiled code.

#### Checked

Returns true if the check box at the current location contains a value.

Example:
If Checked Then
 Call(13|1, Calc)
End If

The example above performs the calculation in 13|1 when the currently active checkbox contains a value.

#### CheckedorYes (cell identifier)

Returns true if checkbox contains an X or a Y.

Example:
If CheckedorYes(1|2) Then
 Call(13|1, Calc)
End If

The example above Calls the Calc case of 13|1 if field 1|2 contains an X or a Y.

#### *Chr$(expression)*

Returns a string which is the ANSI character for the parameter.

Example:
String$ = Chr$(65) + Chr$(66)

The example above yields the ANSI character for 65 concatenated with the ANSI character for 66, or AB. Refer to an ANSI character table for additional information on the ANSI codes 0 - 255.

#### *ClearComp(integer)*

Clears the comps data in the market grid at the location specified by the integer value.

Example:
ClearComp(2)

The example above clears the data contained in Comparable #2 of the market grid.

#### *Close #(integer-expression)*

Closes an open file. The integer-expression refers to the file to be closed, which was opened using the Open command. After the file is closed, additional data cannot be written to the file unless it is opened again. Closing the file releases the integer that identifies the open file for use again with other files.

Example:
Open "c:\\tra\\urar.txt" For Output As #1
Print #1, "Check Age", "Check $" + Chr$(13) + Chr$(10)
Print #1, "Date"
Close #1

The example above opens urar.txt, writes information to it, then closes the file.

See Open (filename) For Append As # (integer)/
Open (filename) For Output As # (integer)6I0X6N and Print # (integer, string-expression-list)136WMZC for additional information.

#### *Comp$ (integer-expression)*

Returns a string expression that is the name of the form or addendum that contains the Comps information indicated by the integer expression. Each time Comp$ is called, a prompt appears asking you to select the type of Comp.

Example:
Comp1Form$ = Comp$(1)
Comp5Form$ = Comp$(5)

The example above uses Comp$ (1) to return the string for the form that contains Comp 1 information, or the "ua2". Comp$ (5) returns the string for the form that contains Comp 5 information, or "cmps46". In this example, two prompts appear, asking you to select the type of Comp.

#### *CompT$ (integer-expression)*

Returns a string expression that is the name of the form or addendum that contains the Comps information for the comparable indicated by the integer expression, but which is also of the same comparable type selected by the last Comp$ command. Unlike Comp$, CompT$ does not provide a prompt asking you to select the type of Comp.

Example:
Comp1Form$ = Comp$(1)
Comp5Form$ = CompT$(5)

The example above uses CompT$(5) to return the string for the form that contains Comp 5 information of the comparable (rental, sales) type provided for Comp$(1).

#### *Compile (source filename, destination filename)*

Compiles the source file indicated by the first parameter into a destination for the compiled file indicated by the second parameter.

Example:
Compile("script.bas", "script.bxe")

The example above compiles the source code in Script.BAS into the compiled code in Script.BXE. Note that the Script.BXE is a larger file than Script.BAS, but runs faster.

#### *Constant$(report name)*

Returns the full path of the current report indicated by the parameter, which is the report name.

Example:
If Constant$("SketchApp") <> "not found" then
 SketchFile$ = Constant$("SketchData") + "\\" +
 cell("Offscrn", "File#") + ".SKT"

The example above, when the Sketch Application is current, provides the full path of the current sketch file, as determined on the user's system.

Constant$ supports these parameters:

|  |  |
| --- | --- |
| **Directory** | **Description** |
| PhotoApp | directory where Wilson's photo exe is stored |
| PhotoMem | directory where Wilson's photo memory files are stored |
| PhotoUser | directory where Wilson's data are stored |
| SketchApp | directory where the sketch application is stored |
| SketchData | directory where the sketch data files are stored |
| ReviewApp | directory where the review program is stored |
| TRAApp | our application directory which by default is c:\tra |
| TRAData | our data directory which by default is c:\tradata\data |
| TRAPriv | our personal user directory which by default is c:\trapriv |

#### *CopyComps (source-integer, destination-integer)*

Copies the comps data in a market grid from the source to the destination indicated by the integer values.

Example:
CopyComps(2, 5)

The example above copies the comps data at Comparable #2 of the market grid to Comparable #5 of the market grid.

#### *Date (expression)*

Returns the numeric version of the system date. This is passed as the first parameter to Format$ (expression, string-expression). See also Format$ (date-expression, string)2MBHYV7.

Example:
NewDate!=Date
2|1=Format$(NewDate!, "mmmm d, yyyy")

The example above assigns NewDate! as a numeric version of the system date which is in turn passed to Format$ and converted to, for example, January 1, 1995.

#### *Date$ (expression)*

Returns the current system date in a 10-character string of the form mm-dd-yyyy, where mm is the month (01-12), dd is the day ( 01-31 ), and yyyy is the year (1980-2099). This is the equivalent of Format$(Now, "mm-dd-yyyy"). See also Format$ (date-expression, string)2MBHYV7.

Example:
UserDate$=InputBox("Enter the Valuation Date.", "TRA", Date$)

The example above creates a dialog box titled TRA, with the prompt, "Enter the Valuation Date." Date$ provides the current system date, such as 01-01-1995, as the default response.

#### *DateValue (string)*

Returns the numerical value of the date which has been represented as a string expression. See also Format$ (date-expression, string)2MBHYV7.

Example:
DeltaMonth!=DateValue(UserDate$)
DeltaMonth!=DeltaMonth! - DateValue(This)

The example above takes a numeric value for the string UserDate and uses that value in a calculation in the next line of code.

#### *DeleteFile% (filename)*

Returns 1 if the file specified in the parameter has successfully been deleted, otherwise the function returns 0.

Example:
a%=DeleteFile%(X12File$)

The example above returns a%=1 if X12File$ is successfully deleted, or 0 if, for example, the file does not exist.

#### *EmbedImages%(state%)*

Embeds the associated images of a report upon saving if state% is 1. EmbedImages%(0) turns off embedding of images for the report.

Example:
EmbedImages%(1)
SaveReport( )

The example above embeds the associated images of a report with the saved report.

Example:
EmbedImages%(0)

The example above turns off the feature which embeds the associated images of a report with the saved report.

#### *FieldCount (expression)*

Returns the number of fields on the line given by the parameter.

Example:
Sub CopyLine(Dst%, Src%)
 J%=1
 While J%<=Fieldcount(Dst%)
 If FieldType(Cell(Dst%, J%))<>MultiHandle Then
 Cell(Dst%, J%)=Cell("ua2", Src%, J%)
 End If
 J%=J%+1
 Wend
End Sub

The example above uses FieldCount to provide the number of fields in the destination line. While the number of fields in the destination line is less than or equal to those in the source line, the sub procedure will copy the field contents of the line from the source to the destination.

#### *FieldType (cell-identifier)*

Returns the type of field (MultiHandle, CheckBox, Multi, Single) of the parameter.

Example:
If FieldType(Cell(Dst%, J%))<>MultiHandle Then
 Cell(Dst%, J%)=Cell("ua2", Src%, J%)
End If

The example above uses FieldType to determine if the destination specified by the cell identifier is other than MultiHandle.

#### *FileExists% (filename)*

Returns 1 if the file specified in the parameter exists, otherwise the function returns 0.

Example:
If FileExists% (SketchFile$) = 0 then
 SketchFile$ = ""
End If

The example above uses FileExists% to determine if a file named SketchFile$ already exists.

#### *ForceAutoSave*

Forces the active report to be auto saved. This is useful when shelling out to another process, ensuring that a recovery file is created if anything goes wrong.

Example:
SendCommandMessage( 24334 ) ;File Save As
ForceAutoSave

The example above provides the Save As dialog box, followed by a forced auto-save.

#### *Format$ (date-expression, string)*

Returns a date in a numeric format. The first parameter is the date and the second parameter is a string that tells the function in what format the date should be returned. See also Date$ (expression)CDAB8B.

Example:
NewDate!=Date
2|1=Format$(NewDate!, "mmmm d, yyyy")

The example above assigns NewDate! as a numeric version of the system date which is in turn passed to Format$ and converted to, for example, January 1, 1995.

Appraise-It supports these Date Formats:

|  |  |
| --- | --- |
| **Format** | **Provides Data As:** |
| "mm-dd" | 09-08 |
| "mm-dd-yy" | 09-08-95 |
| "mm-dd-yyyy" | 09-08-1995 |
| "mm-yy" | 09-95 |
| "mm-yyyy" | 09-1995 |
| "mmm-yy" | Sep-95 |
| "mmm-yyyy" | Sep-1995 |
| "mmmm d" | September 8 |
| "mmmm d, yyyy" | September 8, 1995 |
| "mmmm, yy" | September, 95 |
| "mmmm, yyyy" | September, 1995 |
| "mm/dd/yy" | 09/08/95 |
| "mm/dd/yyyy" | 09/08/1995 |
| "mm/yy" | 09/95 |
| "mm/yyyy" | 09/1995 |
| "yy" | 95 |
| "yyyy" | 1995 |

#### *Function*

Defines a function procedure by its name and arguments and then provides the code that constitutes the function.

Example:
Function Adjustment!(A%)
 Select Case Sign(A%)
 Case 1
 Adjustment! = SalesPrice! - Adjustment!
 Case 0
 Adjustment! = SalesPrice!
 Case -1
 Adjustment! = SalesPrice! - Abs(Adjustment!)
 End Select
Exit Function

The example above defines a function called Adjustment! and calculates the value for the adjustment for positive, negative and zero adjustment values.

#### *GetAttachment$(identifier-string)*

Returns the filename associated with the attachment identified in the parameter.

Example:
Location$ = GetAttachment$("location")

The example above provides the filename for the location map to Location$.

#### *GetAttachments$(flag %)*

Retrieves a comma-separated list of attachments. If flag% is 0, a list of file names alone is returned. If flag% is 1, identifier|filename is returned.

Example:
a$ = GetAttachments$(0)

The example above from a report with two attachments - \_\_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_\_\_\_\_ returns this list to a$:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Example:
a$ = GetAttachments(1)

The example above from a report with two attachments of \_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_\_ returns this list to a$:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#### *GetImage$ (identifier)*

Returns the filename associated with the image specified by the identifier.

Example:
A$ = GetImage$("CMP\_4\_4X6"))

The example above returns in A$ the filename, D:\PHOTOMGR\JPG\NK101202.JPG for the image identified by "CMP\_4\_4X6". See also GetImages$ ( flag expression )IPOWH. .

#### *GetImages$ ( flag expression )*

Returns a list of the filename and/or identifier for all image objects in a report. Specifying (0) as the flag expression returns filenames only. Specifying (1) returns identifier|filename.

Example:
a$ = GetImages$(0)

The example above from a report with two photographs of comp 4 and comp 5, returns this filename list in a$: D:\PHOTOMGR\JPG\NK101202.JPG;D:\PHOTOMGR\JPG\NK102736.JPG;

Example:
a$ = GetImages$(1)

The example above from a report with two photographs of comp 4 and comp 5, returns this identifier|filename list in a$: CMP\_4\_4X6|D:\PHOTOMGR\JPG\NK101202.JPG;
CMP\_5\_4X6|D:\PHOTOMGR\JPG\NK102736.JPG;

See also GetImage$ (identifier)CK5539 .

#### *GetPPE$ ( section string, match string, filename )*

Get Private Profile Entry uses a pattern match to find an exact entry in an INI section. GetPPE returns the first entry in the specified section of the INI file that matches the pattern.

Example:

1.PM:

[PHOTOS]
SUBJECT{­FRONT VIEW}=1958 WOODCHASE BOULE
 ~D:\PHOTOMGR\JPG\NK102736.JPG
SUBJECT{­REAR VIEW}=1958 WOODCHASE BOULE
 ~D:\PHOTOMGR\JPG\OFSS4220.JPG
SUBJECT{­STREET SCENE}=1958 WOODCHASE BOULE
 ~D:\PHOTOMGR\JPG\OFSS4954.JPG
COMP1{­FRONT VIEW}=2001 WOODCHASE BOULE
 ~D:\PHOTOMGR\JPG\OE3A4536.JPG
COMP2{­FRONT VIEW}=1002 WEST POINTER
 ~D:\PHOTOMGR\JPG\NK1O1202.JPG
COMP3{­FRONT VIEW}=4567 PINEY POINT
 ~D:\PHOTOMGR\JPG\OE3A4256.JPG
GENERAL 001 TOP{­VIEW FROM PROPERTY}=4567 PINEY POINT
 ~D:\PHOTOMGR\JPG\NINQ3722.JPG

a$ = GetPPE$ ("PHOTOS", "GENERAL 001 TOP",
 Constant$("File#")+".PM)

The example above when executed on 1.PM returns to a$ the string "GENERAL 001 TOP{­VIEW FROM PROPERTY}

#### *GetPPS$(section string, entry string, default string, filename)*

Get Private Profile String (similar to the Windows API call) returns the item in an INI file that matches the section string and entry string, otherwise it returns the default string.

Example:

Win.INI file:

[Extensions]

cal=calendar.exe ^.cal
crd=cardfile.exe ^.crd
trm=terminal.exe ^.trm
txt=notepad.exe ^.txt

MVB=MVIEWER2.EXE ^.mvb
PDF=C:\ACROEXCH\ACROEXCH.EXE ^.PDF
PDS=C:\ACROEXCH\ACROEXCH.EXE ^.PDF

acrodir$ = GetPPS$("Extensions", "PDF", "N/A", "WIN.INI"

The example above returns in Acrodir$ the item in the Extensions section of the Win.INI file that match "PDF" or C:\ACROEXCH\ACROEXCH.EXE ^.PDF

See also WritePPS% ( section string, entry string, filename )FX1KIC.

#### *GetRequest (global variable identifier)*

Retrieves data placed in a global variable. Data placed in these global variables remain resident until the application terminates and may be accessed by any other macro session or via DDE.

Example:
GetRequest("Glob\_Ident")

The example above returns any text assigned to the global identifier "Glob\_Ident".

Note: The data in "Glob\_Ident" may also be accessed via DDE by setting the topic to "System" and performing a DDERequest on "Glob\_Ident".

See also DDE Request8YOXTG and SetRequest (global variable identifier211WWB3.

#### *Go Command (cell-identifier)*

Moves the editor to the cell specified as the parameter.

Example:
If Selection=Keypress Then
 Go(16|9)
End If

The example above moves the editor from 14|6 to 16|9 if a key is pressed. Or you could simply use Go(16|9) and assign it to a key.

#### *Go Command (alternative)*

Note, you can enter text in a cell without going to it. For example,

14|2="Intended User-Any VA approved Lender"

This is a macro we created for VA to enter this text ("Intended User-Any VA approved Lender") in cell 14|2 without evoking the Lender\Client box. Since there is no macro for closing dialogue boxes, this avoids the pop up box.

P.S. If you wanted to send text to a cell on the report profile without going to it or opening it, you could use something like:

cell( "offscrn", 15,1) = "Intended User-Any VA approved Lender"

cell( "offscrn", 16,1) = " "

cell( "offscrn", 17,1) = " "

This would enter the text in cell 15,1 and wipe out any text in cells 16,1 and 17,1.

One reason for using this alternative is to avoid the pop up dialog box, which cannot be closed with a marco per Profit:

I found a function which is for sending messages to dialogs, and I was trying to see whether I could do something equivalent to pushing the Cancel button -- or hitting the Escape key -- on the Lender/Client Address dialog.  I wanted to close the dialog without changing the address information.

    The macro editor accepted the SendDlgItemMsg() call without raising an error, but the code had no effect.  The function takes four arguments.  I played around with those for a while trying to get something to work.  Nothing did.

    Eventually I figured out that I was wasting my time.  Whether or not this function is valid, and whether or not I was calling it with the right arguments, you cannot use FrankBasic to send any kind of message to this dialog.  The reason is that the dialog is modal.  Appraise-It stops processing while the dialog is visible.  Since the dialog is called in the Arrive event of the lender address cell, the Go call does not return until after the dialog has been dismissed.  You can test that yourself by putting a message box in your macro.

    Go( 14|2 )

    MsgBox "Gone"

The message will not appear until you get rid of the LenderAddress dialog.  Any other line of code you put there will also fail to execute while the dialog is displayed.

Note, you can use the equal sign without specifying the cell. For example.

="The subject neighborhood is a exclusive golf and lake front community on Lake Norman. The golf course is designed by Greg Norman. The subject is located in a water front residential community on Lake Norman about 6 miles west of the city of Mooresville."

But you are limited to 250 characters for this equal command. But with an =this+ you can extend the character limit to almost 2000 characters. For example,

**="**The subject neighborhood is a exclusive golf and lake front community on Lake Norman. The golf course is designed by Greg Norman. The subject is located in a water front residential community on Lake Norman about 6 miles west of the city of Mooresville."

**=this+"** The subject neighborhood has good market appeal and competes favorably with other neighborhoods in the subject market area. Employment centers are easily accessible and commute times during peak traffic periods are considered reasonable."

**=this+"** Schools, parks and shopping are all close by. The stability of employment is considered average. No unfavorable conditions were observed which would adversely affect value or marketability."

#### *HasAddendum%*

Allows you to check for the existence of an addendum before accessing it.

Example:
if HasAddendum%("offscrn") then
 Launch("offscrn")
end if

The example above checks for the offscreen addendum (or the Report Profile) and if one exists, launches that addendum.

#### *If...Then...Else*

Performs the instructions if the stated condition exists or is true. Else provides for alternative conditions.

Example:
If Report$="ua2" Then
 8|1="ua2"|101|1
Else
 **Go(156|1)**
End If

The example above uses If-Then statements to determine if the form is the URAR, and if so, transfers the contents of 101|1 of the URAR to the addendum field 8|1. If the form is not the URAR, the editor goes to 156|1. [But for addendum use Formname$() rather than Report$]

#### *InputBox$ (prompt-string, title-string, default-response-string)*

Creates a dialog box with a prompt, a title, and a default response.

Example:
Insert(InputBox$("What is the zip code?", "Zip Macro", "99999"))

{bmc bm5.BMP}

The example above creates the input box as shown above. Choosing OK inserts the text 99999 into the field.

#### *Insert (string)*

Inserts a string into the editor. If the field already contains data, Insert places the macro information at the location of the insertion point, leaving the existing data intact. See Replace(string)5O41.0 if you wish to replace the existing data, instead of inserting the macro data in addition to the information that already exists.

Example:
Insert("W. Palm Beach")

The example above provides the string expression "W. Palm Beach" at the current field location.

#### *InStr% (start, string, sub-string)*

Returns the position at which the second string occurs within the first string

Example:
BreakPoint = InStr%(1, FullAddress$, ",")

The example above uses InStr% to return the position at which the first comma occurs in a string expression that consists of the full street address, city, state.

#### *Launch ( string )*

Opens the form named in the parameter. See LoadExtended1JBP\_LJ.

Example:
{­
 MenuItem "&Extended Ctrl+E"
 MenuCode
 {­
 Launch("ua2")
 LoadExtended
 }
}

The example above uses Launch to open the URAR form specified by the string parameter.

#### *LaunchComments*

Enables/disables automatic launching of the extended comment window in a macro if data exceeds field space.

Example:
LaunchComments **0**

The example above **disables** the automatic launching of the extended comment window.

Example:
LaunchComments **1**

The example above **enables** the automatic launching of the extended comment window.

#### *Left$ (string, length)*

Returns a string expression of the characters to the furthest left in the string expression. The number of characters returned is determined by the length integer expression.

Example:
BreakPoint% = InStr (1, FullAddress$, ",")
FirstLine$ = Left$ (FullAddress$, BreakPoint%)

The example above uses Left$ to divide a string expression at a breakpoint determined by the location of a comma.

#### *Len (string)*

Returns the number of characters in a string expression.

Example:
Len ("Ashenputtel")

The example above uses Len to return 11, which is the number of characters in the string "Ashenputtel".

#### *LoadExtended*

Provides a special editor for the forms that have extended comments.

Example:
{­
 MenuItem "&Extended Ctrl+E"
 MenuCode
 {­
 Launch("ua2")
 LoadExtended
 }
}

The example above uses LoadExtended to open the extended comment editor of the URAR form specified in the string parameter of Launch.

#### *Mid$ (string, start, length)*

Returns a portion of the string expression that is the first parameter. The starting expression is the character position at which the returned string begins. The length expression indicates how long of a string should be returned. If the length is not provided or if the length provided is longer than the difference between the length of the string and the starting expression, the string returned starts at the starting expression and includes the remainder of the string expression.

Example:
BreakPoint1% = InStr (1, FullAddress$, ",")
BreakPoint2% = InStr (BreakPoint1% + 1, FullAddress$, ","
FirstLine$ = Left$ (FullAddress$, BreakPoint1%)
SecondLine$ = Mid$ (FullAddress$, BreakPoint1% + 1, BreakPoint2% - BreakPoint1%)
ThirdLine$ = Right$ (Full Address$, BreakPoint2%)

The example above uses Mid$ to extract the center portion of the string expression FullAddress$.

#### *MsgBox (string)*

Provides a message box with the message indicated by the message string.

Example:
MsgBox "Appraisal has been cancelled."

The example above provides a message indicating that the apprasial has been cancelled.

#### *NumFormat$ (string expression, integer)*

Takes a numeric string and returns a string to the number of decimal places specified as the integer.

Example:
a$ = NumFormat$("500", 3)

The example above returns 500.000 in a$.

#### *Open (filename) For Append As # (integer)/Open (filename) For Output As # (integer)*

Opens a sequential file3NF3ZG and allows data to be written to the file using the Print #(expression, string-expression-list) command. The first parameter is a string-expression containing the name of the file to be opened and written to. The second parameter is an integer-expression specifying a unique numeric identifier to be associated with the file. This number is an alias for the file name, and any other files opened while this file is open must use a different numeric identifier. It is used in all file related commands such as Print # and Close #. A file can be opened either for output or for append. If a file is opened for output, a new file is created and if the specified file already exists, all information previously contained in that file is erased. If a file is opened for append, new text is added to the end of the existing file.

Example:
Open "c:\\tra\\urar.txt" For Output As #1
Print #1, "Check Age", "Check $" + Chr$(13) + Chr$(10)
Print #1, "Date"
Close #1

The example above opens c:\tra\urar.txt, writes information to and closes the file. The file is open for output, so if urar.txt already exists, it will be rewritten with the new text. When the file is opened, it is assigned the integer alias "1", and is referred to by the integer from this point on.

See Close #(integer-expression)51YIF3 and Print # (integer, string-expression-list)136WMZC.

#### *Print # (integer, string-expression-list)*

Writes data to a sequential file3NF3ZG. The integer-expression refers to the open file being written to. The integer must refer to a file that has already been opened using the Open command. The string-expression-list parameter is a comma-separated list of data that is to be written to the file.

Example:
Open "c:\\tra\\urar.txt" For Output As #1
Print #1, "Check Age", "Check $" + Chr$(13) + Chr$(10)
Print #1, "Date"
Close #1

The example above uses print commands to write "Check Age, Check $" to the open file, adds a paragraph mark (Chr$(13)) and a carriage return (Chr$(10)), then writes "Date" to the next line. Note that the c:\tra\urar.txt is open before Print writes information to the file. Note also that urar.txt is opened For Output As, so that if urar.txt already exists, it will be rewritten with the new text.

See Close #(integer-expression)51YIF3 and Open (filename) For Append As # (integer)/
Open (filename) For Output As # (integer)6I0X6N for additional information.

#### *PrintReport% (printscheme, driver-name)*

Prints the active report using the specified print scheme to the specified driver. If no print scheme is specified, the Print Commander dialog box appears.

Example:
PrintReport% ("", "PDFWRITR")

The example above, without a print scheme specified, provides the Print Commander dialog box set to print to the driver, PDFWRITR. This did not work, tested with VAPDF.BAS, and it worked fine. Other macros also use this call: MAILPDF.BAS and PRINTREP.BAS. You can check those if you want to see working examples. This example in the DDE guide is misleading.  Here's some explanation.

  If you call it as a function you need parentheses like this:

    n% = PrintReport%(ps$, drv$)

If PrintReport succeeds the return value 1 will be assigned to variable n%.  Otherwise n% is set to zero. If you don't care about the return value, issue a command like this:

    PrintReport "", "Acrobat PDF"

 [This one works.]

Here the return value is ignored.  When using this subroutine form you can omit the parentheses and type character, '%'.  This syntax is consistent with other versions of BASIC.  (Although FrankBASIC is not always consistent; GO() and LAUNCH() still require parentheses even though they don't return a value.)

#### *Ques (string)*

Provides an input box containing the contents of the string expression and Yes and No checkboxes.

Example:
If Ques( "Are you sure?" ) Then
 4|1 = "yes"
Else
 4|1 = "no"
End If

The example above provides an input box which inquires "Are you sure?" along with checkbox choices for yes and no.

#### *Register (DLL name, function name, parameter list)*

Allows definition and call of an external function in a DLL.

Example:
Register "UTILITY.DLL", "CopyFile", "SS"

The parameter list is defined as follows:

|  |  |
| --- | --- |
| **Parameter** | **Definition** |
| s | C string |
| i | Short integer ~= 2 bytes |
| u | DWORD = 4 bytes |
| r | Reference to address |

The example above registers a "CopyFile" function from the Utility.DLL, allowing for two C string parameters for the source and destination files.

#### *RemoveAttachment (identifier-string)*

Removes the attachment specified in the parameter.

Example:
RemoveAttachment ( )

The example aboves removes the ---------------------- attachment from the active report.

#### *Replace(string)*

Replaces existing data in a field with a string of text. See Insert (string)Y8RDTD to leave the existing data intact and add the macro information to the information that already exists.

Example:
Replace("None")

The example above replaces the existing text in the field with None.

#### *Report$*

Returns the form identifier of the major form in the current report.

Example:
If Report$="ua2" Then
 8|1="ua2"|101|1
End If

The example above determines if the form is the URAR, and if so, transfers the contents of 101|1 of the URAR to the addendum field 8|1. (If you use form name, then the second one is the one you should use. But it did not work when I tried cmps46.)

#### *RestoreCursor*

Restores the cursor in the event that external processes reset it.

Example:
DDETerminate (chan!)
RestoreCursor

The example above uses RestoreCursor as a precautionary measure after a DDE link has been terminated.

#### *Return*

Advances the editor to the next field. Return is the equivalent of striking Enter while editing a field.

Example:
Replace("W. Palm Beach")
Return
Replace("FL")
Return
Replace("33405")

The example above, after inserting "W. Palm Beach" into the current field, moves the editor to the next field and enters .

#### *Right$ (string, integer)*

Returns a string expression of the characters to the farthest right in the string expression. The number of characters returned is determined by the integer expression.

Example:
BreakPoint% = InStr (1, FullAddress$, ",")
FirstLine$ = Left$ (FullAddress$, BreakPoint%)
SecondLine$ = Right$ (Full Address$, Len(FullAddress$) - BreakPoint%)

The example above uses Right$ to divide a string expression at a breakpoint determined by the location of a comma.

#### *RLS$*

RLS (Remove Leading Spaces) deletes the leading spaces in a string.

Example:
a$ = RLS$(" test")

The example above returns "test" to a$ with no leading spaces.

#### *SaveAttachment(identifier-string, destination-directory-string)*

Saves the identified attachment to the destination directory.

Example:
SaveAttachment( , )

The example above saves the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ attachment to the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ directory.

#### *SaveReport (filename)*

Saves the active report to the specified filename. If no filename parameter is specified the Save As dialog box appears.

Example:
SaveReport (X12File$)

The example above saves the active report as X12File$.

Example:
SaveReport ( )

The example above provides the File Save As dialog box.

#### *ScrolltoLine (expression) and Go Command*

Scrolls to the line given in the parameter.

Example:
Launch("ua2")
ScrolltoLine(12)
Go(13|1)

The example above launches the form named as the string parameter, the URAR, then ScrolltoLine moves line 12 to the top of the screen display, and the Go command moves the editor to field 13|1. *The problem with the go command is that it cannot be used to go to another addenda. You have to launch that addenda before you can go to a cell in that addenda.*

#### *SecureReport*

Currently under development. Will protect a report from future modifications.

#### *Selection*

Returns the last action that happened.

Example:
Select Case Selection
  **Case Keypress
 Go(12|4)**End Select

The example above uses Selection for the Keypress case. If a keyboard key is pressed, the editor moves to 12|4.

#### *SendCommandMessage (menu ID)*

Causes the identified command on the TRA menu to execute. The menu identifier must be known.

Example:
SendCommandMessage( 24334) : File Save As

The example above sends a message to execute the TRA command 2334, File Save As. **A list of TRA commands follows.**

The following are commands that you can use in AI via a macro that will perform the same thing as what is under each of the main menus. All you have to do is to create a macro in the cell where you want to run this command and type the following (where # is the number of the command). It doesn't matter where you are at in the form.

SendCommandMessage #

These will be grouped by the main menu item each falls under (File, Edit, Action, Cell, View, Options, Tools, Window, and Help). The Contents menu is a little complicated. It is described at the end.

##### FILE

New 313

Open 311

File Index 317

Save 312

Save As 24334

Save Skel 183

Import 284

Print Setup 102

Print Preview 286

Print 309

Utilities

 - Report Summary 244

 - Restructure Report 210

 - Import Common Responses 246

 - Printer Fonts 257

 - Directories 263

 - Change User 323

Close 228

Exit 24340

##### EDIT

Undo 322

Cut 303

Copy 304

Paste 305

Clear 24324

Insert As Response 192

##### ACTION

Clear Addendum 324

Swap Comps 308

Copy Comps 195

Clear Comps 196

Spell Check 310

Sign/Unsign Report 247

Insert Image 259

##### CELL

Regular 175

Bold 176

Italics 177

Underline 178

Fonts 173

Format 180

Common Responses 166

Quick Responses 302

##### VIEW

Find Editor 300

Draft Mode 162

Zoom 316

Clean Screen 245

##### OPTIONS

Auto-Save 227

Color 152

Fonts 172

Format 157

Mouse 174

Scroll Mode 150

Images 262

Advanced

 - Auto Text Select 182

 - Auto Cell Transfer 181

 - Extended Comments 258

 - Uppercase Transfers 321

 - Uppercase X's 320

##### TOOLS

Global Responses 218

##### WINDOW

Tile Horizontally 191

Tile Vertically 24336

Cascade 24337

Arrange Icons 24335

Close All 24338

##### HELP

Features and Functions 301

Forms and Addenda 250

Using Help 165

Run Demo 225

About 24346

##### CONTENTS

The content menu is dynamic. All of the numbers will be in the 700 range. The first item under Contents is 701, the second is 702, etc. However, any selection on Contents that has subcategories (i.e. Page 1) will have the count pick up for each of it's items. For example, if Page 1 is the second item on the list, then each of it's subcategories will start with 702, 703, etc. The breakdown will be similar to the following for a default skeleton. If the report is restructured and items are removed or the order is changed this may be different.

##### CONTENTS

Extended Comments 701

Page 1

 - Subject 702

 - Neighborhood 703

 - PUD 704

You get the picture. It may take a little figuring to tackle the Additional Addenda section. The default in AI 3.1 start with 757 (Listing Comps).

#### *SetImage% (identifier, filename)*

Places a graphic image (JPG, GIFF, etc.) into a defined image frame. The first parameter identifies the frame, and the filename identifies the image. See our photo page addenda for frame identifiers.

Example:
SetImage% (CMP\_4\_4X6, D:\PHOTOMGR\JPG\NK101202.JPG)

The example above places the file D:\photomgr\jpg\nk101202.jpg in the image frame identified by CMP\_4\_4X6.

#### *SetRequest (global variable identifier)*

Places data in a global variable. Data placed in these global variables remain resident until the application terminates and may be accessed by any other macro session or via DDE.

Example:
SetRequest("Glob\_Ident", string$)

The example above assigns the text in string$ to the global identifier, "Glob\_Ident".

See also GetRequest (global variable identifier)1V4GVDE.

#### *Shell (filename of application)*

**Starts an application if the application is not already running, or switches the focus** to the application if it is running currently. The string expression parameter specifies the file name of the application to be started.

Example:
Shell("c:\\windows\\notepad.exe")

The example above uses Shell to start NotePad. Or since it is in the Windows directory, you could use:

shell("notepad")

But since wordpad is not in C:\windows, you would have to supply the full path:

shell("c:\\progra~1\\access~1\\wordpad.exe")

See WinExecute(filename of application)FVMQC7 for an alternate command.

#### *SplitScreen (filename of application, filename of application*

Divides two running applications into two equal windows, and displays them simultaneously at their full width and one-half their vertical height. The string expression parameters specify the names of the applications to be displayed. Both applications must be currently running when this procedure is called.

Example:
Shell("c:\\windows\\notepad.exe")
SplitScreen("tra.exe", "c:\\windows\\notepad.exe")

The example above has TRA already running. The Shell command starts Notepad, then SplitScreen is used to display the two applications simultaneously, with TRA in the top half of the display, and Notepad in the bottom half.

See Shell (filename of application)1G2J0F..

#### *Status (string)*

Displays a string on the status bar.

Example:
Status("Please wait while adjustments are recalculated."

The example above displays the message contained in the string expression on the status bar.

#### *Str$ (expression)*

Returns a string representation of the value of a numeric expression.

Example:
20|2 = Str$(Val(22|2) - Val(22|3) - Val(21|2))

The example above uses Str$ to provide a string expression for the numeric value calculated when the values of 22|3 and 21|2 are subtracted from 22|2. The complement to this function returns a numeric value of a string:

#### *Sub*

Defines a sub procedure by its name and arguments and then provides the code that constitutes the sub procedure.

Sub CopyLine(Dst%, Src%)
 J%=1
 While J%<=Fieldcount(Dst%)
 If FieldType(Cell(Dst%, J%))<>MultiHandle Then
 Cell(Dst%, J%)=Cell("ua2", Src%, J%)
 End If
 J%=J%+1
 Wend
End Sub

The example above defines a sub procedure, CopyLine, which copies a line from the URAR to the destination location as long as the number of fields in the source line is less than or equal to the number of fields in the destination line.

#### *Sum (cell range)*

Returns the sum of a range of cells.

Example:
Select Case Selection
 Case Calc
 = Sum(62|2:62|8) + Sum(63|2:63|8) + Sum(64|3:64|9)
 121|1 = This
End Select

The example above calculates the sum of the values in cells 62|2, 62|3, 62|4, 62|5, 62|6, 62|7, and 62|8, plus the sum of the values in cells 63|2, 63|3, 63|4, 63|5, 63|6, 63|7, and 63|8, plus 64|3, 64|4, 64|5, 64|6, 64|7, 64|8 and 64|9. This value is provided to 121|1.

#### *SwapComps (integer, integer)*

Swaps the comps data in a market grid at one location to another location, indicated by the integer values.

#### *This*

Represents the current field and may be used as a cell representation.

Example:
This = 21|2
121|1 = This

The example above uses This in the first statement to provide the information from field 21|2 to the current field. The second statement uses This to provide the information from the current field to 121|1.

#### *Ucase$(string)*

Returns a string in which all letters have been converted to uppercase. Only lowercase letters are converted to uppercase; all uppercase letters and non-letter characters remain unchanged.

Example:
A$=Ucase$(This)

The example above uses Ucase$ to assign A$ as the current cell string in uppercase.

#### *Unwarn*

Removes the warning color flag.

Example:
If Selection = Calc Then
 = Val(14|5) + Val(15|8) + Val (16|8) + Val(17|5) + Val(18|8)
 If Val(This)<>100 Then
 Warn
 Else
 Unwarn
 End If
End If

The example above uses Unwarn to remove the warning flag color when fields 14|5, 15|8, 16|8, 17|5, and 18|8 add to 100.

#### *Val ( string )*

Returns the numeric value of a string of characters. The string-expression is a sequence of characters that can be interpreted as a numeric value. The Val function stops reading the string at the first character that it does not recognize as part of a number. Val also strips blanks, tabs, and line feeds from the string.

Example:
Val (" 1615 198th Street N.E.")

The example above returns the value 1615198. The complement to this function returns the string representation of a numeric value.

#### *Wait*

Sets a timed interval to wait before continuing.

Example:
Wait 10

The example above waits 10 seconds before continuing, allowing other applications to process.

#### *WaitForTerminate (filename of application)*

Waits for a specific application to terminate before continuing in the basic script, but still allows for processing of tasks. Useful when the script launches another application but nothing should be done until the the original application terminates.

Example:
WaitForTerminate "x12\_262.dll"

The example above allows the X12\_262.DLL to terminate before proceeding to the next task.

#### *Warn*

Turns the current cell to the warning color. UnWarn turns off the warning attribute in the current cell.

Example:
If Selection = Calc Then
 = Val(14|5) + Val(15|8) + Val (16|8) + Val(17|5) + Val(18|8)
 If Val(This)<>100 Then
 Warn
 Else
 Unwarn
 End If
End If

The example above uses Warn to change the current field to the warning flag color when fields 14|5, 15|8, 16|8, 17|5, and 18|8 do not add to 100.

#### *Warn (cell-type)*

Turns the cell specified in the parameter to the warning color.

Example:
14|6:
...
If (A$="I") or (A$="3") Then
 Warn(15|9)
End If
...

The example above use Warn to change cell 15|9 to the warning color if A$ = I or 3.

#### *While...Wend*

Performs a set of instructions as long as a specified condition is true.

Example:
Sub CopyLine(Dst%, Src%)
 J%=1
 While J%<=Fieldcount(Dst%)
 If FieldType(Cell(Dst%, J%))<>MultiHandle Then
 Cell(Dst%, J%)=Cell("ua2", Src%, J%)
 End If
 J%=J%+1
 Wend
End Sub

The example above uses While to execute the If-Then Statement. While the number of fields in the destination line is less than or equal to those in the source line, the sub procedure will copy the field contents of the line from the source to the destination.

#### *WinExecute(filename of application)*

Execute an application if the application is not already running, or switches the focus to the application if it is running currently. WinExecute returns

Example:
WinExecute("c:\\windows\\notepad.exe")

The example above uses WinExecute to start NotePad and returns.

See Shell (filename of application)1G2J0F. for an alternate command.

**Note the above concerning WinExecute anticipates a return value and thus has ( ). Shell is inconsistent in that it does not give a return value, but it requires ( ). If you want to use the WinExecute without a return value, then use:**

WinExecute"c:\\windows\\notepad.exe"

Reed's notes in the TRA code say that WinExecute is the same as Shell, except that WinExecute returns a value.  Actually the code that runs in each case is quite different, but both rely on the WinExec() system call.

You should be able to write a macro like,

n% = WinExecute%( "notepad" )
MsgBox Str$( n% )

The return value might give you a clue about what is going on.

If the number is less than 32 it is an error.  Otherwise it "identifies the instance of the loaded module".  That's quoted from the 16-bit API help file.  From the same source here is a list of possible errors:

Value Meaning

0 System was out of memory, executable file was corrupt, or relocations were invalid.
2 File was not found.
3 Path was not found.
5 Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.
6 Library required separate data segments for each task.
8 There was insufficient memory to start the application.
10 Windows version was incorrect.
11 Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
12 Application was designed for a different operating system.
13 Application was designed for MS-DOS 4.0.
14 Type of executable file was unknown.
15 Attempt was made to load a real-mode application (developed for an earlier version of Windows).
16 Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.
19 Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
20 Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
21 Application requires Microsoft Windows 32-bit extensions.

When, as in my example above, the supplied string does not contain path information, the function searches for the file in:

1 The current directory.

2 The Windows directory (the directory containing WIN.COM); the GetWindowsDirectory function retrieves the path of this directory.

3 The Windows system directory (the directory containing such system files as GDI.EXE); the GetSystemDirectory function retrieves the path of this directory.

4 The directory containing the executable file for the current task; the GetModuleFileName function retrieves the path of this directory.

5 The directories listed in the PATH environment variable.

6 The directories mapped in a network.

If you do include path information, remember that it must be the short path name.  Eight dot three, no included space characters, etc.

Shell("C:\\PROGRA~1\\WINDOW~1\\ACCESS~1\\NOTEPAD.EXE")

#### *WritePPS% ( section string, entry string, filename )*

Writes the Private Profile String in an INI file and assigns it to the specified entry and section. May be assigned as an integer or not.

Example:
Conversion%=WritePPS%("Directories", "Output", 0, "X12\_262.ini")

Example:
acrodir$ = GetPPS$("Extensions", "PDF", "N/A", "WIN.INI"
Write PPS "Extensions", "PDS", acrodir$, "WIN.INI"

The first example above assigns Conversion% as the private profile string "output" written in the "directories" section of the X12\_262.INI file.

The second example writes acrodir$ as "PDS" in the "Extensions" section of the Win.INI file. See also GetPPS$(section string, entry string, default string, filename)312R\_5B

#### *= (expression)*

Returns the evaluation of the expression to the right of the equal sign.

Example:
12|3:
Select Case Selection
 Case Calc
 = Val(12|1) + Val(12|2)
 122|1 = Val(12|1) + Val(12|2)
End Select

The example above uses = the first time to calculate the values of 12|1 plus 12|2 and provides this value to 12|3 when the case is Calc. The second use of **=** provides the same calculation to field 122|1, specified on the left side of the equal sign.

#### *= (string)*

Returns the string expression to the right of the equal sign.

Example:
12|3:
Select Case Selection
 Case Leave
 = "Done"
 122|1 = "X"
End Select

The example above uses = the first time to provide "Done" to 12|3 when the case is Leave. The second use of = provides an "X" to field 122|1, specified on the left side of the equal sign.

#### *xCell$ (formname$, line%, across%, fontflag%)*

Returns the complete information stored in the field specified by the string for the form name, and the line and across identifiers. If the fontflag% is 0, no font information is embedded.

The xCell$ function is designed to retrieve field text and extended text associated with a field on the form.

Example:
a$ = xCell$ ("ua2", 20, 1, 0)

The example above uses xCell$ to retrieve the field text and extended comment data associated with the neighborhood boundaries information of the URAR.

#### *SkelState%(skeleton%)*

Sets the current form's status as a skeleton or disables skeleton status. Returns the form to its previous status, either a skeleton or not.

Example:
a% = SkelState%(\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

The example above returns the form indicated by the parameter to its previous state, toggling between skeleton status and non-skeleton status.

#### *FormName$( )*

Returns the name of the current form.

Example:
a$ = FormName$ ( )

The example above provides the name of the current form to a$.

Here is one John helped me work up for Phil:

; For all forms that use standard cmps46 and cmps79 addenda

adj! = VAL( InputBox$( "Enter +- % Adjustment", "+- % Selling Price",""))

if Formname$()="ua2" then

 ; routine for comps 1-3

 if activefield(line|2) then

 result!=adj! \* val(104|2) \* 0.01

 elseif activefield(line|4) then

 result!=adj! \* val(104|3) \* 0.01

 else

 result!=adj! \* val(104|4) \* 0.01

 end if

elseif Formname$()="cmps46" Then ; or Formname$()="cmps79"

 ; routine for comps 4-6

 if activefield(line|2) then

 result!=adj! \* val(13|2) \* 0.01

 elseif activefield(line|4) then

 result!=adj! \* val(13|3) \* 0.01

 else

 result!=adj! \* val(13|4) \* 0.01

 end if

end if

replace(str$(result!))

return

I ad to change first condition from UA2 since UA2 was always true not allowing the second condition to activate. Formname() for forms. No need for val for numbers.

#### *SetFileName (new-name-string)*

Assigns a new file name and path to the active report, with no return value.

Example:
SetFileName (X12File$)

The example above assigns the name X12File$ to the active report.

# Glossary of Terms

## client

An application program that requests and receives information from the serverWXJ36R.

## client/server

An application program that both requests and provides information.

## conversation

The relationship between the application programs that are sharing information.

## dynamic data exchange (DDE)

The medium through which information is shared between application programs.

## functions

ProceduresE786T7 that result in the return of a value.

## procedures

A portion of a program that performs a particular task.

## sequential file

A text file with sequential input and output.

## server

An application program that provides information to the client8.INLA.

## sub procedures

ProceduresE786T7 that do not result in a value being returned.

# Notes

Opening the Report profile does not cause the BAS file in UA2 to transfer the information to the Report Profile. However, closing the Report Profile or switching back to the UA2 after opening the Report Profile causes the information to transfer. I cannot close the Report Profile after I open it with a send command because it will crash with a GPF in TRA.exe, even if I use this command on the main page of the UA2 with the Report Profile closed. I cannot use the Go command to go back to the main form since a go command is limited to the addenda you are on. I cannot use a launch command because it opens a second session of the main page.

This will open the Report Profile and then switch back to the UA2

launch("offscrn")

launch("ua2")

This will do it for any report type:

launch("offscrn")

launch(Report$)

Do not put quotation marks around a string variable. Quotations make it a string so it can use these to assign information to a string variable. Thus, do not put Report$ in quotes!!!

# Additional SendCommandMessages

// =================================================== MAIN MESSAGE ID'S

// If you change any of these ID's, it could effect custom settings for

// Aames Home Loan. We did some custom disabling of menu items for them

// based on menu item ID's. See the Aames Home Loan custom install TRA.INI

// [Disable...] settings. -RR 2/26/96

// 700 - 900 reserved for Contents menu

// 990 - 995 reserved for file history

#define IDM\_FILE\_HISTORY 990

#define IDM\_SCROLLMODE 150

#define IDM\_COLOR 152

#define IDM\_PERMISSION 153

#define IDM\_FIELDGROUP 154

#define IDM\_JUMP 155

#define IDM\_EDITFORM 156

#define IDM\_**FORMAT** 157

#define IDM\_EXTENDED 158

#define IDM\_**FORMATTOGGLE** 159

#define IDM\_STATUSBAR 160

#define IDM\_TOOLBAR 161

#define IDM\_DRAFTMODE 162

#define IDM\_HELP 163

#define IDM\_HELP\_KEYBOARD 164

#define IDM\_HELP\_HELP 165

#define IDM\_COMMONRESPONSES 166

#define IDM\_CUSTSCREENUSE 167

#define IDM\_CUSTSCREENEND 168

#define IDM\_CUSTSCREENOPTIONS 169

#define IDM\_REVIEWER\_REPORT 170

#define IDM\_ADVANCED 171

#define IDM\_FONTS 172

#define IDM\_CELLFONTS 173

#define IDM\_MOUSERCLICK 174

#define IDM\_REGULAR 175

#define IDM\_BOLD 176

#define IDM\_ITALIC 177

#define IDM\_UNDERLINE 178

#define IDM\_GOSPELL 179 // not menu item, msg control

#define IDM\_CELLFORMAT 180

#define IDM\_**AUTOTRANSFER** 181

#define IDM\_AUTOTEXTSELECTION 182

#define IDM\_SAVESKEL 183

#define IDM\_USERINFO 184

#define IDM\_WRAPOPTS 185

#define IDM\_EXTREGIONS 186

#define IDM\_FORMBITOPTIONS 187

#define IDM\_FORMBITS 188

#define IDM\_INDXDATA 189

#define IDM\_FIELDLOCATE 190

#define IDM\_MDITILEHORIZ 191

#define IDM\_INSERTRESPONSE 192

#define IDM\_EDITCOMMENT 193

#define IDM\_EDITTITLE 194

#define IDM\_COPYCOMPS 195

#define IDM\_CLEARCOMPS 196

#define IDM\_PRINTBASICCODE 197

#define IDM\_READBASICCODE 198

#define IDM\_SHOWGROUP 199

#define IDM\_PRINTQFORMCODE 200

#define IDM\_READQFORMCODE 201

#define IDM\_MAINSTATUS 202 // MAINSTATUS, COM\_RESPONSE1-6 are to be used as

#define IDM\_COM\_RESPONSE1 203 // message response ID's for buttons on the

#define IDM\_COM\_RESPONSE2 204 // status bars

#define IDM\_COM\_RESPONSE3 205

#define IDM\_COM\_RESPONSE4 206

#define IDM\_COM\_RESPONSE5 207

#define IDM\_COM\_RESPONSE6 208

#define IDM\_LOADMENU 209

#define IDM\_REPORTBUILDER 210

#define IDM\_ACTIVATECOMMENT 211

#define IDM\_ACTIVATEFRAME 212

#define IDM\_ACTIVATECHILD 213

#define IDM\_RESPONSEBAR 214

#define IDM\_MYNCACTIVATE 215

#define IDM\_LENDERDB 216

#define IDM\_MAJORFORM 217

#define IDM\_GLOBALRESPONSES 218

#define IDM\_NEIGHBORHOODDB 219

#define IDM\_DUMPHEADER 220

#define IDM\_READHEADER 221

#define IDM\_CHANGEUSER 222

#define IDM\_FORMRESPONSES 223

#define IDM\_COMPILECODE 224

#define IDM\_LAUNCHDEMO 225

#define IDM\_ERASEBASICCODE 226

#define IDM\_AUTOSAVE 227

#define IDM\_CLOSECHILD 228

#define IDM\_FORMATMODE 229

#define IDM\_FIRSTFIELD 230

#define IDM\_LASTFIELD 231

#define IDM\_REFORMAT 232

#define IDM\_SAVERESPONSEF3 233

#define IDM\_SAVERESPONSEF4 234

#define IDM\_SAVERESPONSEF5 235

#define IDM\_SAVERESPONSEF6 236

#define IDM\_SAVERESPONSEF7 237

#define IDM\_SAVERESPONSEF8 238

#define IDM\_COMMONRESPONSE 239

#define IDM\_BASIC 240

#define IDM\_TOGGLEDEMORESP 241

#define IDM\_STOPDEMO 242

#define IDM\_CELLMENU 243

#define IDM\_SUMMARY 244

#define IDM\_CLEANSCREEN 245

#define IDM\_IMPRESPONSE 246

#define IDM\_SIGNREPORT 247

#define IDM\_REFORMATREPORT 248

#define IDM\_COMPILEDFORM 249

#define IDM\_FORM\_HELP 250

#define IDM\_FORM\_REPORT 251

#define IDM\_HEADER\_REPORT 252

#define IDM\_DUMP\_FORM\_TEXT 253

#define IDM\_EDITOR\_OPTIONS 254

#define IDM\_CARRYDATA 256

#define IDM\_FONTFILE 257

#define IDM\_COMMENTOPTS 258

#define IDM\_INSERTPICTURE 259

#define IDM\_SELECTOBJECT 260

#define IDM\_DELETEPICTURE 261

#define IDM\_MODIFYPICTURE 262

#define IDM\_DIRECTORIES 263

#define IDM\_NEXTFIELD 264

#define IDM\_PREVIOUSFIELD 265

#define IDM\_BUILDCUSTOMMENU 266

#define IDM\_DESTROYCUSTOMMENU 267

#define IDM\_CLEANBUILD 268

#define IDM\_CLEARACTIVECOMP 269

#define IDM\_GO\_COMP\_S 270

#define IDM\_GO\_COMP\_1 271

#define IDM\_GO\_COMP\_2 272

#define IDM\_GO\_COMP\_3 273

#define IDM\_GO\_COMP\_4 274

#define IDM\_GO\_COMP\_5 275

#define IDM\_GO\_COMP\_6 276

#define IDM\_GO\_COMP\_7 277

#define IDM\_GO\_COMP\_8 278

#define IDM\_GO\_COMP\_9 279

#define IDM\_GO\_COMP\_10 280

#define IDM\_GO\_COMP\_11 281

#define IDM\_GO\_COMP\_12 282

#define IDM\_REFORMATFRAME 283

#define IDM\_IMPORT\_FORM 284

#define IDM\_SIZEBACK\_COMMENTS 285

#define IDM\_PRINT\_PREVIEW 286

#define IDM\_PAGE\_BREAK 287

#define IDM\_ENABLE\_TRANSFERS 288

#define IDM\_REFRESH\_TOOLBAR 289

#define IDM\_FRENCH\_FORM 290

// fill in stuff here...

#define IDM\_ACTIVEFIELD 300

#define IDM\_HELP\_INDEX 301

#define IDM\_QUICKRESPONSES 302

#define IDM\_EDITCUT 303

#define IDM\_EDITCOPY 304

#define IDM\_EDITPASTE 305

 // ID# 306,307, & 308 used for COMPS in UAR.h

#define IDM\_PRINTFORM 309

#define IDM\_SPELLCHK 310

#define IDM\_MDIFILEOPEN 311

#define IDM\_FILESAVE 312

#define IDM\_MDIFILENEW 313

#define IDM\_DESIGNER 314

#define IDM\_JUMPTOLINE 315

#define IDM\_CHOOSE 316

#define IDM\_FILEINDEX 317

// extended comment menu

#define IDM\_NEXTCOMMENT 318

#define IDM\_PREVIOUSCOMMENT 319

#define IDM\_UPPERCASEX 320

#define IDM\_UPPERCASETRANSFERS 321

#define IDM\_EDITUNDO 322

#define IDM\_CHILD\_CHANGEUSER 323

#define IDM\_CLEAR\_ADDENDUM 324

#define IDM\_ADJUSTMENTS 325

#define IDM\_DELETE\_WORD 326

#define IDM\_EDITMENU 327

#define IDM\_MOVETOAI 328

#define IDM\_ANALYIZECOMPLETE 329

#define IDM\_REPLACEFONT 330

#define IDM\_JUSTIFY\_TEXT 331

#define IDM\_SHIFT\_FORM 332

# Miscellaneous Tibits

## ctrl +m then print to put the cell ID in each cell

Note if you do this on the main page and the report profile, you can then open any addenda and know where the information in that addenda is being transferred from.

## ctrl +m then formname to put the formname

The report profile is *offscrn*

for example:

cell("offscrn",26,1)=cell("UA2",9,1)

cell("offscrn",3,3)=cell("EPI",3,1)

## Conditional Statements

The following is a three line conditional statement that I made to transfer Project and Map information from the UAR to the extra cell on the Report Profile:

 if cell ( "offscrn", "Extra" ) = "" then

 cell ( "offscrn", "Extra" ) = "Map: " + cell ("ua2", "MapRef") + "; Project: " + cell "ua2", "Project")

 end if

The first line says that if the cell is blank (i.e., = ""; with a blank between the ""), then to perform the operation. Otherwise, it will not perform it. The second line gives to operation to be performed. The third line is the required statement telling it to end the **if** (i.e, end the conditional statement).

## Long text

Below is a macro that John wrote to insert the text into the 2002 Uspap:

CRLF$ = Chr$(13)+Chr$(10)

Uspap2kFile$ = "C:\\TRA\\uspap2k.txt"

s$ = ""

i% = 1

ReadFile Uspap2kFile$, 33

 f$ = GetLine$(33)

 While i% < 75 And f$ <> "~EOF"

 s$ = s$ + f$ + CRLF$

 f$ = GetLine$(33)

 i% = i% + 1

 Wend

 10|1 = s$

 s$ = ""

 While i% < 80 And f$ <> "~EOF"

 s$ = s$ + f$ + CRLF$

 f$ = GetLine$(33)

 i% = i% + 1

 Wend

 94|1 = s$

Close #33

This will read (input) text from the following file c:\tra\uspap.txt into the two pages of the Uspap. I just created this text file of course. I had to use returns to have it come out correctly on the second page. Further, I had to put another line (a blank return) at the bottom of the text of the second page and change the loop from 149 to 80 in the macro so that the loop ends after the last line of text but before hitting the end of the input. Otherwise, a bug in GetLine() returns garbage (ÿ) when it hits the end-of-file, even from a valid file.

Below I have annotated certain parts of this macro

**CRLF$ = Chr$(13)+**Chr$(10) **; retain original text format for line breaks**

**Uspap2k**File$ = "C:\\TRA\\**uspap2k**.txt" ; or select another name and path for your text file

s$ = "" ; sets this variable to a blank

i% = 1

**ReadFile** Uspap2kFile$, 33 ; 33 is an arbitrary file number you chose. ReadFile is a DOS command.

 f$ = GetLine$(33)

 While i% < 75 And f$ <> "~EOF" ; 75 lines and then end of file

 s$ = s$ + f$ + CRLF$

 f$ = GetLine$(33)

 i% = i% + 1

 Wend

 10|1 = s$ ; this is the cell ID for the first page

 s$ = "" ; sets this variable to a blank. This begins the second loop

 While i% < 80 And f$ <> "~EOF" ; changed this from 149 to 80 to avoid the ÿ

 s$ = s$ + f$ + CRLF$

 f$ = GetLine$(33)

 i% = i% + 1

 Wend

 94|1 = s$ ; this is the cell ID for the second page

Close #33

Text file I created had this with a blank line at the bottom:

APPRAISER'S CERTIFICATION:

The following Certification statements are in addition to and may supercede the signed Appraiser's Certification attached to this

appraisal report. This Appraiser's Certification is compliant with the current edition of the Uniform Standards of Professional

Appraisal Practice.

I certify that, to the best of my knowledge and belief:

The statements of fact contained in this report are true and correct.

The reported analyses, opinions, and conclusions are limited only by the reported assumptions and limiting conditions, and are my

personal, impartial, and unbiased professional analyses, opinions, and conclusions.

I have no present or prospective interest in the property that is the subject of this report, and no personal interest with respect to

the parties involved.

I have no bias with respect to the property that is the subject of this report or to the parties involved with this assignment.

My engagement in this assignment was not contingent upon developing or reporting predetermined results.

My compensation for completing this assignment is not contingent upon the development or reporting of a predetermined value or

direction in value that favors the cause of the client, the amount of the value opinion, the attainment of a stipulated result, or the

occurrence of a subsequent event directly related to the intended use of this appraisal.

My analyses, opinions, and conclusions were developed, and this report has been prepared, in conformity with Uniform

Standards of Professional Appraisal Practice.

I (We) have made a personal inspection of the property that is the subject of this report.

No one provided significant real property appraisal assistance to the person(s) signing this report.

PURPOSE, INTENDED USE, AND INTENDED USER OF THE APPRAISAL:

The purpose of the appraisal is to estimate the market value of the subject property, as defined in this report, as of the effective date

of this report. The intended use of the appraisal is to assist the client and any other intended users in the underwriting, approval,

and funding of the mortgage loan. The intended users of this report are the stated client and any other institutions involved in the

underwriting, approval, and funding of the mortgage loan. No one else, including the purchaser and seller, should rely on the

estimate of value or any other conclusions contained in this appraisal report.

ANALYSIS AND REPORT FORM:

The appraisal is based on the information gathered by the appraiser from public records, other identified sources, inspection of the

subject property and neighborhood, and selection of comparable sales, listings, and/or rentals within the subject market area.

The original source of the comparable data described in the Data Source section of the market grid along with the source of

confirmation is provided, where available. The original source is presented first. The sources and data are considered reliable.

When conflicting information was provided, the source deemed most reliable has been used. Data believed to be unreliable was not

included in the report or used as a basis for the value conclusion. The extent of the analysis applied to this assignment is stated in

the Appraiser's Certification included above and attached to this report.

DEFINITION OF INSPECTION

The term "Inspection", as used in this report, is not the same level of inspection that is required for a "Professional Home

Inspection". The appraiser does not fully inspect the electrical system, plumbing system, mechanical systems, foundation system,

floor structure, or subfloor. The appraiser is not an expert in construction materials and the purpose of the appraisal is to make an

economic evaluation of the subject property. If the client needs a more detailed inspection of the property, a home inspection, by a

Professional Home Inspector, is suggested.

DIGITAL SIGNATURES

The signature(s) affixed to this report, and certification, were applied by the original appraiser(s) or supervisory appraiser and

represent their acknowledgements of the facts, opinions and conclusions found in the report. Each appraiser(s) applied his or her

signature electronically using a password encrypted method. Hence these signatures have more safeguards and carry the same

validity as the individual's hand applied signature. If the report has a hand-applied signature, this comment does not apply.

OPINION OF MARKET VALUE VS ESTIMATE OF MARKET VALUE

The current Uniform Standards of Professional Practice defines the market value conclusion as an opinion of market value and not

an estimate of market value.