

PROGRAMMER'S GUIDE



Monarch<sup>TM</sup>

## **Copyright Notice**

Monarch program copyright © 1999-2005 by Math Strategies.

Monarch Programmer's Guide copyright © 1999-2005 by Datawatch Corporation. All rights reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior written consent from Datawatch Corporation.

## **Acknowledgments**

Monarch was developed by Math Strategies of Greensboro, North Carolina using Microsoft C++, the Microsoft Foundation Class Libraries, the Microsoft Jet engine, the Objective Grid spreadsheet control, the ChartFX charting DLL and the DynaZip file archiving DLL.

Printed in the United States of America  
Ninth Edition  
March 2005 (On Website only)

# Preface

This document describes the set of Monarch properties and methods that have been exposed for use by application developers. These properties and methods can be called from within any application that provides support for COM/OLE/ActiveX. This includes standard visual development tools such as Microsoft Visual Basic, Microsoft Visual C++, Borland C/C++, and Borland Delphi as well as most conventional programming languages and scripting languages such as VBScript.

This manual is intended for programmers who have already learned the fundamentals of their chosen programming language. It is not intended as a learning guide. If you have questions regarding the execution of the properties or method calls or the use of COM within your programming or scripting environment, please refer to the Programmer's Guide included with your language reference materials.

# Table of Contents

|  |          |
|--|----------|
| <b>CHAPTER 1 .....</b>   | <b>1</b> |
| INTRODUCTION.....  | 1        |
| <i>What is COM?</i> .....  | 1        |
| Advantages of using COM.....   | 1        |
| <b>CHAPTER 2 .....</b>   | <b>3</b> |
| MONARCH PROPERTIES AND METHODS .....   | 3        |
| <i>Properties</i> .....  | 4        |
| CurrentFilter.....   | 4        |
| CurrentModel.....  | 4        |
| CurrentPRFModel.....   | 5        |
| CurrentSort.....   | 5        |
| CurrentSummary.....  | 5        |
| FilterCount.....   | 6        |
| SortCount.....   | 6        |
| SummaryCount.....  | 6        |
| <i>Methods</i> .....   | 7        |
| CloseAllDocuments().....   | 7        |
| DisplayWindow(integerSize).....  | 7        |
| Exit().....  | 7        |
| ExportSummary(stringExportFile).....   | 7        |
| ExportTable(stringExportFile).....   | 8        |
| FindText(booleanOccurrenceFlag, stringSearchText, booleanMatchCaseFlag, booleanAllDocsFlag).....                     | 8        |
| GetFilterNameAt(integerIndex).....   | 9        |
| GetSortNameAt(integerIndex).....   | 9        |
| GetSummaryNameAt(integerIndex).....  | 10       |
| IsActive().....  | 10       |
| JetExportSummary(stringExportFile, stringTableName, integerAppendFlag).....  | 10       |
| JetExportTable(stringExportFile, stringTableName, integerAppendFlag).....  | 11       |
| OpenDatabase(stringConnectString, stringPassword, stringTable View, stringModel).....                                | 11       |
| PRFPublish(stringPublishFileName, booleanIncludeTree, booleanIncludeModel, integerIncludeTable, booleanEncrypt)..... | 12       |
| PrintReport(booleanDirectPrintFlag).....   | 12       |
| PrintSummary(booleanDirectPrintFlag).....  | 12       |
| PrintTable(booleanDirectPrintFlag).....  | 13       |
| RunAllExports().....   | 13       |

|   |           |
|---|-----------|
| RunExport(stringProjectExportName).....                               | 13        |
| SetDataSourcePassword(stringLookupName, stringPassword) .....         | 13        |
| SetFieldVisible(stringFieldName, booleanVisibleFlag) .....            | 14        |
| SetFirstView(stringWindowType) .....                                  | 14        |
| SetInputCharacterSet(stringCharacterSet).....                         | 14        |
| SetJoinPassword(stringPassword) .....                                 | 15        |
| SetLogFile(stringLogFile, booleanAppendFlag) .....                    | 15        |
| SetModelFile(stringModelFile).....                                    | 15        |
| SetOutputCharacterSet(stringCharacterSet) .....                       | 16        |
| SetPRFFile(stringFileName, stringPassword) .....                      | 16        |
| SetProjectFile(stringProjectFile) .....                               | 16        |
| SetReportFile(stringReportFile, booleanAddFlag) .....                 | 17        |
| SetRuntimeParameter(stringFieldName, stringFieldValue) .....          | 17        |
| SetTextAppend(booleanAppendFlag).....                                 | 18        |
| SetView(stringWindowType) .....                                       | 18        |
| Version() .....   | 18        |
| WriteToLogFile(stringUserLogInfo).....                                | 18        |
| <b>CHAPTER 3 .....</b>  | <b>19</b> |
| PROGRAMMING MONARCH METHODS AND PROPERTIES .....                      | 19        |
| <i>Monarch COM Registration</i> .....                                 | 19        |
| <i>Calling the Monarch COM Server from a Client Application</i> ..... | 19        |
| Creating the Appropriate Client Object .....                          | 20        |
| Program Control and Termination .....                                 | 20        |
| Program Subroutine Example .....                                      | 21        |
| <b>CHAPTER 4 .....</b>  | <b>23</b> |
| LAUNCHING A MONARCH SESSION FROM AN APPLICATION .....                 | 23        |
| <i>Launching an Interactive Monarch Session</i> .....                 | 23        |
| Program Subroutine Example .....                                      | 23        |
| <b>Appendix A .....</b>   | <b>25</b> |
| ERROR MESSAGES.....   | 25        |
| OLE Automation Server cannot create object.....                       | 25        |
| <b>Appendix B .....</b>   | <b>26</b> |
| PROPERTIES AND METHODS BY VERSION .....                               | 26        |
| <b>Appendix C .....</b>   | <b>28</b> |
| TECHNICAL SUPPORT .....   | 28        |



## CHAPTER 1

# Introduction

A number of Monarch customers have asked for a way to integrate Monarch functionality into their own Windows applications. In response to their requests, we have created a set of Monarch properties and methods that can be called via COM from languages like C/C++ or Visual Basic. These properties and methods provide programmers with all of the commands necessary to launch a Monarch session or to incorporate Monarch's data extraction and export functionality into an application.

## What is COM?

COM is an industry standard that applications can use to expose objects to other programs. With COM, applications can allow their objects to be manipulated remotely, via program control. The application that is providing the object creates and manages it. The controlling application manipulates the object by setting properties and performing methods (i.e. actions) through program method calls.

### Advantages of using COM

Even without COM, any Windows application can be run [launched] from another application by employing the WinExec function. However, a program launched in this manner will run independently of the application which launched it. You cannot be certain that the launched application will finish executing before the next statement in your main application is processed. Furthermore, there is no link between the programs. If your main program terminates, the program that it launched may continue to run. There is no way to say, "if the main program shuts down, turn off any other programs it launched".

By using COM, these problems can be eliminated. Monarch is established as a COM server and the main program becomes the client, using method calls to communicate with the server. Execution of a statement in the calling program will not occur until the previous method has been completed. When the calling program terminates, the server is shut down, closing down the Monarch application.

Another advantage of using COM becomes evident when you have an application that requires automation of multiple passes through Monarch. Using the WinExec method, the entire Monarch application, report file and model must be loaded for each pass. With COM, the Monarch server is loaded only once. You can apply a new sort or filter, load a new model, or load a new report through method calls, resulting in significantly faster execution.

## CHAPTER 2

# Monarch Properties and Methods

This chapter describes each of the Monarch methods. It is important to note that only a subset of the functionality of Monarch can be accessed via COM. Monarch's methods provide commands necessary to launch a Monarch session from another application and to incorporate Monarch's data extraction capability into an application. This subset provides an application developer with the ability to perform the following tasks:

1. Launch Monarch as a COM Server process. This capability allows an application developer to embed Monarch's data extraction and export capabilities into his or her application, behind the scenes. The following tasks apply to the use of Monarch in this role:
  - a. Open a report or a series of reports. Monarch can extract data from any number of reports at-a-time. For information about extracting data from a series of reports, see the section entitled *Opening Multiple Instances of a Report* in the *Monarch On-line Reference Guide*.
  - b. Open a model file.
  - c. Query the model file to determine what filter, sort and summary definitions are available.
  - d. Select a filter definition from the model file to apply to the extracted data set.
  - e. Select a sort definition from the model file to apply to the extracted data set.
  - f. Select a summary definition from the model file to generate a summary report from the extracted data set.
  - g. Export the extracted data set or the summary to any of Monarch's supported file formats (see the *Monarch On-line Reference Guide* for a complete description of each supported export format).

2. Launch an interactive Monarch session. This capability can be used to automatically open a report and model file at the beginning of a Monarch session. This capability is useful if you intend to use Monarch as a report viewing and analysis tool to augment a document management system or report archive system.

**NOTE:** Methods and properties available will vary depending on the version of Monarch that is installed. Refer to Appendix B for information regarding which methods and properties are available in each version.

## Properties

### **CurrentFilter**

**CurrentFilter** is a variable that is used to set or query the name of the currently active filter definition.

**CurrentFilter** accepts a string of up to 31 characters representing the name of the currently active filter definition.

When a model file is opened (via the **SetModelFile** method), **CurrentFilter** is set to the name of the active filter definition established in the model. When a model is opened as part of a project file (via the **SetProjectFile** method), **CurrentFilter** is set to the name of the active filter definition referenced in the project file.

If no filter definition is referenced in the project file, the active filter definition from the model file is used. If no active filter definition is established in either the project or the model, the default value of **CurrentFilter** is blank (an empty string).

This variable may also be used to establish an active filter definition or change the currently active filter definition. To do this, simply assign it the name of one of the filter definitions stored in the model. If the new name assigned does not match any of the values stored in the model, the value of **CurrentFilter** remains unchanged.

### **CurrentModel**

**CurrentModel** is a variable that is used to set or query the name of a currently active model.

**CurrentModel** accepts a string of up to 256 characters representing the name of the currently active model.

### **CurrentPRFModel**

**CurrentPRFModel** is a variable that is used to set or query the name of the model in the currently active Portable report File (PRF).

**CurrentPRFModel** accepts a string of up to 256 characters representing the name of the model within a currently active portable report file.

### **CurrentSort**

**CurrentSort** is a variable that is used to set or query the name of the currently active sort definition.

**CurrentSort** accepts a string of up to 31 characters representing the name of the currently active sort definition.

When a model file is opened (via the **SetModelFile** method), **CurrentSort** is set to the name of the active sort definition established in the model. When a model is opened as part of a project file (via the **SetProjectFile** method), **CurrentSort** is set to the name of the active sort definition referenced in the project file.

If no sort definition is referenced in the project file, the active sort definition from the model file is used. If no active sort definition is established in either the project or the model, the default value of **CurrentSort** is blank (an empty string).

This variable may also be used to establish an active sort definition or change the currently active sort definition. To do this, simply assign it the name of one of the sort definitions stored in the model. If the new name assigned does not match any of the values stored in the model, the value of **CurrentSort** remains unchanged.

### **CurrentSummary**

**CurrentSummary** is a variable that is used to set or query the name of the currently active summary definition.

**CurrentSummary** accepts a string of up to 31 characters representing the name of the currently active summary definition.

When a model file is opened (via the **SetModelFile** method), **CurrentSummary** is set to the name of the active summary definition established in the model. When a model is opened as part of a project file (via the **SetProjectFile** method), **CurrentSummary** is set to the name of the active summary definition referenced in the project file.

If no summary definition is referenced in the project file, the active summary definition from the model file is used. If no active summary definition is established in either the project or the model, the default value of **CurrentSummary** is blank (an empty string).

This variable may also be used to establish an active summary definition or change the currently active summary definition. To do this, simply assign it the name of one of the summary definitions stored in the model. If the new name assigned does not match any of the values stored in the model, the value of **CurrentSummary** remains unchanged.

### **FilterCount**

**FilterCount** returns the total number of filter definitions available in the currently open model file.

If the model contains no filter definitions or if no model file is currently open, the method returns 0.

### **SortCount**

**SortCount** returns the total number of sort definitions available in the currently open model file.

If the model contains no sort definitions or if no model file is currently open, the method returns 0.

### **SummaryCount**

**SummaryCount** returns the total number of summary definitions available in the currently open model file.

If the model contains no summary definitions or if no model file is currently open, the method returns 0.

## Methods

### **CloseAllDocuments()**

**CloseAllDocuments** closes all open report files and model files.

### **DisplayWindow(integerSize)**

**DisplayWindow** gives the current Monarch window focus and optionally changes the size of the window.

If *size* is 0, the window is maximized. If *size* is 1, the window is simply restored. If *size* is 2, the window is minimized.

### **Exit()**

**Exit** closes all open report files, and the associated model file. It also closes the log file and terminates the Monarch session (removing the Monarch Server from memory).

### **ExportSummary(stringExportFile)**

**ExportSummary** performs an export from the Summary window using the Monarch V3 engine.

In order to use newer file formats supported by later versions of Monarch, use **JetExportSummary** (Professional version only, V6 or greater).

**ExportSummary** causes the data to be exported from the Summary window and written to the file *export file*. If the full file name (including drive and path) is not supplied, the default Export Files location stored in the Monarch defaults will be used

The export file type is limited to the Monarch V3 capabilities and determined by the file extension specified for *export file* (for a list of supported export file formats and their corresponding file extensions, see the *Monarch On-line Reference Guide*).

If the file extension is not provided, the default Export File extension in the Windows Registry will be used. If the file cannot be written, an error message will be written out to the designated log file (specified via the **SetLogFile** method).

**Return Value:** Boolean TRUE if the export completed successfully, otherwise returns FALSE.

### **ExportTable(stringExportFile)**

**ExportTable** performs an export from the Table window using the Monarch V3 engine.

In order to use newer file formats supported by later versions of Monarch, use **JetExportTable** (Professional version only, V6 or greater).

**ExportTable** causes the data to be exported from the Table window and written to the file *export file*. If the full file name (including drive and path) is not supplied, the default Export Files location stored in the Monarch defaults will be used.

If the file extension is not provided, the default Export File extension in the Windows Registry will be used. Only those records matching the **CurrentFilter** will be exported to the file. Records are exported in the order specified by **CurrentSort**.

The export file type is limited to the Monarch V3 capabilities and determined by the file extension specified for *export file* (for a list of supported export file formats and their corresponding file extensions, see the *Monarch On-line Reference Guide*).

If the file cannot be written, an error message will be written out to the designated log file (specified via the **SetLogFile** method).

**Return Value:** Boolean TRUE if the export completed successfully, otherwise returns FALSE.

### **FindText(booleanOccurrenceFlag,stringSearchText,booleanMatchCaseFlag,booleanAllDocsFlag)**

**FindText** finds an occurrence (first or next) of the *SearchText* string.

*OccurrenceFlag* is a Boolean value indicating a find first search or a find next search. If *OccurrenceFlag* is true, the search begins at the top of the document. If *OccurrenceFlag* is false, the search continues from the last found occurrence of the search string.

*SearchText* may be any text string up to 128 characters in length. Wildcard characters are not supported.

*MatchCaseFlag* is a Boolean value indicating the case sensitivity of the search. If *MatchCaseFlag* is true, a case sensitive search is performed. If *MatchCaseFlag* is false, a case insensitive search is performed. *AllDocsFlag* is a Boolean value indicating the scope of the search. If *AllDocsFlag* is true, all documents are searched. If *AllDocsFlag* is false, only the currently active document is searched.

**Return Value:** Boolean TRUE if the string was found, otherwise returns FALSE.

### **GetFilterNameAt(integerIndex)**

**GetFilterNameAt** returns the name of the *n*th filter from within the currently open model.

The parameter *n* must be a value between 0 and FilterCount-1 (FilterCount is the value returned by the **FilterCount** method).

**Return Value:** If the value of *n* is not within the legal range, or if no model is open or if there are no filter definitions established for the currently open model, the method returns an empty string. Otherwise, the method will return a string (up to 31 characters) representing the name of the specified filter.

### **GetSortNameAt(integerIndex)**

**GetSortNameAt** returns the name of the *n*th sort definition in the currently open model.

The parameter *n* must be a value between 0 and SortCount-1 (SortCount is the value returned by the **SortCount** method).

**Return Value:** If the value of *n* is not within the legal range, or if no model is open or if there are no sort definitions established for the currently open model, the method returns an empty string. Otherwise, the method returns a string (up to 31 characters) representing the name of the specified sort definition.

**GetSummaryNameAt(integerIndex)**

**GetSummaryNameAt** returns the name of the  $n$ th summary definition in the currently open model.

The parameter  $n$  should be a value between 0 and SummaryCount-1 (SummaryCount is the value returned by the **SummaryCount** method).

**Return Value:** If the value of  $n$  is not within the legal range, or if no model is open or if there are no summary definitions established for the currently open model, the method returns an empty string. Otherwise, the method returns a string (up to 31 characters) representing the name of the specified summary definition.

**IsActive()**

**IsActive** queries the COM server to determine whether the Monarch COM server is active.

If the Monarch server is active, this method returns a positive integer. If the server is not active, an error condition occurs, which must be trapped. Refer to the programming example in Chapter 3 for sample code illustrating the use of this method.

**JetExportSummary(stringExportFile,stringTableName, integerAppendFlag)**

**JetExportSummary** causes the data to be exported from the Summary window and written to the file *ExportFile*, to the table specified by *TableName*.

Valid *AppendFlag* values are 0 for overwrite, 1 for new table or sheet and 2 for append to existing table or sheet. For some formats, it is not possible to specify a table or sheet name and the *TableName* will be ignored. Additionally for these cases, multi table or sheet options are not allowed.

The export file version is defined by the settings in the Monarch Options under Folders & File Types and will apply to the extension specified as part of *ExportFile*.

**Return Value:** Boolean TRUE if the export completed successfully, otherwise returns FALSE.

**JetExportTable(stringExportFile,stringTableName,  
integerAppendFlag)**

**JetExportTable** causes the data to be exported from the Table window and written to the file *ExportFile*, to the table specified by *TableName*.

Valid *AppendFlag* values are 0 for overwrite, 1 for new table or sheet and 2 for append to existing table or sheet. For some formats, it is not possible to specify a table or sheet name and the *TableName* will be ignored. Additionally for these cases, multi table or sheet options are not allowed.

The export file version is defined by the settings in the Monarch Options under Folders & File Types and will apply to the extension specified as part of *ExportFile*.

**Return Value:** Boolean TRUE if the export completed successfully, otherwise returns FALSE.

**OpenDatabase(stringConnectString,stringPassword,  
stringTable|View,stringModel)**

**OpenDatabase** opens an ISAM database file or an ODBC data source.

This method is used in place of the **SetReportFile** method, which is used to open one or more report files.

An automated Monarch session cannot open both the **OpenDatabase** and **SetReportFile** methods, as the Monarch Table window may be populated from only a single source, either a database or a series of reports.

*ConnectString* is either a string representing the entire ODBC connection string or a string containing the path to the data source on the network or the local drive.

*Password* is a string that can be up to 32 characters long which is used to open a password protected data source.

*Table | View* is the name of the table or view from which to import data. This parameter is not used when opening an ISAM file that does not support multiple tables.

*Model* is the path and file name of the model file to use for the session. A model is required for this method. The model contains parameters needed to complete the database import. If the model also contains join parameters

specifying a join to a password protected ODBC source, a **SetJoinPassword** method could be issued prior to opening the model file.

**Return Value:** Boolean TRUE if the database was opened successfully, otherwise returns FALSE.

### **PRFPublish(stringPublishFileName, booleanIncludeTree, booleanIncludeModel, integerIncludeTable, booleanEncrypt)**

**PRFPublish** publishes a portable report file (PRF) from a currently open report and model.

*PublishFileName* is a string indicating the name of the PRF output file. It could also include the path.

*IncludeTree* is a Boolean value indicating the inclusion or exclusion of the navigation tree. If *IncludeTree* is true, the navigation tree is included. If *IncludeTree* is false the navigation tree is omitted.

*IncludeModel* is a Boolean value which when true includes the model file and when false excludes the model file.

*IncludeTable* is an integer value which when non-zero includes the internal database and when 0 excludes the internal database.

*Encrypt* is a Boolean value - if true, the PRF file is encrypted, if false the PRF file is not encrypted.

**Return Value:** Boolean TRUE if the publish completed successfully, otherwise returns FALSE.

### **PrintReport(booleanDirectPrintFlag)**

**PrintReport** sends the contents of the Report view to the printer.

If *DirectPrintFlag* is true, all pages in the report are printed using the default printer settings. If *DirectPrintFlag* is false, the Print dialog will be invoked, giving the user the opportunity to adjust settings before printing.

### **PrintSummary(booleanDirectPrintFlag)**

**PrintSummary** sends the contents of the Summary view to the printer.

If *DirectPrintFlag* is true, the entire summary is printed using the default printer settings. If *DirectPrintFlag* is false, the Print dialog will be invoked, giving the user the opportunity to adjust settings before printing.

### **PrintTable(booleanDirectPrintFlag)**

**PrintTable** sends the contents of the Table view to the printer.

If *DirectPrintFlag* is true, the entire table is printed using the default printer settings. If *DirectPrintFlag* is false, the Print dialog will be invoked, giving the user the opportunity to adjust settings before printing.

### **RunAllExports()**

**RunAllExports** executes all Project Exports in the currently loaded Project.

Note that a call to **SetProjectFile** must have been made previously and a Project Export of the specified name must exist.

**Return Value:** Boolean TRUE if the exports were performed successfully, otherwise returns FALSE.

### **RunExport(stringProjectExportName)**

**RunExport** executes a Project Export of the name specified in *ProjectExportName* in the currently loaded Project.

Note that a call to **SetProjectFile** must have been made previously and a Project Export of the specified name must exist.

**Return Value:** Boolean TRUE if the export was performed successfully, otherwise returns FALSE.

### **SetDataSourcePassword(stringLookupName,stringPassword)**

**SetDataSourcePassword** sets the password for an external lookup specified by *LookupName*.

More than one call to **SetDataSourcePassword** can be made to specify password for each lookup associated with a model. This method must be called before calling **SetModelFile**.

Unlike the single password set with **SetJoinPassword**, passwords set with **SetDataSourcePassword** apply only to the very next model load, after which they are discarded. A password set with **SetDataSourcePassword** takes precedence over one set with **SetJoinPassword**. A password set with **SetDataSourcePassword** always gets applied, even if it's empty.

### **SetFieldVisible(stringFieldName,booleanVisibleFlag)**

**SetFieldVisible** specifies whether a hidden field is displayed or not.

*FieldName* is a string that is up to 62 characters long.

*VisibleFlag* is a Boolean value indicating whether the hidden field is displayed or not. If *VisibleFlag* is true, the hidden field is displayed. If *VisibleFlag* is false, the hidden field is not displayed.

If the model for the report contains any hidden fields, then those fields will not be displayed unless they have been set visible through this function.

### **SetFirstView(stringWindowType)**

**SetFirstView** specifies which Monarch window will be displayed the first time that a window is invoked.

If *window type* is "T" the Table window is displayed, if it is "S" the Summary window is displayed, and if it is "R" the Report window is displayed. If this method is not issued, or an invalid value is used, the Report window is the default.

This method does not cause the Monarch window to be displayed. If *window type* is "R", the Report window will be displayed when a subsequent **SetReportFile** method is issued. If *window type* is "T" or "S", the corresponding Table or Summary window will be displayed when a subsequent **SetModelFile** method is issued.

The **SetFirstView** method has no effect upon a **SetProjectFile** method.

### **SetInputCharacterSet(stringCharacterSet)**

**SetInputCharacterSet** establishes the character set, ANSI or ASCII, for interpreting report data that is loaded via the **SetReportFile** method.

This method overrides the InputCharacterSet default setting found in the Monarch\\Settings section of the Windows registry only for reports loaded

via the **SetReportFile** method. Reports opened interactively are always interpreted based upon the InputCharacterSet registry setting. *CharacterSet* is a string value of either “ANSI” or “ASCII”.

### **SetJoinPassword(stringPassword)**

**SetJoinPassword** establishes the password to be used when opening a password protected data source.

*Password* is a string that can be up to 32 characters long. A **SetModelFile** method or an **OpenDatabase** method should follow this method.

### **SetLogFile(stringLogFile,booleanAppendFlag)**

**SetLogFile** establishes the file specified as *LogFile* as a log file for Monarch error and export messages.

If *AppendFlag* has a value of true, new messages will be appended to the end of the existing log file. If *append flag* is false, the previous contents of the log file are deleted.

If this method is not used, error or status messages will be written to a new file named FILExxxx.ERR in the current working directory (where *xxxx* is unique number, beginning with 0001, which is incremented for each Monarch session).

**Return Value:** Boolean TRUE if the log file was established successfully, otherwise returns FALSE.

### **SetModelFile(stringModelFile)**

**SetModelFile** opens the model specified by *ModelFile* for processing.

If the full file name (including drive and path) is not supplied, the default Model Files location stored in the Monarch defaults will be used. If a model file is already open when the **SetModelFile** method is called, the previously opened model is closed before the new model is opened.

The **SetModelFile** method may also cause the Monarch Table or Summary window to be displayed. If a **SetFirstView** method has been issued with *window type* set to “T” or “S”, the appropriate window will be displayed. If a **SetFirstView** method has been issued with *window type* set to “R”, or if no **SetFirstView** method has been issued, then the window remains unchanged.

**Return Value:** Boolean TRUE if the file was opened successfully, otherwise returns FALSE.

### **SetOutputCharacterSet(stringCharacterSet)**

**SetOutputCharacterSet** establishes the character set, ANSI or ASCII, for exporting data to text and Delimited text formats.

This method overrides the OutputCharacterSet default setting found in the Monarch\\Export section of the Windows registry. Reports that are exported interactively are always interpreted based upon the OutputCharacterSet registry setting. *Character set* is a string value of either "ANSI" or "ASCII".

### **SetPRFFile(stringFileName,stringPassword)**

**SetPrffFile** opens the portable report file specified by *FileName* for processing.

If the full file name (including drive and path) is not supplied the default Publish Files location stored in the Monarch defaults will be used. The *Password* can be a string up to 32 characters in length.

**Return Value:** Boolean TRUE if the file was opened successfully, otherwise returns FALSE. If the wrong password is entered, **SetPrffFile** returns FALSE and the PRF file is not opened.

### **SetProjectFile(stringProjectFile)**

**SetProjectFile** opens the Project (XPRJ/PRJ) file specified by *ProjectFile* for processing.

If the full file name (including drive and path) is not supplied, the default Published Files location stored in the Monarch defaults will be used.

Any previously opened report or model files are closed before the **SetProjectFile** method is executed.

**Return Value:** If the report file(s) and model file referenced by the project or PRF file are all successfully opened, the method returns a value of boolean TRUE. Otherwise, it returns a value of FALSE.

**SetReportFile(stringReportFile,booleanAddFlag)**

**SetReportFile** opens the report file specified by *ReportFile* for processing.

If the full file name (including drive and path) is not supplied, the default Report Files location stored in the Monarch defaults will be used.

If *AddFlag* is true, the report file is added to the list of open reports. There is no limit to the number of reports that may be open at one time. If *AddFlag* is false all previously opened reports are closed before the new report file is opened.

The **SetReportFile** method may also cause the Monarch Report window to be displayed. If a **SetFirstView** method has been issued with *window type* set to "R", or if no **SetFirstView** method has been issued, the Report window will be displayed.

If a **SetFirstView** method has been issued with *window type* set to "T" or "S", the window remains unchanged.

**Return Value:** Boolean TRUE if the file was opened successfully, otherwise returns FALSE.

**SetRuntimeParameter(stringFieldName,stringFieldValue)**

**SetRuntimeParameter** allows the setting of a runtime parameter value of the Runtime Parameter field specified by *FieldName* to the value specified in *FieldValue*.

**SetRuntimeParameter** may be called either *before* or *after* a model is loaded.

If **SetRuntimeParameter** is called before loading the model, then the value is cached so that it becomes effective at the next model load. Multiple calls may be made so that one can set all the runtime parameters for a particular model before actually loading the model, and thereby suppress the Runtime Parameters dialog that would otherwise pop up at model load time. Calls to **SetRuntimeParameter** that occur before loading a model are always successful, since there is no field list against which to test field names or types.

If **SetRuntimeParameter** is called after a model is loaded, the new value thus provided takes effect immediately. Such calls will fail if the specified field name can't be found in the model, or if the specified value cannot be converted to the proper data type.

**Return Value:** Boolean TRUE if the file was parameter was set successfully, otherwise returns FALSE.

### **SetTextAppend(booleanAppendFlag)**

**SetTextAppend** determines whether Monarch will overwrite or append to an existing text or delimited text file.

This method is called prior to calling the **ExportTable** or **ExportSummary** method. If *Append Flag* is false, the existing text or delimited text file will be overwritten. If *Append Flag* is true, data will be appended to an existing file.

### **SetView(stringWindowType)**

**SetView** specifies which Monarch window will be displayed.

If *windowtype* is "R", the Report window is displayed, if *windowtype* is "T" the Table window is displayed, if *windowtype* is "S", the summary window is displayed. If this method is not issued, or an invalid value is used, the Report window is the default.

### **Version()**

**Version** returns the Monarch version number.

**Return Value:** String in the form "Version X.XX".

### **WriteToLogFile(stringUserLogInfo)**

**WriteToLogFile** writes the specified *string* to the currently established log file.

If a log file has not been explicitly established using **SetLogFile**, the string will be written to the currently open default log file, which is named FILExxxx.ERR (where *xxxx* is a unique number, beginning with 0001, which is incremented for each Monarch session).

## CHAPTER 3

# Programming Monarch Methods and Properties

## Monarch COM Registration

The first time that Monarch is run, it registers itself as an COM server. If the program being launched is the 32-bit version the server is named “Monarch32” (Version 3 or later). Once registered, the server becomes available for COM transactions.

## Calling the Monarch COM Server from a Client Application

Before the client application can use any of the Monarch methods, it must create the COM object that will contain pointers to the Monarch COM server. Once this object has been created, the client application has control of Monarch.

In Visual Basic, the statement used to create the object would take the form,

```
Set MonarchObj=CreateObject(“Monarch32”) (V3 or later)
```

Once the object has been created, you may use any of the Monarch methods.

## Creating the Appropriate Client Object

A client application can be written to run independent of which version of Monarch is being launched. Using the appropriate commands from your chosen programming language, the code would attempt to launch the 32-bit version. Following is sample code showing how this would be done in Visual Basic:

```
Sub Form_Load()
    Dim MonarchObj as Object
    '
    'Set up to trap errors
    '
    On Error Resume Next
    '
    'Attempt to create an object (Monarch version 3, or later)
    '
    Set MonarchObj = CreateObject("Monarch32")

    'Go back to default error handler
    On Error GoTo 0

    'Display error message if no Monarch object created

    If Err.Number <> 0 Then
        MsgBox "Cannot create Monarch OLE object", 32, "Error"
        CloseApp
    End If
End Sub
```

## Program Control and Termination

When the client application makes a call to one of the Monarch methods or properties, the request is routed through the COM API. Control is not passed back to the client application until the method has completed operation. This ensures that a new operation is not started until the previous one has properly terminated.

Monarch operates only as a single-instance server. It cannot be used by more than one program (or user) at a time. No other application or user can make use of Monarch until it is released by the client application. Monarch is automatically released when the client application terminates. It can also be explicitly released using the **Exit** method.

## Program Subroutine Example

Following is a sample Visual Basic subroutine that invokes Monarch, opens a report and model, applies a couple of filters and exports the resulting table to Excel:

```
Private Sub Form_Load()

Dim MonarchObj As Object

Dim openfile, openmod, t As Boolean

'If Monarch is currently active GetObject will use Monarch. If it is not use the CreateObject() to
'open another copy of Monarch.

Set MonarchObj = GetObject("", "Monarch32")

If MonarchObj Is Nothing Then

    Set MonarchObj = CreateObject("Monarch32")

End If

t = MonarchObj.SetLogFile("C:\MonTemp\MPrG_G5.log", False)

openfile = MonarchObj.SetReportFile("C:\Program Files\Monarch\Reports\Classic.prn",
False)

If openfile = True Then

    openmod = MonarchObj.SetModelFile("C:\Program Files\Monarch\Models\Lesson14.mod")

    If openmod = True Then

'Set filter for each frame and export to Excel

        MonarchObj.CurrentFilter = "Fandangos records"

        MonarchObj.ExportTable ("C:\Program Files\Monarch\Export\Fandangos.xls")

        MonarchObj.CurrentFilter = "No Returns"

        MonarchObj.ExportTable ("C:\Program Files\Monarch\Export\No Returns.xls")

    End If

End If
```

End If

MonarchObj.CloseAllDocuments

MonarchObj.Exit

End Sub

## CHAPTER 4

# Launching a Monarch Session from an Application

Some application developers may want to use COM to launch an interactive Monarch session from their application. This is most useful if you intend to use Monarch as a report viewing and analysis tool to augment a document management system or report archive system. Using COM, you can launch directly into the Monarch viewer. The end user can then use Monarch to search the report, perform analysis, or extract and export data. The integration with Monarch is seamless and simple.

## Launching an Interactive Monarch Session

To launch Monarch, simply create the object. Monarch will launch automatically, presenting you with the main menu screen. You may also load report and model files, set a filter, apply a sort and do other processing as part of the launch process.

Whenever you create the Monarch object, you need to ensure that you don't attempt to create it again. A call to create the object a second time would produce an error which would cause the program to terminate. Similarly, a call to exit Monarch cannot be validly made unless the object has been created. You will want to create a function to check to see if the server is active before issuing a command.

### Program Subroutine Example

Following is a subroutine that a Visual Basic programmer would add to his or her application's General object. (The declaration `Dim MonarchObj as Object` must appear in the General object.)

'This is the Main routine

```
Sub Main()  
Monarch_Launch.  
Do While IsServerActive()  
    DoEvents  
While  
End Sub
```

---

'The following subroutine illustrates the Launching process

```
Sub Monarch_Launch ()  
Dim openfile, openmod as Boolean  
Dim ServerOn as Integer  
  
ServerOn=IsServerActive()  
If ServerOn=0 then  
    Set MonarchObj = CreateObject("Monarch32")  
End If  
  
,  
' Open CLASSIC.PRN as the report file and LESSON12.MOD as the model  
,  
openfile = MonarchObj.SetReportFile("C:\Program Files\Monarch\Reports\classic.prn", False)  
If openfile = True Then  
    openmod = MonarchObj.SetModelFile("C:\Program Files\Monarch\Models\Lesson12.mod")  
End If  
  
End Sub
```

---

'This subroutine checks to see if Monarch is Active

```
Function IsServerActive()  
    On Error Goto NoServer  
    If MonarchObj.IsActive > 0 then  
        IsServerActive = 1  
    End If  
Exit Function  
NoServer:  
    IsServerActive = 0  
Exit Function  
End Function
```

## A p p e n d i x A

## Error Messages

**OLE Automation Server cannot create object.**

This error message is issued when the Monarch COM routines are inaccessible to the calling program. This error will occur if the routines have not been registered or are already in use. You can check that the routines are registered by running REGEDIT.EXE and examining the list of program registrations. If another program is using the routines, that program must either terminate or issue an Exit command to release the COM routines.

You may also want to check that Monarch is installed properly.





## A p p e n d i x C

## Technical Support

Datawatch provides technical support to all registered owners of Monarch. Technical support for using Monarch programmatically is limited to questions regarding specific methods. We do not provide application development support. You may contact us at:

**In the U.S.A.:**

Datawatch Corporation  
Suite 503  
175 Cabot Street  
Lowell, MA 01854

FAX: 978-454-8886  
Phone: 978-441-2200

**In Europe:**

Datawatch Europe Ltd  
The Software Centre, East Way  
Lee Mill Industrial Estate  
Ivybridge  
Devon  
PL21 9GE  
UK

FAX: +44-(0)1752 894 833  
Phone: +44-(0)1752 893 100

Before contacting Datawatch, please re-read the relevant sections of the Monarch documentation - you may find a solution that you previously overlooked.

Technical support is available by telephone Monday to Friday, from 9:00 a.m. to 5:00 p.m. (Eastern Time if calling our U.S. office or Greenwich Mean Time if calling our European office).