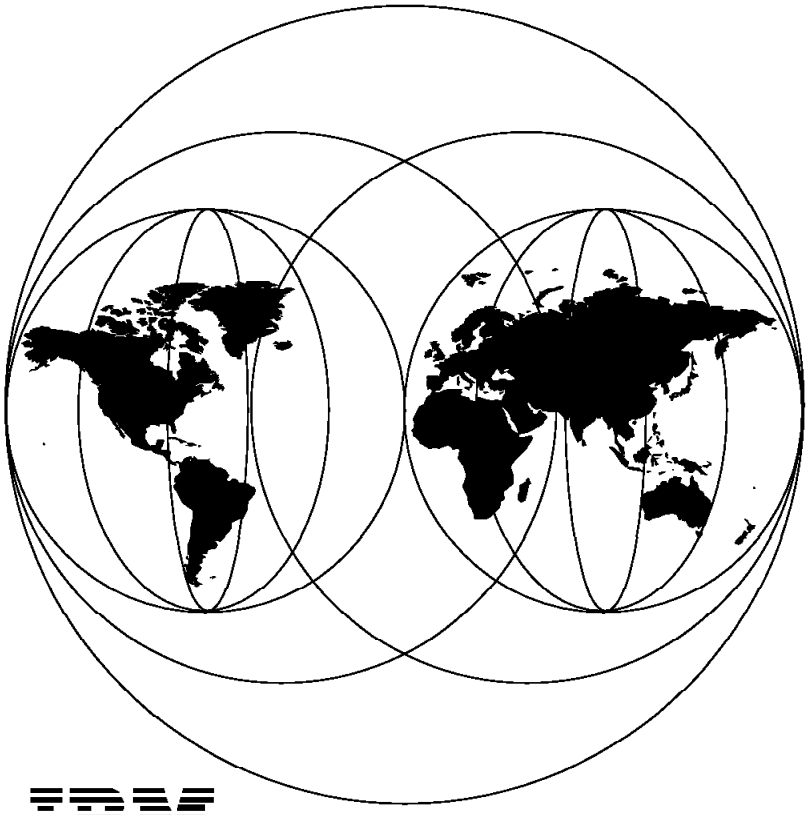
SG24-2184-00

Image and Workflow Library: Sample Applications for FlowMark

October 1997



International Technical Support Organization
Poughkeepsie Center

International Technical Support Organization

SG24-2184-00

Image and Workflow Library: Sample Applications for FlowMark

October 1997

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix E, “Special Notices” on page 147.

First Edition (October 1997)

This edition applies to Version 2 Release 3 of IBM Flowmark, Program Number 5697-216 for use with the Windows NT, Windows 95 or OS/2 operating systems.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Preface

Do you want to integrate structured workflow into your business processes but need some programming samples to get started? This redbook on IBM FlowMark V2.3 is written for you.

We begin with an introduction to FlowMark and its application programming. Then, using a business workflow scenario, we list some tasks and show how to program them in a FlowMark environment. We also include some application development techniques. A compact disk is provided that contains source for the samples.

This redbook is written for technical professionals who are involved in producing FlowMark demos, services or customizing run-time clients. The information applies to Windows NT, Windows 96 and OS/2 operating systems.

We have assumed that you have some knowledge of FlowMark and your programming environment and tools.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the IBM German Software Development Lab (GSDL) in Boeblingen.

Eric Eller is a Senior Systems Consultant and Developer with SYSCOM, Inc., a Baltimore, Maryland based leader in systems integration, document management and enterprise workflow solutions. Eric has over 8 years of systems development and integration experience. He holds a graduate degree in mechanical engineering from The Johns Hopkins University. His areas of expertise include cross-platform systems development, automated workflow modeling, and integration of high volume workflow and document management solutions. Eric may be reached through e-mail at eeller@syscom-inc.com or by accessing www.syscom-inc.com.

Carlos Gutierrez is a Solution Architect with IBM Argentina. He has 5 years of experience in solution architecture. He holds a graduate degree in computer science from Universidad de Buenos Aires. His areas of expertise include image, workflow and document management.

Jon Seymour is a Software Engineer with IBM Global Services Australia. He has 7 years of experience in software development and services. His areas of expertise include software tool implementation, workflow technology, and object oriented programming and design.

Tomislav Begovac is at the IBM FlowMark Competency Center in Boeblingen responsible for FlowMark technical support in EMEA (Europe, Middle East, and Africa). Before his current assignment, he was managing the development tools department in the IBM German Software Development Lab. Tomislav has coordinated this project and can be reached at beg@de.ibm.com.

This project was sponsored by the International Technical Support Organization (ITSO).

Mike Ebbers is a Senior Software Specialist at the International Technical Support Organization, Rochester/Cambridge Center. He has worked for IBM for 23 years. He writes extensively on all areas of workflow and imaging. Before joining the ITSO 3 years ago, Mike worked in Education and Training, U.S.A. as an instructor/developer

Thanks to the following people for their invaluable contributions to this project:

International Technical Support Organization, Boeblingen Center

Wolfgang Kulhanek
IBM German Software Development Laboratory
FlowMark Competency Center, Boeblingen

Denise Bailey
IBM Solution Developer Marketing
FlowMark Technical Support, Dallas

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 167 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users `http://www.redbooks.ibm.com`
  For IBM Intranet users `http://w3.itso.ibm.com`

- Send us a note at the following address:

  `redbook@vnet.ibm.com`

Chapter 1. Introduction

Workflow management helps you manage and control your business processes, pinpoint areas for improvement, and streamline your procedures for speedier cycles and shorter response times. When you define the flow of work, each person gets notified of tasks that need to be completed, and is presented with the information and an appropriate application to perform the task.

It is common to write a customized workflow application for all but the most simple processes. Integrating FlowMark with office applications, electronic mail, legacy databases, and the Internet is becoming common for most implementations. Also, most companies have a variety of computers and operating systems that must be supported by the workflow solution. Therefore, developing the skills to integrate FlowMark with these systems on various platforms is important to the success of any implementation.

1.1 Objectives

The objectives of this redbook are:

- To describe FlowMark integration with various technologies such as:
  - ActiveX
  - The Internet/World Wide Web
  - Office Applications using OLE Automation
  - Databases using ODBC
  - Java
- To use various development environments and languages, for example:
  - Java
  - HTML/JavaScript/Net.Data
  - Powerbuilder
  - REXX
  - Microsoft Visual Basic
  - Microsoft Visual C++
- To demonstrate integration techniques with an example scenario of a process.
- To highlight key design strategies.
- To provide a usable demonstration of the example process.

1.2 How This Book is Organized

First, a scenario is presented that conveniently includes requirements for all of the topics we want to cover. Although simple, the scenario tries to reflect an actual real world customer situation where significant FlowMark integration is required. An overview of the business process provides descriptions of the actions to be performed by the customers, by the employees, and by the system to complete the order placement and production of the product.

The process modeling of this scenario is discussed in the next chapter. Here, the factors that shaped the final design are described, and the decisions we made are explained. Each activity in the processes is detailed so that it is clear

which programs are to run, how they are run, what data containers are used, and how they can affect decision points further along in the process. To enhance these examples, certain sub-processes are shown with different activity programs, thus necessitating separate process models. The purpose of each of these ″duplicate″ sub-processes is also explained.

The following chapters in the book are dedicated to in-depth reviews of the example activity programs and auxiliary programs. Each chapter covers one program in detail, summarizing the techniques involved and illustrating them with plenty of program code fragments. When applicable, additional information about the technique is presented along with possible alternative solutions for the integration program.

Finally, there are several appendixes (A,B,C and D) to aid in installing and configuring the example programs discussed in this book and included on a compact disk (inside the back cover).

1.3 Workflow Coalition Interfaces and FlowMark

As stated in the objectives, this redbook discusses sample applications that show how to integrate various technologies with Flowmark. Although several FlowMark processes are used as a basis for the sample applications, the development of the process models is not discussed in this book.

Within the workflow product industry, there is an organization who defines workflow system standards, so that applications can be written independently of a specific workflow system. Independent workflow systems should be able to work together in an environment where they coexist. The Workflow Management Coalition has provided a reference model of a generic workflow system that defines the possible interfaces for other workflow systems and for independent workflow applications. This is shown in Figure 1.

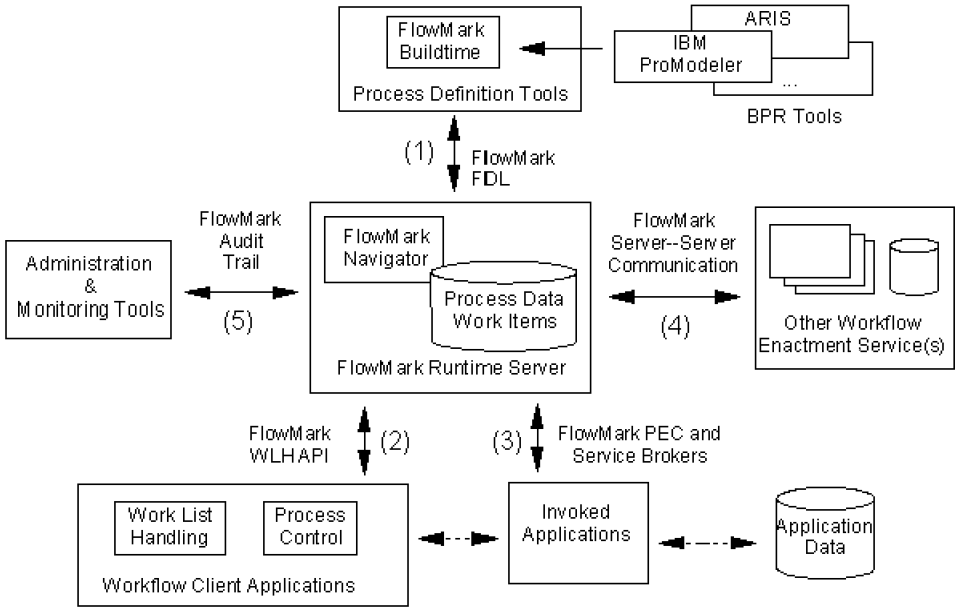


Figure 1. Workflow Coalition Interfaces

Notice that there are five interfaces defined. This redbook concentrates on applications designed for interfaces two and three. These interfaces define the

integration of the workflow system with workflow client applications and invoked applications respectively. Workflow client applications are programs that perform actions such as starting a process, querying a work list, or starting a work item. Invoked applications are the programs that are initiated by the workflow system as part of a running process. These applications are the program activities in FlowMark terminology that are required to complete the business process represented by the workflow process.

FlowMark provides a default implementation for interface two, called the Run-time Client programs. These programs provide the ability to manipulate process templates and instances, view user information, and display and start work items. Custom run-time clients can be created with the supplied C++ work list handler API set, the FlowMark ActiveX control set, the Lotus Notes API set, or the Workflow Coalition C API set. The so-called language APIs (C, Visual Basic, REXX, and COBOL) provide process control functions that supply the basic actions available in the run-time clients such as starting, suspending, and resuming processes. Several examples of custom run-time clients and process control programs are included in this redbook.

Interface three is handled by the FlowMark process execution client and the service broker technology. FlowMark integrators can use many provided language APIs such as C, REXX, Visual Basic, and COBOL to access the FlowMark data container and process information from within program activities. The C++ work list handler APIs, the FlowMark ActiveX control set, and the Lotus Notes APIs can also be used for program activity implementation. Custom run-time clients can start application programs just like the default FlowMark client, or they can be designed as integrated program activity applicati ons that process work items without using the program execution client.[1] This book includes sample program activity applications that use most of the available options.

1.4 FlowMark Application Integration

When a developer or programmer first approaches the FlowMark product, there are sometimes misconceptions about what the product is, what it does, and what services it provides. An exhaustive answer to these questions is best left to the FlowMark documentation, but we will try to clarify certain aspects of Flowmark as they relate to creating FlowMark integration applications.

FlowMark is a workflow management system that helps organizations to define, document, test, control, execute and improve their business procedures. It provides facilities for creating a model of your business and for bringing work to your employees, along with the appropriate business application to accomplish the task.

When we talk about FlowMark application integration, we mean using the tools provided by the FlowMark product to satisfy the requirements of the modeled business processes. This is accomplished by using the correct combination of the supplied programs and custom programs developed with the FlowMark APIs. This often requires the development of one or more applications to create and manipulate process instances and many applications for the program activities

[1] However, if the client uses the ExmWorkitem::Start() method, then the program execution client is required.

within the process model. These two types of programs interact with the FlowMark workflow system through the two interfaces described in the previous section.

Depending on the functionality required and the environments available, different levels of integration may be required. These types of integrations commonly fall into one of the following categories:

- Simple invocation
- Building blocks
- Language APIs
- Service brokers

Simple program invocation is a form of integration that is handled completely by the existing FlowMark programs. For example, if a program activity just needs to launch a program, such as a 3270 or 5250 emulator, the file name is specified in the program registration for that activity. When the activity is started, the emulator program is loaded and the activity completes. With this type of integration, data container information can be passed to the program as command line parameters using replacement variables, but output data container members cannot be set.

The next level of integration uses **building blocks**, which are programs or script files supplied with FlowMark or available from other sources. These blocks usually supply methods for performing some of the most common actions with FlowMark. Building blocks are the same as mini-applications and are generally configurable, but are not truly flexible enough to handle any task. Therefore, it is useful to search for an existing building block that may make your integration easier, but you may have to resort to actual API programming. An MQSeries building block is included with the FlowMark product. It allows a system to start, suspend, resume, and terminate a process on another FlowMark system by using MQSeries.

Application integration with the **language APIs**, such as C, REXX, or Microsoft Visual Basic, is the most common form of integration. These APIs provide the full suite of functions to control processes and manipulate data containers.

The **work list handler** C++ APIs are also supplied as ActiveX controls. This method of integration is the most flexible and potentially the most complex, but it provides the only means for creating a custom run-time client. With a custom run-time client, there are no limitations to the type of interface that can be designed for the user. The only component that is not available in the current FlowMark release (2.3) is the graphical process monitor, which is part of the default FlowMark run-time client.

The **service broker** concept is designed to allow users of workflow systems or other applications to work with multiple tools in multiple interactions without the need to reload the tool each time or perform multiple logons to server sessions. This allows all required sessions and tools to be available during the work session without the users needing to be aware of the application execution or logic. The details for developing and implementing brokers and services are documented in the *IBM FlowMark: Application Integration Guide, SH12-6267*.

1.5 Guide to Examples

The following table is provided as a means for quickly looking up the topics that interest you. For each example topic that is covered in the redbook, the appropriate chapters where the topic is discussed are noted.

Table 1. Example Topic and Chapter Cross Reference

Example Topic	Chapter										
	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
ActiveX		X		X	X		X	X			X
Custom Run-Time Clients							X	X	X	X	
Database Access	X	X	X	X	X						X
Internet/World Wide Web	X					X				X	
Java						X				X	
Net.Data	X										
OLE Automation/Office Integration		X		X	X						X
Power Builder							X				
REXX	X		X								
Microsoft Visual Basic		X		X	X			X			X
Microsoft Visual C++									X		

Chapter 2. Scenario Description

To demonstrate FlowMark application development and integration with various systems, we needed to create certain activity programs. Instead of showing unrelated programs for each point we want to demonstrate, we have written a sample scenario that contains all of the situations and integrations to be discussed.

2.1 The SignCo Company

This scenario involves a company that makes custom signs. They want to enhance their process by using workflow and by allowing customers to place orders through the Internet. The customers can specify certain attributes such as the type of material, the text on the sign, the text font and size, and the surface finish of the sign. Based on the specifications for the sign, a price is determined and quoted to the customer. If the customer agrees to the price, the order is confirmed and the process moves to the production phase.

In the production phase, the design details are retrieved and a series of manufacturing steps are performed to complete the sign. Each manufacturing task requires specific information about the sign, and for efficiency, the work must be presented in certain orders or groupings for each task. After it is completed, the sign and a bill are sent to the customer.

2.1.1 Order Entry Phase

The customer has two options for placing an order. The first is through the traditional channels of visiting the store or placing an order over the telephone. Here, an application at the company′s site displays the customer price at the time the order information is entered. If the customer agrees to the price, this application updates the customer and order databases and creates a FlowMark process instance for the production phase.

The second method for placing an order is through the company′s new site on the World Wide Web. A Web page allows the customer to enter the design details for the sign they want created. When all of the form fields are completed and the form is submitted, a FlowMark process instance is created to prepare a price quote for the customer.

The Prepare Quote process begins by calculating the price of the sign. For this step, a spreadsheet document is used for the calculation and updated with the order information. The order table is also updated with the calculated price. Next, an automatic activity reads the customer information into the data container for the following steps. The next step in the process is to create a confirmation letter for the customer using a word processing application. The letter is automatically created and formatted based on the information in the data container. If an e-mail address was not provided by the customer, the letter is printed and mailed to the customer. If the customer provided an electronic mail address, the file is saved to disk and the next activity, which runs automatically, sends the electronic letter to the customer′s e-mail address. The final step in this process is to update the tables to reflect that the confirmation letter has been sent.

Once the confirmation letter has been sent to the customer, the order has been placed but not yet confirmed, and the production process has not been initiated. Again, the customer has two methods to confirm the order and begin manufacturing their sign. First, the customer may call the customer service representative who can take their confirmation over the phone. The same application used to take a new order can bring up the details of an existing order, confirm the order, and initiate a FlowMark instance for the production process.

The second way a customer can confirm an order is by revisiting the SignCo Web site. From there, a confirm order option takes them to a separate page where the order number and a secret confirmation number must be entered. This confirmation number is mailed to the customer with the confirmation request letter. Once the form is submitted, the FlowMark production process instance is created and started.

2.1.2 Production Phase

The production process, Manufacture Sign, begins with two automatic activities. The first updates the ODBC tables to show that the production process has begun. The second automatic activity reads the design specifications from the database into the data container for use by later activities.

Next, the workflow enters the set of activities that represent the actual manufacturing steps. The people responsible for the actions during this phase may have specific requirements and specialized computer equipment. So, custom FlowMark run-time clients are used for cutting, engraving, filling, and surface finishing activities.

When the custom sign is completed, it must be labeled and shipped to the customer. The final activity in the sample process scenario retrieves customer information from the database, formats a shipping label, and prints the label for use. At the end of this step, the product is sent.

Chapter 3. FlowMark Model Description

This chapter describes the FlowMark model used for scenario described in Chapter 2, “Scenario Description” on page 7.

We have defined five processes:

- Prepare Quote
- Manufacture Sign
- Manufacture Web
- Manufacture Non Web
- Insert Customer

3.1 Prepare Quote Process

The Prepare Quote process shown in Figure 2 is started when the customer uses a Web form to submit a request for a quote. This is described in more detail in Chapter 4, “Starting a FlowMark Process from the Web” on page 17. The process calculates the price for the quote using a spreadsheet, formats the quote with a word processor and then, if the customer′s e-mail address is known, sends the quote to the customer by e-mail. At the conclusion of this process, the database is updated to show that the quote has been prepared.

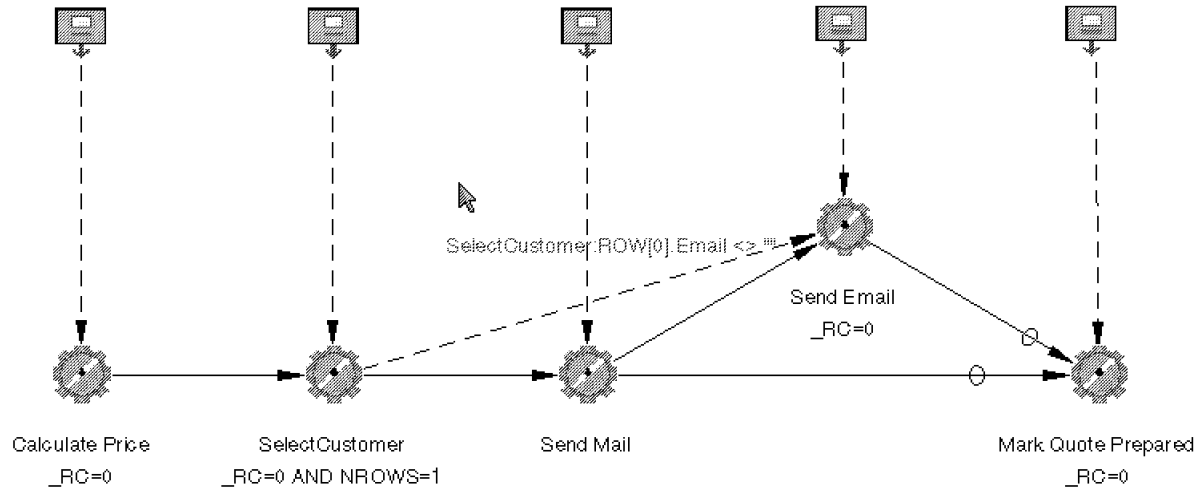


Figure 2. Prepare Quote Process

All the details of an order are stored in an external database. The only detail of the order that needs to be passed in the input container of the process is its key. This is because the activities within the process perform their own database access, if needed.

As you can see, there are many source symbols in Figure 2. Rather than using just one source symbol, we used several to improve the presentation of the diagram.

3.1.1 Calculate Price Activity

This activity starts a Visual Basic application to calculate and store a price for the order in the database. The order number is provided in the input container.

The program used to carry out this activity is described in Chapter 7, “Integrating with Microsoft Excel” on page 49.

3.1.2 Select Customer Activity

This activity automatically retrieves the customer information from the database so that the customer′s e-mail address can be used in a later transition condition and as input to the ″Send E-mail″ activity.

The exit condition on this activity ensures that exactly one customer record is successfully retrieved from the database.

This activity is done by the *ExecSQL* program that is described in Chapter 6, “Using a Relational Database in a FlowMark Process” on page 39.

3.1.3 Send Mail Activity

This activity starts a Visual Basic application that uses the order number provided in the input container to retrieve the related order details from the database. The application uses these details to generate a Microsoft Word document from a template document. If an e-mail address is present in the order details, the document is saved as a plain text file so that it can be incorporated in an e-mail message generated by a later ″Send E-mail″ activity.

This activity is done by the *Send Mail Message* program that is described in Chapter 8, “Printing and Sending a Reply to a Customer” on page 57.

3.1.4 Send E-mail Activity

This activity starts automatically if there is an e-mail address associated with the customer order. Its function is to e-mail the plain text file prepared by the ″Send Mail″ activity to the customer.

This activity is done by the *Send E-mail Message* program that is described in Chapter 9, “Sending E-mail with Java” on page 63.

3.1.5 Mark Quote Prepared Activity

This activity automatically updates the order′s status to show that a quote for the order has been prepared.

This activity is carried out by the *ExecSQL* program that is described in Chapter 6, “Using a Relational Database in a FlowMark Process” on page 39.

3.2 Manufacture Sign Process

When the customer accepts the quote and confirms the order (either using the Web forms described in Chapter 4, “Starting a FlowMark Process from the Web” on page 17 or while talking with a sales representative on the phone), a new instance of the Manufacture Sign process is created and started. This process tracks the progress of the order through all the manufacturing steps until the order is dispatched to the customer.

Figure 3 on page 11 shows the diagram of Manufacture Sign process.

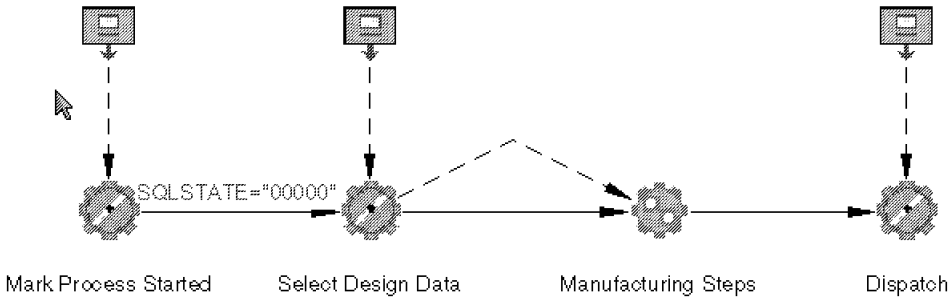


Figure 3. Manufacture Sign Process

All order information is stored in a database. The input container of the Manufacture Sign process is a simple structure that contains just the key of the order.

3.2.1 Mark Process Started Activity

This activity starts automatically and updates the order with its new status and the identifier of the related FlowMark process.

The SQL, exit condition, and following transition condition of this activity are used to ensure that multiple Manufacture Sign processes are not started for the same order. This protection is required because it is possible for a user of the ″Confirm Order″ Web form to inadvertently press the Submit button multiple times.

This activity is carried out by the *ExecSQL* program that is described in Chapter 6, “Using a Relational Database in a FlowMark Process” on page 39.

3.2.2 Select Design Data Activity

This activity starts automatically and fills its output container with the design details of the order.

This activity is done by the *ExecSQL* program that is described in Chapter 6, “Using a Relational Database in a FlowMark Process” on page 39.

3.2.3 Manufacturing Steps Sub-Process

The manufacturing steps involved in the production of a sign are: Cut Material, Engrave Text, Fill Text, and Finish Surface. Rather than create activity programs to do each step, we chose instead to create custom run-time clients. These run-time clients provide a single user interface that can be used to view the list of available work and also to carry out the work itself.

There are two possible implementations of the ″Manufacturing Steps″ sub-process. To select between these alternatives, simply edit the definition of the ″Manufacturing Steps″ activity and alter the name of implementing process. Choose ″Manufacturing Web″ if you want to exercise the Web-based run-time client. Otherwise, choose ″Manufacturing Non Web″ to exercise the other examples.

The input container to this sub-process consists of all the order's design details. By passing all this data in the input container, we can eliminate the need to perform database access within the activities of the implementing sub-process.

3.2.4 Dispatch Activity

This activity starts a Visual Basic application that formats and prints a shipping document for the order. It uses the order number provided in the input container to look up the relevant shipping details in the database and then substitutes these details into a template Microsoft Word document.

Source code and executables for this example can be found in the directory on the CD called \examples\dispatch. The Microsoft Word template should be copied into the directory that is specified as the first command line parameter of the FlowMark program registration, *Dispatch*.

3.3 Manufacture Non Web Process

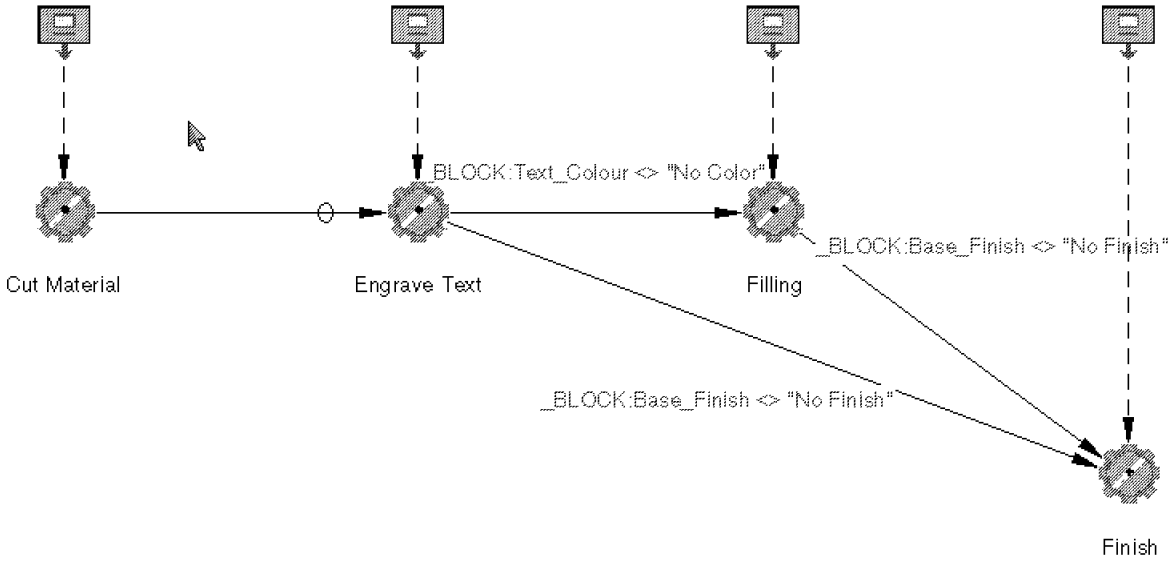


Figure 4. Manufacture Non Web Process

The diagram in Figure 4 illustrates the Manufacture Non Web Process. This process can be used as an implementation of the ″Manufacturing Steps″ activity of the ″Manufacture Sign″ process. The activities of this process are not done by normal FlowMark activity programs. Instead, custom run-time clients are used to both display the work list and to complete each activity on the work list.

For example, the run-time client described in Chapter 10, “Custom FlowMark Run-time Client - Powerbuilder Version” on page 69 is tailored to processing ″Cut Material″ activities while the run-time client described in Chapter 11, “Custom FlowMark Run-time Client - Visual Basic Version” on page 77 is tailored to processing ″Engrave Text″ activities. The run-time client described in Chapter 12, “Custom Run-Time Client - Visual C++” on page 85, while not as heavily optimized as the other two, can be easily reconfigured to process any of the four manufacturing steps.

Even though the activities of this process are not intended to be carried out by programs invoked by the FlowMark program execution client, the FlowMark build time forces us to assign a FlowMark program to each activity. We choose to assign the *No Execute* program to each activity. This program is a simple Visual Basic program that displays a message indicating that a custom run-time client should be used and then restarts the activity, thereby ensuring that the activity remains on the work list.

3.4 Manufacture Web Process

The diagram in Figure 5 illustrates the Manufacture Web Process. This process can be used as an implementation of the ″Manufacturing Steps″ activity of the ″Manufacture Sign″ process.

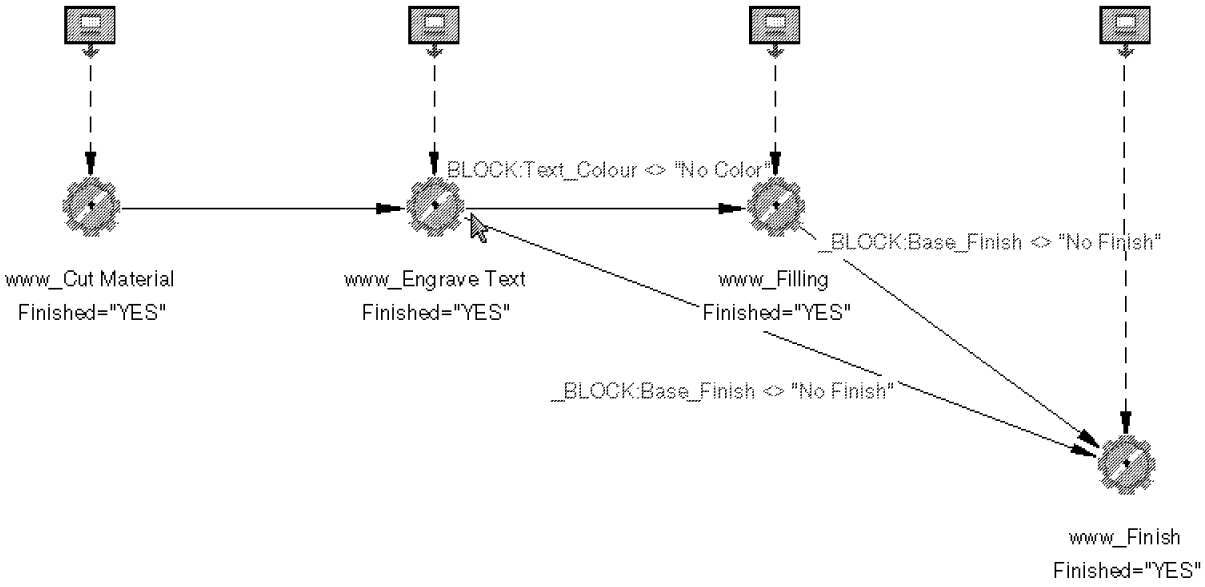


Figure 5. Manufacture Web Process

This process contains the same set of activities as the ″Manufacture Non Web″ process but differs from that process in that it has been tailored to allow it to be used from the WWW. The differences are as follows:

- The name of each activity is prefixed with www_.
- The input container of each activity is *Design Web*, a structure that contains the *Design* structure plus an additional member, *html_template_name*, that specifies the name of the HTML template to be used for the activity.
- The program registration is *WebEnabledActivity*, a program that is performed by a standard part of the IBM Internet Connection for Flowmark framework.
- The exit condition tests the value of a container variable to determine if the activity has completed successfully.

For a detailed description of the implementation of this process, refer to Chapter 13, “Web-Enabled Activities” on page 103.

3.5 Insert Customer Process

The Insert Customer process is used to illustrate some points described in Chapter 6, “Using a Relational Database in a FlowMark Process” on page 39. The top-level block of the process, shown in Figure 6, consists of a subordinate block called ″Retry SQL″.

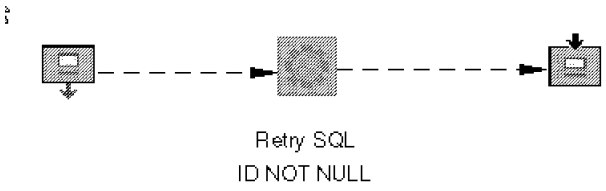


Figure 6. Insert Customer Process

The ″Retry SQL″ block (its decomposition is shown in Figure 7) executes a sequence of SQL statements until a customer record is successfully inserted into the database.

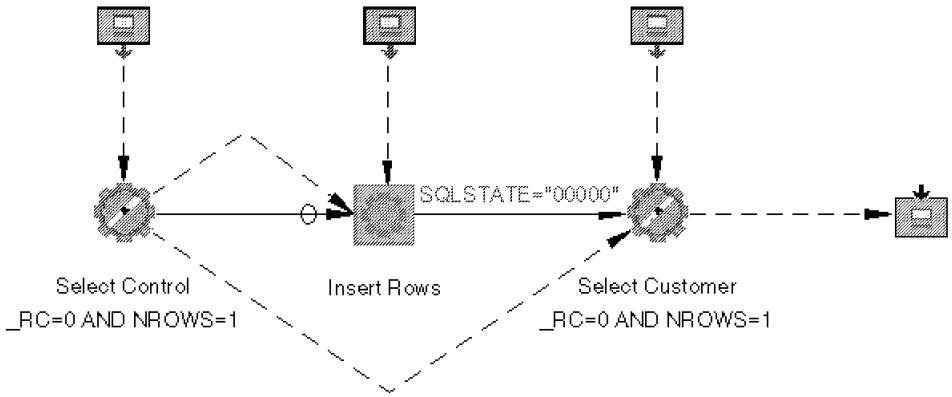


Figure 7. Retry SQL Block

Unlike the ″Retry SQL″ block, which uses an exit condition to ensure that the block is restarted until it succeeds, the ″Insert Rows″ block is used simply to group together a set of related activities that share the same input container. These activities are shown in Figure 8 on page 15.

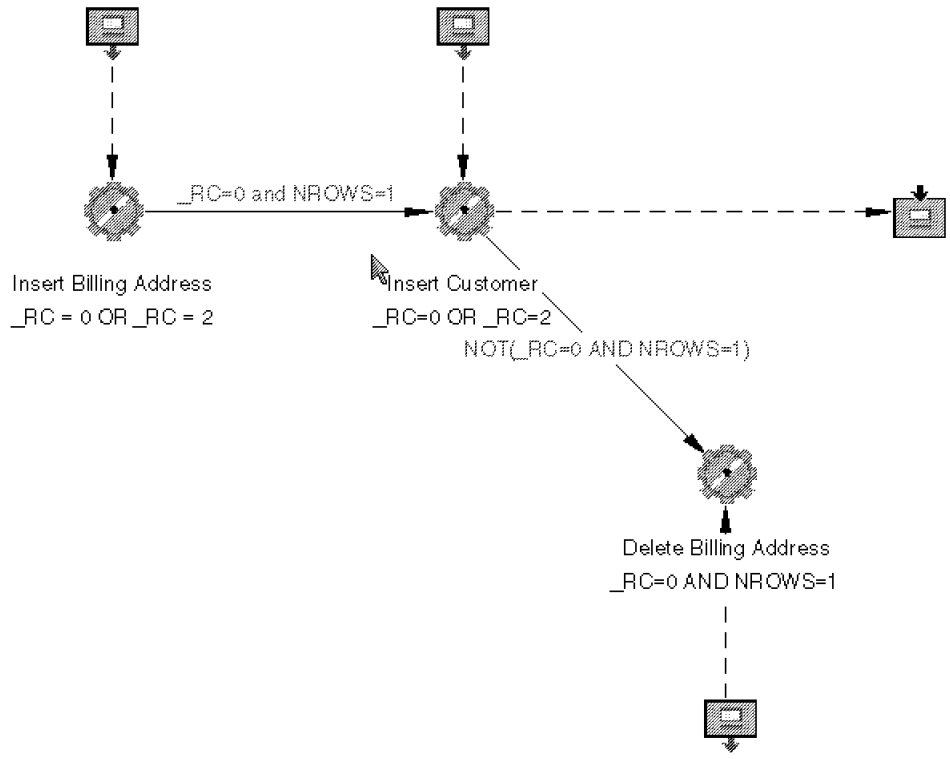


Figure 8. Insert Rows Block

3.6 Source Files and Installation

Source files:

- \examples\fdl\SG242184.fdl

Installation:

- Make a copy of \examples\fdl\SG242184.FDL.

  The following instructions apply to copying the .FDL file that you make in this step.

- Edit the following program registrations: ″Send E-Mail Message″, ″ExecSQL″, ″WebEnabledActivity″

  Change the TCP/IP ADDRESS declaration to equal the IP address of your OS/2 server.

- Edit the following program registrations: ″Send E-Mail Message″, ″Send Mail Message″, ″Dispatch″

  Replace occurrences of the string, ″r:\tmp″, with the name of a shared network directory that is common to both the Windows NT clients and the OS/2 server. If possible, use the UNC name of the shared directory so that you do not need to enforce identical drive letter mappings on all machines.

- Import the modified .FDL file into your FlowMark database.

  For details on how to import an FDL file, please refer to *IBM FlowMark: Managing Your Workflow, SH19-8243*.

- Select the implementation of the ″Manufacturing Steps″ sub-process in the ″Manufacture Sign″ process.

  Choose either ″Manufacture Web″ or ″Manufacture Non Web″ depending on whether you wish use the Web or the non-Web implementation of this sub-process.
- Translate the processes.

Chapter 4. Starting a FlowMark Process from the Web

Many companies are now conducting business across the Internet. These companies typically use a Web-based application to collect order details from the customer, which are then used by the company's internal processes to fulfill the order. If those companies also use FlowMark, they can initiate FlowMark processing for an order when the customer submits it, thereby providing end-to-end tracking of a customer's order.

There are many different tools and techniques for building interactive Web interfaces. Since our scenario involves the use of a relational database, a good choice of tools is IBM Net.Data (a tool to rapidly build HTML interfaces to relational databases). As this example shows, IBM Net.Data is also a suitable tool to use to build HTML interfaces to FlowMark applications, particularly in those cases where a relational database is also being used as an electronic document store by the FlowMark process.

4.1 Where This Example is Used

The first part of this example is used to collect order details from a customer using a Web browser. On receipt of these details, the Web server starts an instance of the Prepare Quote process to prepare a quote for the order. This example uses IBM Net.Data and the FlowMark REXX API to start a new FlowMark process.

The second part of this example is used by a customer to confirm the details of the order and send acceptance of the quote. In response to this indication, the Web server starts an instance of the Manufacture Sign process to coordinate the manufacture and delivery of the order. This example uses IBM Net.Data and the IBM Internet Connection for Flowmark to start a new FlowMark process.

4.2 What Techniques This Example Demonstrates

This example demonstrates:

- Integration between:
  - IBM FlowMark
  - IBM Internet Connection for Flowmark
  - IBM Net.Data
  - IBM Database 2
  - IBM Internet Connection Secure Server
- Two techniques for starting IBM FlowMark processes from a CGI-compliant Web server

4.3 Understanding Net.Data Web Macro Files

A Net.Data web macro file consists of a sequence of blocks. Blocks typically look similar to the following example:

```
% block_specific_declarator {
   block_specific_contents
%}
```

where the syntax of *block_specific_declarator* and *block_specific_contents* depend on the type of block involved. In our example, we use the following kinds of blocks:

- %define

  A %define block contains variable declarations that are used elsewhere in the macro file.

- %html

  An %html block contains an HTML template. Besides HTML, this template may contain macro variable references (for example: `$(variable)`) and function calls (for example: `@function(variable, variable)`). It is the ability to make calls to external functions that allows a Net.Data macro to imbed information from a relational database or interact with external APIs such as the FlowMark API.

- %function

  A %function block contains the declaration and definition of a function that can be called from other places in the Web macro file. The declaration of a %function block specifies its name (the name of its input and output parameters and return value) and the language environment that is responsible for interpreting the contents of the function block. In our example, we use two language environments: DTW_REXX and DTW_SQL. These environments provide access to the REXX and SQL languages, respectively.

  The contents of a %function block contains statements written in the language of the environment responsible for interpreting the block. As in an HTML template, the content section of a %function block may contain macro variable references and function calls. In such cases, Net.Data performs variable substitution and function evaluation on the contents of the block before passing the resulting text to the language environment for evaluation.

For more information about coding Net.Data web macros, refer to: http://www.software.ibm.com/data/net.data/docs/

Our example is contained in the file called signco.d2w. It is organized as a sequence of four parts: variable declarations, SQL functions, REXX functions, and HTML templates.

4.4 Displaying a Data Entry Form

To request a quote for manufacturing and delivering a new sign, a customer of our sign company accesses the following URL:

```
http://web-server/cgi-bin/db2www/signco.d2w/InputQuoteRequest
```

On receiving this request, the Net.Data cgi-bin program, db2www, looks in the Net.Data macro directory for a macro file called signco.d2w and within that file for a HTML block called InputQuoteRequest. Net.Data then interprets the contents of the block, evaluates any function references, and performs variable substitutions before sending the resulting HTML to the requesting Web client.

The `InputQuoteRequest` block uses the `SelectRowCustomer` function to read attributes of the customer from the database into variables with the prefix of `Customer`. These are used within the macro to set elements such as the HTML

title and default values for the shipping address. The Options function is called in several places to populate drop-down list boxes with value and description pairs read from the STRINGS table in the database.

Notice that in the definition of the SelectRowCustomer function, the CGI variable, REMOTE_USER, is used to locate the customer record associated with the current user. This assumes that the Web server has been configured to require authentication for URLs matching the pattern: /cgi-bin/db2www/*. If this is not so, REMOTE_USER is blank and the selection fails.

One of the most important features of this template is the specification of the form's ACTION attribute, namely:

```
...
<FORM ACTION="$(SCRIPT_NAME)/$(DTW_MACRO_FILENAME)/SubmitQuoteRequest"
      METHOD="POST">
...
```

Figure 9. Specifying the Handling Form

This shows that the SubmitQuoteRequest HTML block of the same macro file is responsible for handling data submitted from this form. A description of this HTML block is found in the next section.

4.5 Starting a FlowMark Process Using Net.Data and the FlowMark REXX API

The HTML block SubmitQuoteRequest is responsible for handling data submitted from the InputQuoteRequest form. Its functions can be summarized as follows: query the database, validate the input data, update the database, start a new FlowMark process, and display an HTML report summarizing the results of the action.

In the text that follows, small fragments of the HTML block SubmitQuoteRequest in the Web macro file signco.d2w are quoted, where they help to illustrate the accompanying text. You might find it helpful to browse that file while reading this section.

4.5.1 Querying the Database

```
...
@SelectRowCustomer()
@SelectRowNextIDs()
...
```

Figure 10. Querying the Database

These two function calls populate variables with the prefixes Customer and NextIDs, respectively. These variables are used subsequently to help determine the updates that need to be made to the database.

4.5.2 Validating the Input Data

```
...
@ValidateOrder(CreateOrder)
...
```

Figure 11. Validating User Input

This function call performs some minimal validation on the Input variables received from the HTTP request. If the validation succeeds, the macro variable CreateOrder is set to ″TRUE″. The ValidateOrder function itself is done in REXX because REXX offers a rich environment for constructing such tests.

4.5.3 Updating the Database

This section of the HTML block demonstrates a sequence that is used to insert new rows into the database. The REXX function CalcShipToRef is called to determine if a new shipping address needs to be created for the shipment by comparing the customer's billing address with the shipping address from the form.

If a new address is required, the SQL function InsertAddress is called to create a new row in the ADDRESS table.

If an order needs to be created, the SQL functions InsertDesign, InsertShipment, and InsertOrder are called in sequence to create rows in each of the DESIGN, SHIPMENT, and ORDER tables.

Finally, if a FlowMark process is started successfully, the ORDER row is updated with the name of the started process. Otherwise, all updates to the database are rolled back with a call to the SQL function, Rollback.

```
%if("$(CreateOrder)" == "TRUE")
@CalcShipToRef(NextIDsADDRESS, ShipToRef, CreateAddress)
%endif

%if("$(CreateAddress)" == "TRUE")
@InsertAddress(ShipToRef, InputStreet, InputLocality,
               InputRegion, InputCountry, InputPostCode)
%endif

%if("$(CreateOrder)" == "TRUE")
@InsertDesign(NextIDsDESIGN, InputBaseHeight, InputBaseWidth,
              InputBaseThickness, InputBaseMaterial, InputBaseFinish,
              InputTextColor, InputTextStyle, InputTextSize,
              InputText)
@InsertShipment(NextIDsSHIPMENT, InputInstruction, ShipToRef)
@InsertOrder(NextIDsORDER, CustomerID, NextIDsDESIGN, NextIDsSHIPMENT)

...

%if("$(ProcessStarted)"=="TRUE")
@SaveProcessID(NextIDsORDER, FMPID)

...

%else
@Rollback()

...
```

Figure 12. Updating the Database

4.5.4 Starting a New FlowMark Process

The REXX function StartProcess is called with the identifier of the new order as an input parameter. StartProcess uses the REXX FlowMark API to establish a process control session with the nominated FlowMark server and then creates and starts a new instance of the process specified by the PROCESS_PREPARE macro variable. If the new process is started successfully, the macro variable FMPID contains its name and the macro variable ProcessStarted contains "TRUE".

```
...
@StartProcess(NextIDsORDER, FMPID, ProcessStarted)
...

%function(DTW_REXX) StartProcess(IN order, OUT InstanceName, ok) {

   ok = "FALSE";
   if RxFuncQuery('ExmLoadFuncs')\=0 then
   do
      rc = RxFuncAdd('ExmLoadFuncs', 'exmporex', 'ExmLoadFuncs')
      if (rc \= 0) then
      do
         say "Unable to load FlowMark rexx functions from exmporex"
         return
      end

      Call ExmLoadFuncs
      if (rc \= 0) then
      do
         say "Unable to call ExmLoadFuncs"
         return
      end
   end

   rc = ExmLogon('HANDLE', '$(FMUSER)', '$(FMPWD)', '$(FMDB)', '$(FMSERVER)')
   if (rc \= 0) then
   do
      say "Unable to logon to FlowMark, rc="rc
      return
   end

   OrderNumber=order;

   /* note: trailing commas are REXX continuation characters */

   rc= ExmStartUniqueProcess(HANDLE, ,
               '$(PROCESS_PREPARE)', ,
               'InstanceName',       ,
               'OrderNumber,string')

   if (rc \= 0) then
   do
       say "Unable to start FlowMark process, rc="rc
       return
   end

   ok = "TRUE";

   rc = ExmLogoff('HANDLE');
%}
```

Figure 13. Starting a New FlowMark Process

4.5.5 Displaying an HTML Report

The last few lines of this HTML block contain a report that differs according to the success of the attempt to start the FlowMark process.

4.6 Starting a FlowMark Process Using the IBM Internet Connection for Flowmark

The second example is displayed by fetching a URL of the form:

`http://web-server/cgi-bin/db2www/ConfirmOrder?InputOrderID=999`

where 999 is the identifier of an order for which the BILLABLE_AMOUNT column is non NULL.

In this example contained within the HTML block `ConfirmOrder`, we do not use the REXX FlowMark API to start a new FlowMark process. Instead, we display a form whose input is directed to an instance of the IBM Internet Connection for Flowmark. When you submit this form to the IBM Internet Connection for Flowmark, the gateway creates and starts an instance of the FlowMark process specified by the form variable `wf-api-proc-template` using form variables that begin with the prefix `wf-act-in-` as input container data. If the attempt to create and start a new process is successful, the Web page specified by the form variable `wf-cgi-html` is used as a template for the following HTML page. Otherwise, the IBM Internet Connection for Flowmark generates an HTML page containing an error message describing the failure condition.

```
<FORM action="/cgi-prot/exmp5cgi.exe" method="POST">
<input type=hidden name="wf-api-proc-template" value="$(PROCESS_CONFIRM)">
<input type=hidden name="wf-act-in-OrderNumber" value="$(InputOrderID)">
<input type=hidden name="wf-cgi-submit" value="2">
<input type=hidden name="wf-cgi-html"
       value="/sg242184/html/OrderConfirmed.html">
<input type=hidden name="wf-fmig-key" value="$(REMOTE_USER)">
<input type=submit value="Accept Quote and Confirm Order">
</FORM>
```

Figure 14. Starting a FlowMark Process with the Internet Connection for FlowMark

More information about building FlowMark applications with IBM Internet Connection for Flowmark, including more detailed examples, can be found in the IBM Internet Connection for Flowmark documentation that is available from: http://www.software.ibm.com/ad/flowmark/exmn0b21.htm and in Chapter 13, “Web-Enabled Activities” on page 103.

4.7 Discussion

Let us now compare the two approaches to starting FlowMark processes that you saw in the preceding examples.

4.7.1 Validation of User Input

In the first example, we use Net.Data′s ability to call arbitrary REXX functions to perform validation on the input data just before starting the FlowMark process using the REXX FlowMark API. If there are errors in the input data, the Web user can be informed of these errors and resubmit corrected data immediately if so desired.

In the second example, where the form variables are submitted directly to the IBM Internet Connection for Flowmark, there is no opportunity to perform server-side validation before starting the FlowMark process. However, if your validation requirements can be met with client-side validation, you might consider using Java Script, as we discuss in Chapter 13, “Web-Enabled Activities” on page 103. Remember, though, that if you do use client-side validation, you must be prepared to trust your clients!

When designing processes that are to be initiated from the Web, consider what the initial validation requirements are. If it is difficult or costly to correct invalid data once the process has been started, you might create a starter application that allows invalid data to be detected and corrected before using a FlowMark API to start the process, rather than simply passing the process start variables directly to the FlowMark Internet gateway without validation.

In the first example, it was worth performing validation. It does not make sense to proceed without valid numeric values for each of the dimensions since correcting such mistakes requires additional contact with the customer (a potentially time-consuming process). In the second example, validation is not as important. The input data normally consists only of hidden variables that have just been prepared by the server and can, therefore, be assumed to be valid when they are returned to the server on the next submission of the form.[2]

Recommendation

If you want to validate process start variables before starting a FlowMark process from the Web and you are using:

- IBM Internet Connection for Flowmark, then:
  - Use a cgi-bin program to prepare a validated set of hidden form variables that can be submitted to the IBM Internet Connection for Flowmark, or use Java Script to perform validation at the client.
- The start process API, then:
  - Include validation checks before your call to the FlowMark start process API.

4.7.2 Security Considerations

In the first example, the password of the FlowMark user ID that is used to logon to the FlowMark API is stored in the Net.Data macro file as a variable. To keep this password secret, you must ensure that the Net.Data macro file is stored on a secure file system. In the second example, which uses the IBM Internet Connection for Flowmark to start a FlowMark process, this is not an issue since

[2] Ignore, for the moment, the possibility of malicious use of the system.

the IBM Internet Connection for Flowmark is responsible for securely storing the passwords it uses to logon to the FlowMark server.

4.7.3 Authorization Checks

When designing a FlowMark application, particularly one accessible from the Internet, remember that the application might be abused by malicious users.

Here are some definitions: an *authentication* check verifies that a person is who they claim to be; an *authorization* check verifies that a person is entitled to some privilege. The security mechanisms of the Web server and the IBM Internet Connection for Flowmark take care of *authentication* checks but the FlowMark developer is still responsible for implementing appropriate authorization checks as part of process definition or implementation.

To illustrate: in the second example, it is possible for an authenticated user to confirm an order belonging to another user simply by creating an HTTP request with the appropriate value of `InputOrderID`. To detect this kind of unauthorized use of the system, you need to have an activity within the Manufacture Sign process that compares the FlowMark user ID of the process starter with the FlowMark user ID of the customer to whom the order belongs.[3]

Another way to prevent this kind of attack is to include authorization checks in a more general server-side validation procedure just before starting the FlowMark process. This, in turn, implies using the start process API rather than the IBM Internet Connection for Flowmark since there is no way to carry out server-side validation with the IBM Internet Connection for Flowmark.

Recommendation

If authorization of started processes is important to you and to start processes from the Web you are using:

- IBM Internet Connection for Flowmark, then:
  - Maintain a strict one-to-one mapping between Web user IDs and FlowMark user IDs.
  - Include an initial authorization check step in your process to detect abuses.
- The start process API, then:
  - Include authorization checks in your validation code before calling the start process API.

[3] Of course, this check is only foolproof if there is a strict one-to-one mapping between Web user IDs and FlowMark user IDs. If you have configured IBM Internet Connection for Flowmark so that many unrelated Web users share the same FlowMark user ID, the FlowMark user ID does not tell you the precise identity of the Web user that started the FlowMark process and so this kind of authorization check is not very effective.

4.7.4 Saving the FlowMark Process ID

In the first example, the SubmitQuoteRequest HTML block creates a record of the order in the database and stores the identifier of the FlowMark process with this record, which is started to handle it. This allows applications navigate from a known entity such as a customer or order identifier to the corresponding FlowMark process.

In the second example, the IBM Internet Connection for Flowmark creates the process and to create an association between the record of the order and the FlowMark process, it is necessary to insert an activity into the FlowMark process model that creates that association.

Recommendation

If you want to create an association between a FlowMark process and a database record and to start FlowMark processes you are using:

- IBM Internet Connection for Flowmark, then:
  - Include an initial activity in the process to create the association.
- The start process API, then:
  - Include some additional code to create the association using the process name that is returned from the call to the start process API.

4.7.5 Performance Considerations

If the first example, the Net.Data macro StartProcess is called once for each new process that is started. This macro logs on to FlowMark, starts a new process instance, and logs off. There are two problems with this. First, several incoming requests may cause contention for the use of the FlowMark user ID. Second, logging onto FlowMark is an expensive operation for both the client and server. These issues are less likely to arise if you are using the IBM Internet Connection for Flowmark to start processes since its design takes these issues into account.

A variation of the first example to add efficiency is to move the process-starting logic from the Net.Data macro to a dedicated, long-running daemon. The daemon is permanently connected to both FlowMark and the database manager. It periodically polls the database looking for new orders. When a new order is found, it uses the FlowMark start process API to start a new FlowMark process. This solution eliminates the overhead of establishing new FlowMark connections for each request. Such a solution also introduces a degree of fault tolerance, since the operation of the cgi-bin program is unaffected by outages of the FlowMark server.

4.8 Source Files and Installation

Source files:

- \examples\webstrtr\OrderConfirmed.html
- \examples\webstrtr\signco.d2w

Installation:

- Set up the example database as described in Appendix C, “Setting Up the Example Database” on page 139.

- Set up the Web server as described in Appendix D, “Setting Up the Web Server” on page 143.
- Copy OrderConfirmed.html into the directory mapped by /sg242184/html/ in your Web server configuration.
- Copy signco.d2w into the macro path specified by the db2www.ini file of your Net.Data configuration.

Chapter 5. Starting a FlowMark Process Using Microsoft Visual Basic

Starting a workflow process is usually a response to specific events. In our scenario, these events are:

- The customer submits an order.
- The customer approves the company's price quote.

Sometimes, the events are captured and handled by people who start the process manually. However, we can also start the process from within an application. This is an interesting option, because we can transparently integrate our existing applications with the processes, and we can automate the responses to the events we want to handle.

Microsoft Visual Basic has became one of the most used application development tools of the Windows platform. Some of its more important features are:

- Simplicity
- Integrable with the Microsoft Office applications
- Rapid application development or prototyping
- ActiveX control container
- ActiveX control generator (version 5)

5.1 Purpose of This Example

In the previous chapter, we have shown how to start a process from the Web. In this chapter, we want to show how to start a FlowMark process from a Microsoft Visual Basic application.

As a secondary purpose, this example shows the use of a third party ActiveX component within the Flowmark activity program and the access to an ODBC compliant database.

5.2 Where This Example is Used

This example is used to create and start a new process instance of the Manufacture Sign process. See Section 3.2, "Manufacture Sign Process" on page 10.

It is used to collect information about the order. The sales representative interacts with the application while they talk with the customer on the phone. The representative receives the details of the order from the customer, and using the application, generates a new order, stores it in the database and calculates the price. If the customer approves the quote, the Manufacture Sign process is started from within the application.

If the customer rejects the quote, no FlowMark process is started.

5.3 What Techniques This Example Demonstrates

This example demonstrates the integration between:

- IBM FlowMark
- Microsoft Visual Basic version 5
- IBM Database 2
- Third party ActiveX components (RBExcel and RBRegistry)

See Chapter 14, “The Redbook Utility ActiveX Component” on page 121 for a discussion on RBExcel and RBRegistry.

5.4 The Microsoft Visual Basic Development Environment

To use the components of this application within a Visual Basic program, they must be referenced in the development environment. To do this, select Project from the menu, and then select References, which brings up a dialog with a list box. Check the following items and press OK:

- FlowMark Redbook Utility Component
- Microsoft DAO 3.5 Object Library

Then, select Project from the menu, and select Components, which brings up a dialog with a list box. Check the following items and press OK:

- Microsoft Windows Common Control 5.0 (tabStrip control)

5.5 Discussion

Microsoft Visual Basic can be used as a development environment for applications that:

- Execute process control functions, such as:
  - Creating a new instance from a process template.
  - Starting and ending a process instance.
  - Suspending and resuming a process instance.
- Are assigned to a program activity and automatically invoked by Flowmark when the activity is started.

A Microsoft Visual Basic application can use the FlowMark APIs for Visual Basic, which provides support for process control and container access. An alternative method is to use the FlowMark ActiveX Toolkit, which provides the following four controls:

- Server control
- ProcessList control
- Work List control
- Container control

Read Chapter 7, “Integrating with Microsoft Excel” on page 49 and Chapter 8, “Printing and Sending a Reply to a Customer” on page 57 for the use of container controls. Read Chapter 10, “Custom FlowMark Run-time Client - Powerbuilder Version” on page 69 and Chapter 11, “Custom FlowMark Run-time Client - Visual Basic Version” on page 77 for the use of server, work list and

container controls. You can also find information in the documentation that comes with the FlowMark ActiveX toolkit.

This chapter, however, will focus on the FlowMark Visual Basic API functions. In this sample, we show the API to create and start a new instance of the process.

The Microsoft Visual Basic application allows:

- Creating new orders (storing all of the details in a database and using the RBRegistry component).
- Calculating the quote of the order (using the RBExcel component).
- Creating the process to handle the manufacturing of the sign if the customer approves the quote.
- Querying the order status and its details.

In the following section, we describe the highlights of the application.

5.5.1 Interaction with the Database

The greater part of the application is dedicated to collecting details for the orders, managing the GUI, and interacting with the database.

Visual Basic provides data access capability based on the Microsoft Jet database engine. It handles the mechanics of storing, retrieving, and updating data, and provides you with the powerful DAO object-oriented programming interface that gives you total control of the database.

DAO is a collection of object classes that model the structure of a relational database system. They provide properties and methods that allow you to accomplish all of the operations necessary to manage such a system, including facilities for creating databases, defining tables, fields, and indexes, and querying the database. The underlying database engine translates these operations on data access objects into physical operations on the database files.

Database programming in Visual Basic consists of creating data access objects (database, fields, recordset, index objects) and the use of the properties and methods of these objects to perform operations on the database.

If the target database is accessed through ODBC, Microsoft Visual Basic allows you to specify the bypassing of the Jet engine. This is called ODBC direct access. However, you can still use the DAO model to interact with the database. We will use this technique (ODBC direct + DAO) to interact with the DB2 database.

We need to create an ODBC Workspace. We can pass the user ID and password. This creates a session, and all operations performed during this session are subject to permissions determined by the logon user name and password. You can open more than one workspace at the same time. In our example, we have used just one for all of the database access. The constant dbUseODBC sets the workspace′s type to ODBC direct.

For more information about the Registry Component, please see section 14.4, “The RBRegistry Class” on page 122.

```
Set ws = DBEngine.CreateWorkspace("ODBCWksp", _
                            oRBRegistry.ODBCUserName, _
                            oRBRegistry.ODBCUserPassword, _
                            dbUseODBC)
ws.DefaultCursorDriver = dbUseClientBatchCursor
```

Figure 15. Create an ODBC Workspace

After that, we have to open a connection with the database. The connection object represents a connection to an ODBC database (ODBCDirect workspace only). When you open a connection, you need to specify the name of the registered datasource, the updatability of the data accessed through this connection, and, optionally, a valid connect string that supplies parameters to the ODBC driver manager. Different ODBC data sources may require different parameters in the connect string.

```
'Open Connection
strConnect = "ODBC;UID=" + oRBRegistry.ODBCUserName + _
             ";PWD=" + oRBRegistry.ODBCUserPassword + _
             ";DSN=" + oRBRegistry.ODBCDataSource

Set cnn = ws.OpenConnection(oRBRegistry.ODBCDataSource, _
                        dbDriverComplete, _
                        False, _
                        strConnect)
```

Figure 16. Open a Connection

Now, we show the retrieval of data. The key object class is the RecordSet. It is a group of records representing a database table or the result of a query. There are different types of RecordSet. Depending on the operations you need to do, you must use the appropriate recordset type. Specify an SQL statement that defines the resulting set of records to be retrieved.

```
Set rs = cnn.OpenRecordset("Select ID, TEXT from " + _
                           oRBRegistry.StringTable + _
                           " where Class=" + Str(CLASS_TSTYLE))
```

Figure 17. Open a RecordSet

The recordset object provides a set of methods to retrieve and navigate through the records. The code in Figure 18 on page 33 shows a loop through all of the records retrieved in the recordset. Fields is a collection of field objects, and you can access a specific field by referring to its name.

```
i = 0
'For each record, retrieve the TEXT and ID fields ...
'...  and add them to the combo and array, respectively
With rs
  Do While Not .EOF
    MainForm.cmbTextStyle.AddItem (.Fields("TEXT").Value)
    iTStyleArray(i) = .Fields("ID")
    i = i + 1
   .MoveNext
  Loop
End With
rs.Close
```

Figure 18. Using the Retrieved Data

To insert a record, you need to specify the target table, the type of recordset, and the type of lockedits. When the recordset is opened, execute the AddNew method, which adds a record to the table and makes it the current record. Assign the values to each field and execute the Update method, which updates the table.

```
...
' Create a RecordSet, with the new record
Set rs = cnn.OpenRecordset(oRBRegistry.OrderTable, __
                      dbOpenDynaset, 0, _
                      dbPessimistic)
' Add the record
With rs
  .AddNew
  !ID = IDOrder
  !Customer_Ref = CustomerRef
  !Design_Ref = DesignRef
  !Shipment_Ref = ShipmentREf
  !BILLABLE_AMOUNT = BillableAmount
  !Status = OrderStatus
  !FMPID = FMPID
  .Update dbUpdateCurrentRecord, True
  .Bookmark = .LastModified
End With
```

Figure 19. Insert a Record

Updating a record is similar to adding a new one. Instead of using the AddNew method, you have to use the Edit method. This method copies the current record from an updatable Recordset object to the copy buffer for editing. Any changes made to the current record are copied to the copy buffer. You need to use the Update method to apply or save your changes.

If you edit a record, change some fields, and move to another record without first using the Update method, your changes are lost and no warning is given.

```
...
'Open the record set with the record to be updated
Set rs = cnn.OpenRecordset("Select * from " + _
                           oRBRegistry.OrderTable + _
                           " WHERE ID = " + Str(IDOrder), _
                           dbOpenDynaset, 0, dbOptimistic)
'Edit the record
rs.Edit
rs !BILLABLE_AMOUNT = Price
'Update the record
rs.Update
rs.Close
```

Figure 20. Update an Existing Record

The Workspace object (really, the Workspace class) provides a set of methods to manage transaction processing during a session. Using these methods, you can specify the beginning of a new transaction, the end of the current transaction, the commitment of all changes, or the end of the current transaction, restoring the database to the state it was in when the current transaction began.

These methods are BeginTrans, CommitTrans and Rollback, respectively.

Typically, you use transactions to maintain the integrity of your data when you must update records in two or more tables and ensure changes are completed in all tables or none at all.

Remember that transactions are global to the Workspace beyond the scope of a connection. If you perform operations on more than one connection within a Workspace transaction, using the CommitTrans or Rollback method affects all operations on all connections within that workspace.

You can nest transactions and, if you need simultaneous independent transactions, you can create additional workspace objects to contain the concurrent transactions.

```
...
ws.BeginTrans
...Insert Customer record
...Insert Shipment record
...Insert Design record
...Insert Order record
If EveryThing is OK then
  ws.CommitTrans
Else
  ws.Rollback
End if
```

Figure 21. Handling Transactions

When you create a workspace, connection, or recordset object, it is added to a collection. When the object is no longer necessary, it can be closed. These objects provide the Close methods, which delete the objects from the collections.

```
Public Sub CloseConnectionToDB()
  cnn.Close
  ws.Close
End Sub
```

Figure 22. Closing Connections and Workspaces

5.5.2 Using the RBExcel Component

The RBExcel component allows you to calculate the price of a new order depending on the sign design details. This component uses OLE automation to interact with Excel.

This component provides these functions:

- AddOrder: inserts a new order in the spread sheet and calculates the price.
- GetPrice: retrieves the calculated price.
- UpdateOrderTable: updates the price in the order table.
- Quit: saves the spread sheet and closes Excel.

```
Private Function CalculateOrderPrice(OrderPrice As Currency) _
                                 As Long
  Dim oRBExcel As RBExcel
  Dim textLength As Double

  Set oRBExcel = New RBExcel
  textLength  = CInt(Len(txtText.Text) * 2 / 3 * _
                CInt(txtTextHeight.Text))
  Call oRBExcel.AddOrder(IDOrder, _
                Now, _
                txtBaseHeight.Text, _
                txtBaseWidth.Text, _
                txtBaseThickness.Text, _
                sBMaterialArray(cmbBaseMaterial.ListIndex), _
                sBFinishArray(cmbBaseFinish.ListIndex), _
                txtTextHeight.Text, _
                txtLength, _
                sTColourArray(cmbTextColour.ListIndex) _
                )

  OrderPrice = oRBExcel.GetPrice(IDOrder)
  CalculateOrderPrice = 0&

  ' Quit Excel, saving the file
  oRBExcel.Quit

End Function
```

Figure 23. Using the Component

5.5.3 Creating and Starting the New Instance of the Process

To create and start a new instance of the Manufacture Sign process, we have used the FlowMark Visual Basic API.

Before you can use any process or container function, you have to call the ExmStartApi. The last call in your program must be ExmFinishApi. We strongly recommend doing this to avoid memory leaks after the program is terminated.

```
...
lRC = ExmStartApi()
...Logon
...Create and start process instance
...Logoff
lRC = ExmFinishApi()
```

Figure 24. ExmStartApi and ExmFinishApi Calls

To begin a process control session, we used the Logon API. We have to fill the SessionData structure with the all of the information about the user ID, password, database, and server name. After a successful execution we get a handle, which is required by all of the other process control functions.

```
Dim FM_Handle as Long
Dim SessionData as ExmApiBegin
...
' Set variables to logon
With SessionData
  .Password = Password
  .UserID = UserID
  .Database = oRBRegistry.FMDatabase
  .Server = oRBRegistry.FMServer
End With

' Logon to FlowMark Server
lRC = ExmLogon(SessionData, FM_Handle)
```

Figure 25. Logon

To create and start a new process instance, we used the ExmStartProcess2 API. This API provides better performance than the ExmStartProcess API, and all values for data items defined by the data structure of the input container are specified as character strings.

```
...
Dim TemplateName As String
Dim InstanceName As Variant
Dim DataSize As Long
Dim MemberData(1) As ExmApiMemberData
Dim Size As Integer

...
' Try to create the Process Instance
TemplateName = "Manufacture Sign"
InstanceName = "Order-" + Str(OrderNumber)
Size = 1
MemberData(0).Name = "OrderNumber"
MemberData(0).DataArea = Str(OrderNumber)

lRC = ExmStartProcess2(FM_Handle, TemplateName, InstanceName, _
                       Size, MemberData())
```

Figure 26. Start Process

Here is the code to end a process control session.

```
...
' Logoff to FlowMark Server
lRC = ExmLogoff(FM_Handle)
```

Figure 27. Logoff

5.6 FlowMark Definitions

This example uses the following FlowMark definitions:

Processes

- Manufacture Sign

Programs

- Manufacture Sign

Data Structures

- OrderNumber

Because this program creates and starts a new process instance, it does not appear in the process specification. However, it needs to know the name of the Template that is used to create the new instance of the process. Also, it needs to know the name of the members of the input data structure of the process. Here, the data structure used is Order Number.

5.7 Source Files and Installation

Source files:

- \examples\vbstrtr\GenerateOrder.vbp
- \examples\vbstrtr\GenerateOrder.vbw
- \examples\vbstrtr\MainForm.frm
- \examples\vbstrtr\LogonForm.frm
- \examples\vbstrtr\Exmbabsu.frm
- \examples\vbstrtr\General.bas

Installation:

- Run `\examples\vbstrtr\install\setup.exe`.

Note

The redbook utility component is required to run this example program and should be installed first. Please refer to Chapter 14, “The Redbook Utility ActiveX Component” on page 121 for information about this component and for installation instructions.

Chapter 6. Using a Relational Database in a FlowMark Process

A FlowMark process often needs to use data stored in an external database, such as IBM Database 2. One method is to include a database API in each activity program that needs access to the database. An alternate technique is to code dedicated database access activities within the FlowMark process model that will satisfy the database access requirements of neighboring activities. This chapter illustrates a building block that can be used to carry out the second technique.

6.1 Where This Example Is Used

The program discussed here has a FlowMark program registration called: ExecSQL. ExecSQL is used in both the Prepare Quote and Manufacture Sign processes to execute various kinds of SQL statements. In the Prepare Quote process, ExecSQL is used to select a customer record from the database and also to mark the order to show that the quote has been prepared. In the Manufacture Sign process, ExecSQL is used to select a design record from the database and also to mark the order to show that the order has been confirmed.

Also, another example process, Insert Customer, has been created to show the use of ExecSQL to perform a more complex sequence of SQL as a sequence of FlowMark activities.

6.2 What Techniques This Example Demonstrates

This example illustrates the design and use of a reusable building block for executing single SQL statements as activities of a FlowMark process. It also illustrates the use of:

- The REXX FlowMark Container API to get and set container variables
- The REXX FlowMark Container APIs to learn about the structure of the FlowMark input container and output container
- The DB2 REXX SQL APIs

The process, Insert Customer, shows how a single, general-purpose building block can be reused in different ways to perform a complex task. Also, we show the principle of creating compensating activities to undo the effects of partially complete workflow transactions that have failed. We discuss both the benefits and costs of building FlowMark processes from general purpose building blocks.

6.3 A Common Task: Executing a Single SQL Statement

Consider the Select Customer activity in the Prepare Quote process. Our task here is to make the contents of the customer record available to the FlowMark process so that the workflow engine can use the e-mail attribute decide whether to send an e-mail or mail item.

Figure 28 on page 40 shows an SQL statement that will extract this information from the database.

```
SELECT EMAIL
FROM
   SG242184.ORDER    ORD,
   SG242184.CUSTOMER CST
WHERE
   AND CST.ID=ORD.CUSTOMER_REF
   AND ORD.ID=?
```

Figure 28. An SQL Statement to Retrieve the E-mail Address

We could realize the requirement to execute this SQL by writing a single special purpose program in any language that allows us to access both an SQL API and the FlowMark API. This program would: read the SQL variables from the input container; connect to the database; execute the SQL; write the result set into the output container; disconnect from the database; exit with a return code indicating the success of the task. This is not a complex task, in principle, but if there are several such activities to do, the burden of maintaining such special purpose programs can start to become significant.

The solution to this dilemma is to write a single program that is sufficiently flexible to deal with a variety of different applications; in other words, to create a reusable building block.

6.4 Designing the Input Container

The inputs to our building block are:

- The details required to establish a connection to the database
- The SQL statement to be executed
- Values for the SQL statement's variables

The connection details are described by a single FlowMark data structure, SQL Connection, in Figure 29. Usually, only the DATABASE member should be set as it is more secure to delegate resolution of the user ID and password to the User Profile Management subsystem on the machine that executes the activity.

```
STRUCTURE 'SQL Connection'
   'DATABASE':     STRING
   'USERID':       STRING
   'PASSWORD':     STRING
END 'SQL Connection'
```

Figure 29. FlowMark Data Structure: SQL Connection

Also, an SQL statement is easily represented by a single STRING member. To eliminate the need for our program to perform parameter substitution, we will use the same convention that the underlying SQL API (here, DB2 REXX) uses to represent parameters; that is, we will use question marks (?) within the text of the SQL statement as parameter markers.

An arbitrary SQL statement may contain an arbitrary number of parameters and these parameters may be of arbitrary types. This means that no single FlowMark data structure can adequately describe the parameter sets used by all possible SQL statements. The solution to this dilemma is to use the FlowMark

APIs to discover at runtime the precise number and types of variables to be substituted into the SQL statement.

There are several ways to represent arbitrary collections of variables. One way is to create, for each different collection of variables, a single data structure type that has one member of the correct type for each variable in the collection. A problem with this approach is that it can lead to an explosion in the number of FlowMark data structure definitions that need to be created and maintained. An alternative approach, used here, solves this problem by using arrays of a structure type we call SQL Variable.

```
STRUCTURE 'SQL Variable'
   'V_STRING' : STRING
   'V_LONG'   : LONG
   'V_FLOAT'  : FLOAT
END 'SQL Variable'
```

Figure 30. FlowMark Data Structure: SQL Variable

This type has one member for each primitive FlowMark container item type: STRING, LONG and FLOAT. The process modeler chooses the type of a particular variable by configuring a data connection only to the SQL Variable member that matches the variable's type.

The constraints we place on the input container of any activity that is done by our program are that it must contain:

- A mandatory member, named CONNECTION, of type SQL Connection
- A mandatory member, named SQL, of type STRING
- An optional member, named VAR, of type SQL Variable[...]

  The size of the VAR[] array must match exactly the number of parameter markers in the SQL statement.

  By our convention, FlowMark data structures conforming to these constraints are named as follows:

  – SQL Input 0
  – SQL Input 1
  – SQL Input 2
  – ...
  – SQL Input *N*

  where *N* is the size of the VAR array, if present, 0 if not.

6.5 Designing the Output Container

The outputs of our building block are:

- The SQLSTATE after execution of the statement (SQLSTATE:STRING)
- The number of rows fetched or otherwise affected by the statement (NROWS:LONG)
- If the SQL statement is a SELECT statement, the result set (ROW[max-rows]:some-type)

The element type of the ROW array is chosen so that its members match the corresponding columns of the SELECT statement. The size of the ROW array is chosen to match maximum number of rows that the SELECT statement is expected to return.

By convention, the output container is named `SQL Output structure N` where `structure` is the type of an element of the ROW array and N is the size of the ROW array. The structure called `SQL Output 0` can be used with SQL statements that do not generate a result sets.

6.6 Implementing the Program

`ExecSQL` is typical of FlowMark application programs written in REXX. It has 5 main sections: mainline, initialization, execution, termination and error handling.

- Mainline

  The mainline section does some preliminary initialization then calls the initialization, execution and termination sections in sequence.

- Initialization

  The initialization section initializes the FlowMark and SQL APIs and reads the input parameters from the FlowMark input container to initialize some control variables and data structures used in other sections.

  The procedure `GetSQLVars` copies the contents of the VAR array into an SQLDA-like data structure that can be used with the SQL API. The FlowMark data type of the initialized member of each VAR element is used to select the matching SQL datatype for the corresponding SQL variable. This procedure illustrates how to determine the size of an array of structures, in this case the VAR structure, using the information in the METAIN. array.

  The procedure `BuildColumnMap` illustrates how to determine the member names of an element of the ROW array by using the METAOUT. array.

- Execution

  The execution section connects to the DB2 database and executes the SQL statement. If the SQL statement is a SELECT statement, the procedure `WriteResultRow` is used to append each row to the OUT.data structure that is written into the FlowMark output container during termination processing.

- Termination

  The termination section is called during normal termination processing and also for an abnormal exit using a call to the Abort procedure. Its responsibility is to write the contents of the OUT. variable into the output container and to disconnect from the DB2 database.

- Error handling and message logging

  The error handling section contains a few functions that are responsible for logging messages (to STDERR:) and handling fatal errors. One important design point in the design of the `Abort` function is that it tries to execute the normal termination sequence. This is important since the normal termination sequence contains the code that updates the FlowMark output container. If this code was not called, then important information such as the SQLSTATE at the point of failure would not get written into the output container.

The variable EXITCODE is used to detect if Abort has been called recursively. If it has, all further abort processing is abandoned and the command exits immediately with the exit code of the first call to Abort.

6.7 Using the Program

To use ExecSQL to execute an SQL statement in a FlowMark process model, you need to perform several steps. To show those steps, we will use the example of configuring the Select Customer activity in the Prepare Quote process.

- Select the row data type for the output container

  We want to obtain the e-mail address of the customer. Rather than simply extract the e-mail address, we will extract the entire customer record since we do not want to create a special structure that contains just an e-mail address and we already have the Customer data structure available.

- Build the data type for the output container

  We expect to receive only one customer record in response to our query. So, we build a data type called: SQL Output Customer 1 by cloning the SQL Output structure 1 data type and changing the type of the ROW array to be Customer

```
STRUCTURE 'SQL Output Customer 1'
   'SQLSTATE':    STRING;
   'NROWS': LONG;
   'ROW':   'Customer'(1);
END 'SQL Output Customer 1'
```

Figure 31. FlowMark Data Structure - SQL Output Customer 1

  If, instead of being a SELECT statement, the statement of interest had been an INSERT, UPDATE or DELETE statement we could have used the SQL Output 0 structure.

- Build the SQL statement

  The SQL statement must generate a result set whose columns map exactly into the members of the FlowMark output container.

  Parameter markers should be used in the SQL statement as place holders for parameters that are intended to be read from the FlowMark input container. Here, our parameter is the identifier of the order for which the process has been created, so our statement will contain exactly one parameter marker.

```
SELECT
   CST.ID, NAME, EMAIL, PHONE, WEBUSER,
   ADR.ID, STREET, LOCALITY, REGION, COUNTRY, POSTCODE
FROM
   SG242184.ORDER ORD,
   SG242184.CUSTOMER CST,
   SG242184.ADDRESS ADR
WHERE
       CST.BILLTO_REF=ADR.ID
   AND CST.ID=ORD.CUSTOMER_REF
   AND ORD.ID=?
```

Figure 32. SELECT Statement to Fetch an Order's Customer

- Build the data type for the input container

  Our SQL statement contains exactly one parameter. Therefore, we need an input data type that contains exactly one variable, namely: SQL Input 1

```
STRUCTURE 'SQL Input 1'
   'SQL':    STRING;
   'VAR':    'SQL Variable'(1);
   'CONNECTION':  'SQL Connection';
END 'SQL Input 1'
```

Figure 33. FlowMark Data Structure - SQL Input 1

- Configure the FlowMark activity

  Now, we are ready to configure the FlowMark activity. First, we set the Program to ExecSQL. Next, we set the input container to SQL Input 1 and the output container to SQL Output Customer 1. Finally, we set the start and exit conditions to automatic with an expression for the exit condition that ensures that the SQL executes without errors and generates exactly one row, namely: _RC=0 AND NROWS=1

- Configure the activity's input connections and defaults

  Now, we connect the OrderNumber member of the block's input container to VAR[0].V_LONG member of the Select Customer activity's input container. We choose to connect to V_LONG because OrderNumber is a LONG member. Next, we type in the SQL we built as a default for the SQL member. Finally, we configure the default value of the DATABASE member of the CONNECTION data structure with the name of the database, here: SIGNDB.

  If our SQL statement had multiple parameters, we would need to note the order of the parameter markers in our SQL statement when making connections to the members of the input container to ensure that the FlowMark data items are bound to the correct SQL variables.

For other examples of using this program, refer to the FlowMark process models that accompany this book.

6.8 A More Complex Example

The Insert Customer process is a 5-activity process that uses ExecSQL multiple times to add a new customer to the database. ExecSQL is used twice to execute SELECT statements, twice to execute INSERT statements and once to execute a DELETE statement. The first SELECT activity is used to acquire the identifiers to be used for the new CUSTOMER and ADDRESS rows. The first INSERT activity is used to insert a row in the ADDRESS table, the second INSERT activity is used to insert a row in the CUSTOMER table. Both the INSERT activities use identifiers retrieved by the first SELECT activity. The DELETE activity is used to delete the new ADDRESS row from the table if the CUSTOMER row could not be inserted.[4]

Notice that we need to explicitly issue the DELETE statement because all the statements in this example execute in different database manager transactions and so we cannot rely on the database manager to undo an incomplete transaction. The DELETE activity is an example of a compensation activity, the purpose of which is to undo the effects of a workflow transaction that has failed.

The top-level block, Retry SQL, is used to retry the entire workflow transaction until it is completely successful.

It should be noted that this example achieves a rather simple end (inserting two new rows in the database) with complex means. This suggests a flaw in our approach. And, in fact, there is one — the problem is that the building block we are using (that is: our program ExecSQL) isn't quite flexible enough to deal with our requirement to execute a set of related SQL statements within the scope of a single SQL transaction. To compensate for this lack of flexibility we have moved what is essentially programming logic out into the FlowMark process model.

Recommendation

Be wary of trying to use the FlowMark Buildtime environment as a programming tool. If your process model is becoming cluttered with activities that address low-level technical concerns rather than high-level business issues, reconsider the division of responsibilities between the process model and your application programs.

6.9 Limitations of This Example

The ExecSQL program that is the focus of this chapter is a genuinely reusable building block that could be used in other scenarios unrelated to the ones described by this book. However, if you are considering doing this it would be worth taking into account its limitations.

[4] One reason that the CUSTOMER row may fail to insert is that a race condition may result in two instances of the Insert Customer process attempting to insert rows into the CUSTOMER table with the same identifier resulting in one process encountering a violation of the CUSTOMER table's uniqueness constraint.

6.9.1 Performance

Every SQL activity carried out by ExecSQL establishes its own connection to the database. This is potentially a significant performance problem since SQL connections are expensive. One solution to this problem would be to restructure the implementation of the ExecSQL program so that it sends the contents of its input container to a server process that maintains a permanent or semi-permanent connection to the database, executes the SQL and then sends the results back. The FlowMark Service Broker architecture provides a framework for implementing services of this kind. More information about the FlowMark Service Broker can be found in *IBM FlowMark: Application Integration Guide, SH12-6267*.

6.9.2 SQL Complexity

ExecSQL is limited to executing single SQL statements. As we saw in 6.8, “A More Complex Example” on page 45 the FlowMark process model can become quite complex if several SQL statements need to be executed as an atomic transaction not the least because, by executing each statement in its own transaction, one loses the support of the rollback capabilities of the database manager.

If you find that your database access requirements cannot be met with one or two simple SQL statements, then you should consider using a scripting language that has access to an SQL API to carry out your data access rather than attempting to reuse the building block described here.

6.9.3 Support for Other Data Sources

This example was done using the REXX DB2 Client API. If your data source is not a DB2 database you may wish to consider using a middleware technology such as IBM Data Joiner or, alternatively, rewrite the example to take advantage of an ODBC API.

6.10 Discussion

This chapter has illustrated the design, implementation and use of a reusable building block for executing single SQL statements with the activities of a FlowMark process.

The key advantage to using a FlowMark activity to perform database access is that it can significantly simplify the development of neighboring activities, since the programmers of those activities do not need to take account of database access issues. The disadvantage of this approach is that the level of control over the interaction with the database may be reduced because, among other reasons, database operations tend to happen out-of-phase with user interactions. Also, there is the significant issue of maintaining data consistency that can arise if data is copied into a FlowMark process flow and hence outside the scope of a single database transaction, thereby introducing an inconsistency between the database′s view of the data and the view of the FlowMark process.

An alternative approach, described in other chapters of this book, is to encapsulate knowledge of the external database within the application programs that carry out the program activities of the FlowMark process. The alternative approach has the advantage of permitting the tightest possible control over the interaction with the database, since the programmer of each application program

can use the features of a rich programming environment to deal with most functional and non-functional requirements. The disadvantage of this approach is that it may require every application used by a FlowMark process to have knowledge of the external database or alternatively, may encourage the introduction of contrivances whereby activities that can use the database are forced to take account of the data access requirements of neighboring activities that cannot. Both these tendencies work against the development of well-encapsulated, reusable building blocks, thereby making application development more costly over the long term.

As in most things, there is no single correct approach. The most practical approach in a given situation is likely to be a blend of the two approaches.

6.11 FlowMark Definitions

This example uses the following FlowMark definitions:

Processes

- Prepare Quote
- Manufacture Sign
- Insert Customer

Programs

- ExecSQL

Data Structures

- SQL Input 0, SQL Input 1, SQL Input...
- SQL Output 0, SQL Output Customer 1, SQL Output...
- SQL Connection
- SQL Variable

6.12 Source Files and Installation

Source files:

- \examples\execsql\xfmsql.cmd
- \examples\execsql\insttest.cmd

Installation:

- Edit the FlowMark program registration of the ExecSQL program

  On the **Network** page enter the IP address of the OS/2 server on which the REXX program, xfmsql.cmd, will execute. Ensure that the **Run program unattended** check box is checked.

- Translate the affected FlowMark processes: Prepare Quote, Manufacture Sign and Insert Customer .
- Copy xfmsql.cmd into a directory of the PATH of OS/2 server identified in the first step
- Configure User Profile Management on the OS/2 server so that the OS/2 command:

  ```
  DB2 CONNECT TO SIGNDB
  ```

succeeds without generating a logon prompt.

- Test the installation

  Run the command \examples\execsql\insttest.cmd. This command will create an instance of the Insert Customer process that should run to completion with the effect of inserting new rows into both the CUSTOMER and ADDRESS tables.

Chapter 7. Integrating with Microsoft Excel

In most business processes, no single individual or computer program makes all of the decisions required to complete the process. The interaction of many people and several systems is usually necessary. Generally, the logic and business rules used to make these decisions are not stored in a central location, but distributed among the participants in the flow of work who have access to tools specifically designed for their task.

One such tool that is commonly used for business rule calculations is the spreadsheet. Complex formulas and scripts can be developed and used by the business experts themselves. Integration with this type of office software is important to many customers.

7.1 Where This Example is Used

The examples discussed in this chapter are part of a sample FlowMark scenario described in Chapter 2, “Scenario Description” on page 7. The FlowMark implementation of this scenario is discussed in Chapter 3, “FlowMark Model Description” on page 9. The FlowMark process model details of the sample activity programs presented here are described in:

- Section 3.1.1, “Calculate Price Activity” on page 10.

This example program is implemented as the Calculate Price activity in the Prepare Quote process. When the activity is started, the user is presented with a window displaying the relevant order and design information. At this time, the user presses the Update push button to add the information to a Microsoft Excel spreadsheet, calculate the customer price, and update the order table with the calculated price. The spreadsheet program and the worksheet file are automatically loaded for the user. After the row has been entered and the price automatically calculated, the price is returned to the activity program display. The user can then complete the activity program, which saves the worksheet, and exit the spreadsheet application.

7.2 What Techniques This Example Demonstrates

This example shows the integration of a FlowMark run-time activity with Microsoft Excel. As part of the integration, the use of the FlowMark ActiveX container control is demonstrated. An external database is used to retrieve additional information necessary for the spreadsheet application and OLE automation is used for the communication between the FlowMark activity and Excel.

Another purpose for this example is to show the use of Visual Basic as a development tool for FlowMark activity programs. All parts of this example program were developed using Visual Basic version 5.0, including the redbook utility component. The help files and online books supplied with the Visual Basic product describe everything necessary for developing ActiveX controls and using OLE automation to control other Windows applications.

A final purpose of this example is to show the use of a third party ActiveX component within the FlowMark activity program. Any control or component can

be used to develop the application; we used the redbook utility component. This code component is designed to provide utility functions for our programs that can be easily reused. In this example, methods for accessing the Windows registry and controlling Excel are used from the component. This component is described in detail in Chapter 14, “The Redbook Utility ActiveX Component” on page 121.

7.3 Automation

Automation (formerly known as OLE Automation) is a feature of the Component Object Model (COM), an industry-standard technology that applications use to expose their objects, methods, and properties to development tools, macro languages, and other applications. For example, a spreadsheet application might expose a worksheet, chart, cell, or range of cells, each as a different type of object. A word processor might expose objects such as an application, document, paragraph, bookmark, or sentence.

When an application supports Automation, the objects the application exposes can be accessed through Visual Basic. You can use Visual Basic to manipulate the objects by invoking methods or by getting and setting properties of the objects.

7.4 Setting Up the Visual Basic Development Environment

As previously mentioned, this application uses several ActiveX components. To use these components within a Visual Basic program, they must be referenced in the development environment. To do this, select **Project** from the menu, and then select **References...** which brings up a dialog with a listbox. Check the following items and press OK:

- FlowMark Container OLE Control module
- FlowMark Redbook Utility Component
- Microsoft DAO 3.5 Object Library
- OLE Automation

Since the referenced FlowMark component contains a custom control, you see an icon that represents the control added to the toolbox. To use this control, click on the tool icon, and place it anywhere on your form. It is an invisible control, so you can place it anywhere. We left our control with the default name, `ContainerCtrl1`.

7.5 Accessing FlowMark Container Data

Generally, almost all FlowMark activity programs are going to use the data supplied by the input container. In our example, the input container contains a member named `Order_ID`, that, not surprisingly, contains the order number generated by the process starter program. This order number is our key to all of the other data about the order. This is a good rule of thumb for FlowMark process and application design — use the data container only for key data and information necessary for process control. Our application defines a function to get the data from FlowMark, as shown in Figure 34 on page 51.

```
Private Function GetFMContainerData() As Long
    Dim inContainer As Container
    Dim sessionID As String
    Dim lngOrder As Long
    Dim returnCode As Long
    ...
```

Figure 34. Objects Required to Access Container Data

The local variables used in this function are also displayed in Figure 34. The object types Container and ContainerElementArray are defined by the FlowMark ActiveX control ContainerCtrl. Declaring these objects only creates a variable to reference an object of the specified type. We have to specify or create the objects, and set the references to refer to these objects. This is shown in Figure 35.

```
    ...
    Set inContainer = ContainerCtrl1.Container
    sessionID = inContainer.GetExecutionSessionID
    returnCode = inContainer.GetInContainer(sessionID)
    If returnCode <> 0 Then
        GetFMContainerData = returnCode
        Exit Function
    End If
    ...
```

Figure 35. Retrieving Container Data into a Local Object

First, we set our container variable to refer to the container object that is part of our control. ContainerCtrl1 is the name of the container control that was placed on the form during development. At this point, the container does not contain any information, it is just an empty object. Next, we get the execution session ID, which is a handle required by some of the methods we will invoke. Finally, we get the input container data from FlowMark into our container object. Our error routines would need to be enhanced for production applications.

Now that we have a container object with all of the input container data, we can access any of the members we need. For our example, we need to retrieve the order number for this particular process instance. This is shown in Figure 36.

```
    ...
    returnCode = inContainer.GetValueLng("Order_ID", lngOrder, False, 0& )
    If returnCode <> 0 Then
        GetFMContainerData = returnCode
        Exit Function
    End If
    ...
```

Figure 36. Getting the Value of a Container Data Member

Now that we have the value in the variable lngOrder, we can use it as we need. In this example, the value is displayed on the form in a text box with the code txtOrder.Text = lngOrder.

Output container information is set in a similar way, with the addition of a call to SetOuptutContainer() to send the information back to FlowMark. It is not actually sent back at the time of the function call, but buffered by the program execution client until the program ends. This example program does not send an output container value, but if it did, it would look similar to Figure 37.

```
Private Function SetSomeContainerData() As Long
    Dim outContainer As Container
    Dim sessionID As String
   Dim returnCode As Long

    Set outContainer = ContainerCtrl2.Container
    sessionID = outContainer.GetExecutionSessionID
    returnCode = outContainer.GetOutContainer(sessionID)
    If returnCode <> 0 Then
        ...
    End If

    returnCode = outContainer.SetValueStr("StrMember", "Value", False, 0 &)
    If returnCode <> 0 Then
        ...
    End If

    returnCode = outContainer.SetOutContainer(sessionID)
    If returnCode <> 0 Then
        ...
    End If

    SetSomeContainerData = 0&
End Function
```

Figure 37. Setting the Value of a Container Data Member

Notice that the output container was set from the object ContainerCtrl2. This is a different control from the one .* Index text: controls, two separate that was used for the input container information. If both the input container and the output container are going to be used within the activity program, two separate controls must be used on the form; otherwise, an error is generated at run time.

Another point to mention is using predefined data structure members. As of this writing, the C++ work list handler API set, from which the ActiveX controls are built, does not support accessing the predefined members directly. That is, it is not possible to use the technique shown in Figure 37 to set the value of a member such as _RC. To suggest an error condition within the program, you need to either set a user defined data member, or investigate the Windows API set for a method to return a code to the environment. The latter method is analogous to using a return statement at the end of the main() function in a C program.

7.6 Retrieving ODBC Database Data

The next step in our program is to get the data from the database that is necessary to make our price calculation. If you inspect the sample spreadsheet and the formulas it contains, you find that all of the design criteria are used to determine the costs of each production phase, the total cost, and the customer price. In our example, we are using an ODBC database, so the actual database

can be any that has an ODBC driver. First, we need to connect to our database, as shown in Figure 38 on page 53.

```
...
strConnect = "ODBC;UID=" & rbr.ODBCUserName & _
             ";PWD=" & rbr.ODBCUserPassword & _
             ";DSN=" & rbr.ODBCDataSource
Set ws = DBEngine.CreateWorkspace("ODBCWorkSpace", _
                                  rbr.ODBCUserName, _
                                  rbr.ODBCUserPassword, _
                                  dbUseODBC)
ws.DefaultCursorDriver = dbUseClientBatchCursor
Set cnn = ws.OpenConnection(rbr.ODBCDataSource, _
                            dbDriverComplete, _
                            False, _
                            strConnect)
...
```

Figure 38. Connecting to an ODBC Database

Notice in the code several variables that start with rbr. The variable rbr is an object of class RBRegistry, that is included as part of the redbook utility component. When this object is created (not shown in the previous code) it reads and stores several values from the Windows registry to use in the sample programs. This way we do not have to hard-code the values, and it is easier to get the code up and running. More information on this class is in Chapter 14, "The Redbook Utility ActiveX Component" on page 121.

After connecting to the database, we must retrieve the data for our order. The following section shows how to select the appropriate record and set some display fields based on the field data. See the sample source files for the complete code listing. They are listed in section 7.10, "Source Files and Installation" on page 55.

```
...
Set rs = cnn.OpenRecordset("select * from " & rbr.DesignTable & _
                           " where ID=" & txtOrder.Text)
txtBH.Text = rs.Fields("BASE_HEIGHT").Value
txtBW.Text = rs.Fields("BASE_WIDTH").Value
txtBT.Text = rs.Fields("BASE_THICKNESS").Value
txtTH.Text = rs.Fields("TEXT_HEIGHT").Value
...
```

Figure 39. Selecting a Record

7.7 Automating Microsoft Excel

The complete information necessary to update the spreadsheet is now available. Now we use another class defined by the redbook utility component, RBExcel, which we have used to encapsulate our common automation for Excel. As an example, other programs use the GetPrice method to retrieve the customer price for the order. The code for adding the order to the spreadsheet, retrieving the calculated price, and then updating the order table is shown in Figure 40 on page 54.

```
    ...
    rbe.AddOrder txtOrder.Text, datToday, txtBH.Text, _
                 txtBW.Text, txtBT.Text, txtBM.Text, _
                 txtBF.Text, txtTH.Text, dblTextLen, _
                 txtTC.Text

    txtPrice.Text = rbe.GetPrice(txtOrder.Text)

    rbe.UpdateOrderTable txtOrder.Text, txtPrice.Text
    ...
```

Figure 40. Automating an Excel Spreadsheet

There is not really much to it, although most of the processing is handled within the RBExcel object, rbe. The inputs to the AddOrder method are the design details retrieved from the database and a couple of calculated fields such as the current date and the text length in millimeters. The GetPrice method takes one argument, the order number, and can, therefore, retrieve the price for any order in the spreadsheet. The UpdateOrderTable method encapsulates the database access processing that we described previously. Once you have determined all of the requirements for your applications, it makes sense to create reusable components to make the programming task less complex.

7.8 Ending the Application and Destroying Objects

Finally, when the user presses the Exit button, the application saves the spreadsheet and closes Excel with the code shown in Figure 41. The Exit push button just unloads the main form, that triggers the Form_Unload event. This subroutine closes the connection and destroys all dynamically created objects. When an object is destroyed, its destructor method is called. This is where saving and closing Excel occurs. The destructor code is shown in Chapter 14, “The Redbook Utility ActiveX Component” on page 121.

```
Private Sub cmdExit_Click()
    Unload Me
End Sub

Private Sub Form_Unload(Cancel As Integer)
    cnn.Close
    Set rbe = Nothing
    Set rbr = Nothing
    Set cnn = Nothing
    Set ws = Nothing
End Sub
```

Figure 41. Exiting the Application

7.9 FlowMark Definitions

This example uses the following FlowMark definitions:

Processes

- Prepare Quote

Programs

- Calculate Price

Data Structures

- Order Number

7.10 Source Files and Installation

Source files:

- \examples\vbexcel\UpdateExcel.vbw
- \examples\vbexcel\UpdateExcel.vbp
- \examples\vbexcel\UpdateExcel.frm
- \examples\vbexcel\frmSplash.frm

Installing the UpdateExcel program:

- Run `\examples\vbexcel\install\Setup.exe`

Note

The redbook utility component is required to run this example program and should be installed first. Please refer to Chapter 14, “The Redbook Utility ActiveX Component” on page 121 for information about this component and for installation instructions.

Chapter 8. Printing and Sending a Reply to a Customer

Some of the most widely used applications are word processors and spreadsheets. So it is not surprising that they are a common part of a workflow process. How can these applications be integrated with FlowMark? If we look at the Microsoft Office products, we have the OLE Automation technology available. These applications provide services or functionality to other applications (and to people).

Let's introduce some basic terminology. The applications that provide services are called *automation servers*. The applications capable of driving automation servers are called *automation controllers*. Automation controllers can do almost the same thing you can do when you directly access the automation servers. The Microsoft Office products are automation servers.

In this chapter, we want to show how the FlowMark ActiveX Toolkit offers an alternative to the APIs. In this example, we use the container control to access the input and output containers.

8.1 Where This Example is Used

This sample implements the topic 3.1.3, "Send Mail Activity" on page 10, which is part of the "Prepare Quote" process.

8.2 What Techniques This Example Demonstrates

This example demonstrates the integration between:

- IBM FlowMark
- Microsoft Visual Basic
- Microsoft Office (Microsoft Word 97)
- Access to ODBC database
- FlowMark ActiveX Control - Container Control

8.3 Discussion

The goal of this activity is to generate a letter to be sent to the customer with information about the quote of the order. The letter contains all of the details of the order.

We have developed a Visual Basic application to:

- Access the input container (using Container Control). From the container, the application retrieves the Order Number.
- Access the database to retrieve the order details using the order number.
- Creates a Microsoft Word document based on a predefined template, and updates the document with the details of the order. When the updates are finished, the document is printed. Or, if the customer's e-mail address is known, a plain text file is saved, to be used in an electronic note. To do this, we have used OLE Automation.

8.4 Setting Up the Visual Basic Development Environment

This application uses some components. To use them within a Visual Basic program, they must be referenced in the development environment. To do this, select Project from the menu and select References, which brings up a dialog with a list box. Check the following items and press OK:

- FlowMark Redbook Utility Component
- Microsoft DAO 3.5 Object Library
- OLE Automation
- Microsoft Word 8.0 Object Library

Then, select Project from the menu and select Components, which brings up a dialog with a list box. Check the following items and press OK:

- FlowMark Container OLE Control module

8.5 Access Container

As an alternative to using the Flowmark Visual Basic APIs, you can use the FlowMark ActiveX Toolkit (only Windows NT). This toolkit provides a set of controls that allow you to access and set FlowMark information.

In particular, our application needs to access the input container to retrieve the order number. To do this, we have used the Container Control.

Before reading the input container, you need to get the SessionID. Using this sessionID, you can read or populate the Input Container object with the container element's values.

```
' Get a Container Object
Set inContainer = MainForm.ContainerCtrl1.Container
' Get the sessionID
sessionID = inContainer.GetExecutionSessionID
' Read the Input Container
lRC = inContainer.GetInContainer(sessionID)
If lRC <> 0 Then
  MsgBox ("Error, return code = " + CStr(lRC))
End If
```

Figure 42. Access to Input Container

To retrieve the value of the container element, we used the GetValueLng container's method, specifying the full name of the element. For each basic type (Long, String and Float), the control provides a specific method to retrieve values of these types.

In other contexts, it is helpful to carry out a more generic approach, querying the structure of the container and then retrieving its elements.

```
lRC = inContainer.GetValueLng("OrderNumber", IDOrder, False, 0& )
If lRC <> 0 Then
  MsgBox ("Error, RC =" + CStr(lRC))
  Exit Sub
End If
```

Figure 43. Retrieving Data Item's Values

In this sample, we do not need to set the output container. If this is the case, get a new container object for the output container, retrieve the output container of the activity using the *GetOutContainer(sessionID)*, and then use the SetValue*type* method to set the output container's elements.

The version of the FlowMark ActiveX Toolkit we are using does not support the setting of the fixed member *_RC*. The developer team is working on this problem.

8.6 Interacting with Microsoft Word

In this sample, we need to create a new Microsoft Word document, using a predefined template, to show all of the details of the order, including the price, the customer and shipping information, and the design details. This document, when finished, is mailed (printed and sent by postal mail, or saved and sent by e-mail) to the customer. We have developed a Visual Basic application that uses OLE Automation techniques to accomplish these tasks.

First, we create a new *"Word.Application"*object. This starts Microsoft Word and establishes a session with it. After specifying that Word remains visible, we add a new document, using a specific template file. The path of the template file must be specified as a command line parameter to the application program.

```
'Start MS Word application
Set X = CreateObject("Word.Application")
'Specify that Word remains visible
X.Visible = True
'Create a document using a template
X.Documents.Add (WordTemplate)
```

Figure 44. Starting Microsoft Word

After retrieving all of the order details from database, we use *FindReplaceWord* to update the document with the appropriate values. When these steps are finished, we print the document or save it as a plain text file (for e-mail).

```
...
FindReplaceWord X, "BILLABLE_AMOUNT", BillableAmount
...
FindReplaceWord X, "INSTRUCTIONS", SpecialInstructions
```

Figure 45. Updating the Document

The code in Figure 46 on page 60 shows the *FindReplaceWord* routine. The template file contains two-column tables for the customer (shipping and design

information). The first column contains the name of the field and the second contains a special label that is replaced with the correct value of the field.

To achieve this, we used Word's Find and Replace facility. For each element, we call the function *FindReplaceWord*, which searches the special label related to this element and then replaces it with the appropriate value.

```
Public Sub FindReplaceWord(X, findWord, replaceWord)
  ...
  With X.Selection.Find
    ' Ignore formatting
    .ClearFormatting
    ' Specify the search text
    .Text = findWord
    .Replacement.ClearFormatting
    ' Specify the replace text
    .Replacement.Text = replaceWord
    ' Run the search and replace
    .Execute Replace:=wdReplaceAll, Forward:=True, _
        Wrap:=wdFindContinue, MatchCase:=True, _
        MatchWholeWord:=True
  End With
  ...
End Sub
```

Figure 46. Find and Replace Routine

After the document is printed (or saved as a text file), we close the Active Document, without saving the changes, and then quit Word.

```
If IsMSWordOpen Then
  X.ActiveDocument.Close SaveChanges:=wdDoNotSaveChanges
  X.Quit
End If
```

Figure 47. Closing the Application

Note

The same techniques are used to format and print a shipping document when the Dispatch program activity is started. This activity is part of the Manufacture Sign process.

8.7 FlowMark Definitions

This example uses the following FlowMark definitions:

Processes

- Prepare Quote

Programs

- Send Mail Message

Data Structures

- Order Number

8.8 Source Files and Installation

Source files:

- \examples\vbword\GenerateLetter.vbp
- \examples\vbword\GenerateLetter.vbw
- \examples\vbword\GenerateLetter.frm
- \examples\vbword\GenerateLetter.bas
- \examples\vbword\Sign.dot

Installation:

1. Run \examples\vbword\install\setup.exe.
2. Copy the Microsoft Word template Sign.dot to the directory where you store this type of template. Update the NT page of Send Mail Message registration program, specifying the path where the template is located as a command line parameter to your application.

Note

The redbook utility component is required to run this example program and should be installed first. Please refer to Chapter 14, “The Redbook Utility ActiveX Component” on page 121 for information about this component and for installation instructions.

Chapter 9. Sending E-mail with Java

Use of the Internet for business ventures is increasing at a rapid pace. Although there are new technologies that make the Internet interactive, the main method of communication is through asynchronous electronic mail. Many Web sites accept orders through e-mail and send confirmations by return e-mail. Our scenario uses an interactive strategy for receiving the customer order, but uses the dependable delivery of e-mail to send information back to the customer.

This solution is a good fit for our scenario because there are several steps that must be completed after the order details are accepted before we are ready for more user intervention. In particular, the customer price for the order must be calculated, and the letter to be sent to the user must be created and formatted.

9.1 Where This Example is Used

The example discussed in this chapter is part of a sample FlowMark scenario described in Chapter 2, “Scenario Description” on page 7. The FlowMark implementation of this scenario is discussed in Chapter 3, “FlowMark Model Description” on page 9. The FlowMark process model details of the sample activity program presented here are described in:

- Topic 3.1.4, “Send E-mail Activity” on page 10.

This example is used as an automatic program activity within the Prepare Quote process. After the letter has been generated and saved in an ASCII file format, this program reads the file and send it through electronic mail to the customer. This activity is only run if the customer provided an e-mail address at the time of placing the order. This program is generic enough that it can be used in other processes whenever a file needs to be sent through e-mail.

9.2 What Techniques This Example Demonstrates

This example shows an automated activity program implemented with a programming language for which FlowMark does not have an application programming interface. Normally, this is not advantageous to do, but by using a platform independent language such as Java, we can use this program on any system that supports Java without any changes or recompiles. Therefore, this program is common to both of the scenario implementation platforms, OS/2 and Windows NT.

9.3 Accessing FlowMark Container Data

When an activity program is executed, it usually retrieves the input data container information by using the FlowMark language APIs, the work list handler C++ API, or the ActiveX controls. Since these methods are not available for Java, and because we do not need to set any members of the output container, we can use replacement variables as program parameters to access the input container data.

The example program in our scenario sends an e-mail message to a specific address by creating a socket connection to an SMTP server, sending the SMTP header data, sending the contents a message text file, and finally closing the

connection to the server. The program requires three parameters to accomplish this task: the SMTP server address, the e-mail address to receive the message, and the path and file name of the text file to be sent. Two data container members are used as parameters to satisfy these requirements. The program registration for this program has the attributes shown in Figure 48.

```
Path and file name     : java.exe
Command line parameters: smtpClient.class sg242184
                         %Customer.Email% /textpath/%Order_ID%.txt
```

Figure 48. Send E-mail Program Registration

When the program activity is executed, the program java.exe will run. This is a Java virtual machine for OS/2, which takes as a parameter the class to execute; in this case, smptClient.class. This Java application then accepts the rest of the parameters as command line arguments. At the time of execution, these are just literal values, the replacement variables having been replaced by the contents of the data container members.

9.4 Sending the E-mail Message

Now, we look at the Java class that actually processes the information to send the message. First, Figure 49 shows the beginning of our file, the declaration of function main(), and the initialization of the local variables used.

```
import java.net.*;
import java.io.*;

class smtpClient {
   public static void main( String args[] )
   throws FileNotFoundException, IOException {
      Socket s = null;
      DataInputStream sIn = null;
      DataOutputStream sOut = null;
      FileInputStream IStream = null;
      BufferedInputStream IBufStream = null;
      ...
```

Figure 49. The Main() Function

The socket object, s, is used to connect to the specified SMTP server. The data streams, sIn and sOut, are used to communicate with the server after the connection is established. The task of reading the contents of the message text file is handled by the stream objects IStream and IBufStream. Now, let's check for the necessary arguments and attempt to open the specified file. This is shown in Figure 50 on page 65.

```
...
try {
   if (args.length < 3) {
      System.out.println("Missing argument(s)");
      System.out.println("Usage: java]jview smtpClient " +
                         "SMTPServer email file");
      return;
   }
   IStream = new FileInputStream(args[2]);
   IBufStream = new BufferedInputStream(IStream, 100);
} catch( FileNotFoundException e) {
   System.out.println( "File not found: " + args[2] );
} catch( IOException e ) {
   System.out.println( e );
}
...
```

Figure 50. Opening the Message File

This version of the application just outputs messages to the standard output stream, generally the console. A production version should at least output to a log file. A BufferInputStream object is used to access the text file more efficiently. This particular example creates a buffered input stream over a file input stream with a buffer size of 100 bytes.

The next step is to connect to the SMTP server and create the data streams through which we communicate with the server. Figure 51 displays the code to accomplish this task.

```
...
try {
   s = new Socket( args[0], 25 );
   sIn = new DataInputStream( s.getInputStream() );
   sOut = new DataOutputStream( s.getOutputStream() );
} catch( UnknownHostException e ) {
   System.out.println( "Unknown host, could not connect" );
} catch( IOException e ) {
   System.out.println( e );
}
...
```

Figure 51. Connecting to the SMTP Server

A new socket object is created with the server address and port number for the connection. Here we are using the first command line argument, arg[0], as the server address, and the value 25 for the port, which is commonly the port reserved for SMTP communication.

After successfully connecting to the server, we first need to send the SMTP header information, including the e-mail address of the recipient, as shown in Figure 52 on page 66.

```
...
if( s != null && sIn != null && sOut != null ) {
   try {
      sOut.writeBytes( "MAIL From: sales@SignCo.com\n" );
      checkResponse(sIn, "220");

      sOut.writeBytes( "RCPT To: " + args[1] + "\n" );
      checkResponse(sIn, "220");

      sOut.writeBytes( "DATA\n" );
      checkResponse(sIn, "250");

      sOut.writeBytes( "From: sales@SignCo.com\n" );
      checkResponse(sIn, "250");

      sOut.writeBytes( "Subject: Order confirmation\n" );
      checkResponse(sIn, "354");
      ...
```

Figure 52. Sending the Header Information

The checkResponse() method is used to compare the SMTP server response against an expected return code that signifies success. The implementation of this method is shown in Figure 53.

```
private static void checkResponse(DataInputStream sIn, String okCode)
throws IOException {
    String response;

    response = sIn.readLine();
    System.out.println( "Server: " + response );
    if( response.indexOf( okCode ) == -1 ) {
        throw new IOException(response);
    }
}
```

Figure 53. Checking SMTP Server Responses

Once the header information has been sent to the server, the message body must be sent. The following loop reads the contents of the text file a byte at a time and sends the data to the SMTP server. The actual file read is not occurring for each stream read since the input is buffered. After the contents of the file are read and sent, the characters that end an SMTP transmission are sent. The server now knows it is ready to send the e-mail message. This is shown in the code fragment of Figure 54 on page 67.

```
            ...
            int count = IBufStream.available();
            int b;
            while (count > 0 ) {
               b = IBufStream.read();
               sOut.write(b);
               count--;
            }
            sOut.writeBytes( "\n.\n" );
            checkResponse(sIn, "250");
            ...
```

Figure 54. Sending the Message Text

The final step in the application is to close all data streams and to close the socket connection to the server. This is accomplished with the code shown in Figure 55.

```
            ...
            IBufStream.close();
            IStream.close();
            sOut.close();
            sIn.close();
            s.close();
         } catch( UnknownHostException e ) {
            System.out.println( "Trying to connect: "+e );
         } catch( IOException e ) {
            System.out.println( e );
         }
      }
   }
}
```

Figure 55. Closing Connections

As you can see, the code necessary to accomplish our task is short and simple. Of course, it should be strengthened for production use, but when finished, you have an application that can run on any platform for which there is a Java virtual machine. No changes to the code or recompilations are necessary.

9.5 FlowMark Definitions

This example uses the following FlowMark definitions:

Processes

- Prepare Quote

Programs

- Send e-mail message

Data Structures

- Order Number+e-mail

9.6 Source Files and Installation

Source files:

- \examples\javasmtp\smtpClient.java

Installing the smtpClient application:

- Copy \examples\javasmtp\install\smtpClient.class to the \exm\bin\ directory of the automatic activity server. In our setup, the Web server, FlowMark server, and automatic/unattended activity server are the same machine.

Chapter 10. Custom FlowMark Run-time Client - Powerbuilder Version

Currently, FlowMark provides run-time clients on several platforms:

- Standard FlowMark Run-time Client for Windows (3.1, 3.11, 95, NT)
- Standard FlowMark Run-time Client for OS/2
- Lotus Notes Run-time Client for OS/2
- Lotus Notes Run-time Client for Windows 95 and Windows NT

Sometimes we need a specific function, or we want to modify the default ones, or we want to display a new task-oriented GUI that hides all the details of work items and process instances. FlowMark provides a set of APIs to allow the development of custom run-time clients:

- C++ Work List APIs for OS/2, Windows and AIX

But, we have another interesting possibility; we can use the FlowMark ActiveX Toolkit on the Windows NT or 95 platform, which provides the following controls:

- Container control
- Server control
- Work List control
- Process List control

All of the controls provide a GUI except the container control. Without much effort, you can develop a custom run-time client based on this set of controls. Using the default GUI, you can save time and can dedicate your effort to developing the functions you need.

10.1 Where This Example is Used

This example is used in handling the ″Cut Material″ activity, which is part of the Manufacturing Sign process.

10.2 What Techniques This Example Demonstrates

This example demonstrates the integration between:

- IBM FlowMark
- Powerbuilder
- FlowMark ActiveX Toolkit:
  - Server control
  - Work List control
  - Container control

10.3 Discussion

In this sample, we have developed a Powerbuilder application that behaves like a FlowMark run-time client. Not all of the functionality of the standard run-time client is supported. We have just included the functionality needed to execute the ″Cut Material″ activity, which is part of the ″Manufacturing Sign″ process. However, the FlowMark ActiveX Toolkit provides all of the support needed to

develop a custom run-time client with the same functionality that the standard one has.

The goal is to provide a GUI application to the end user that:

- Allows starting, executing, and finishing jobs.
- Hides all of the concepts of activity, work list, process, and so on.
- Provides a new abstraction:
  - A job that has to be done.
  - A list of jobs that are waiting to be done.
- Provides mapping between the FlowMark abstraction and the application abstraction to ensure the dynamic of the process.

The user interacts with the application in this way: show me the job list, show me the next job, give me the job, and set the job as done.

The following detailed overview shows the most interesting part of the application.

10.3.1 Setting Up the Powerbuilder Development Environment

This application uses some FlowMark components. To use them within a Powerbuilder program, you open a window and choose Add OLE object. When the Insert Object window appears, choose Create New, and select the FlowMark component you need. Click on the window to add the control.

In this sample, we have used the following components:

- FlowMark Server OLE Control module
- FlowMark WorkList OLE Control module
- FlowMark Container OLE Control module

The first two controls have a GUI. In this sample, we do not want these GUIs to appear. So, using the properties of these controls, we set them as not visible. The container control does not have a GUI.

10.3.2 Server Registration and Logon

To carry out our custom run-time client, we need to log on to a FlowMark run-time server, query the work lists and work items, and access the elements of the input container of each work item.

The logon facility is provided by the server control. First, we need to register the server into a special structure of server arrays, which is maintained by the control itself. After a successful registration, we instantiate a server object and log on to the FlowMark run-time server using the *Logon* method.

The following code is related to Open Event for w_main window:

```
// adding server control to array
lSrvIndex = -1
IF ServerCtrl1.object.ServerArray.GetSize < 1 THEN
   lSrvIndex = ServerCtrl1.object.ServerArray.Add(REF sServer,
                                                  REF sDBName)
END IF
IF lSrvIndex = 0 THEN
  // Logon to FlowMark server
  oServer = ServerCtrl1.object.ServerArray.GetAt(lSrvIndex)
  lRC = oServer.Logon (REF sUserID, REF sPassword)
  IF lRC <> 0& THEN
    MessageBox ("Error", "... logon to server. RC=" + string(lRC))
  END IF
ELSE
  MessageBox ("Error", "... adding to server array. RC="
                                              + string(lSrvIndex))
END IF
```

Figure 56. Server Registration and Logon

10.3.3 Work List Setup

To set up the Work List, first we query the size of the Work List array object into the server object. If the size is at least one (that is, there is at least one work list), we instantiate a new work list object with it.

```
lWLIndex = oServer.WorkListArray.GetSize
IF lWLIndex > 0 THEN
  oServer.WorkListArray.SetAt((lWLIndex - 1),
                              WorkListCtr1.object.WorkList)
  WorkListCtrl1.object.RefreshWorkList
ELSE
  IF lWLIndex = 0 THEN
    MessageBox("Information", "No work list Found.")
  ELSE
    MessageBox("Error", "WorkListArray.GetSize. RC=" +
                                                 string(lWLIndex))
  END IF
END IF
```

Figure 57. Work List Setup

10.3.4 Filtering the Work List

In this sample, we are only interested in work items that meet the following conditions:

- They are related to the activity "Cut Material"
- Their state is "ready"

We used filters to work with the items that meet both conditions. To set a filter, you must specify:

- The intention: if you want to filter the work list, work items, process instances, or process templates.

- The filter specification: the name of the process or activity, the state, category, description, and so forth.
- Type of criteria:
  - String
  - Long
  - Date/time

```
//WorkItem=4, WorkList=3
idx = WorkListCtrl1.object.FilterArray.Add(4)
WLFilter = WorkListCtrl1.object.FilterArray.GetAt(idx)
strValue = ACTIVITY_NAME
filterSpec = 1
lRC = WLfilter.SpecifyCriterionStr(filterSpec, strValue) //1=Name
IF lRC <> 0 THEN
  MessageBox ("Error", "Filter.CriterioStr RC=" + string(lRC))
END IF
// READY=1, STATE=9
lngValue = 1                                                    ct Browse
lRC = WLfilter.SpecifyCriterionLong(9, lngValue)                ct Browse
IF lRC <> 0 THEN
    MessageBox ("Error", "Filter.CriterioLong RC=" + string(lRC))
END IF
```

Figure 58. Filtering the Work List

To apply the filters, you use the QueryWorkItems method, specifying the filter.

```
// Apply the filter to the WorkList
lRC = WorkListCtrl1.object.WorkList.QueryWorkItems(WLfilter)
```

Figure 59. Applying the Work List Filters

10.3.5 Container Access

Before accessing the input container, we have to initialize an Container object.

```
...
// Get an Container Object
inContainer = ContainerCtrl1.object.Container
...
```

Figure 60. Container Setup

10.3.6 Refreshing the Job List

Each time the user clicks the Refresh Job List button, or a job is terminated, the program refreshes the Job List. To do this, internally we refresh the work list by applying the specified filters. The final set of work items is displayed.

```
Integer ItemCount, i
OleObject WItem

// Refresh the Work List
WorkListCtrl1.object.RefreshWorkList
// Apply filter again ...
WorkListCtrl1.object.WorkList.QueryWorkItems(WLfilter)

// Get the number of items
ItemCount = WorkListCtrl1.object.GetItemCount

lastIndex = 0

FOR i= 1 TO ItemCount
  WItem = WorkListCtrl1.object.GetItemAt(i - 1)
  WItemA[lastIndex + 1]= WItem
  lastIndex = lastIndex + 1
NEXT
```

Figure 61. Refreshing the Job List

For each job or work item that is shown, we retrieve all of the values of the elements in the input container, as in Figure 62.

```
// Get the Input Container of the Work Item
lRC = WItemA[curIndex].GetInContainer(inContainer)

lRC = inContainer.GetValueLng("ID", REF lngID, False, 0)
IF lRC <> 0 THEN
  MessageBox("Error", "inContainer.GetValueLng")
  Return lRC
END IF
sle_ordernumber.Text = string(lngID)

lRC = inContainer.GetValueLng("Base_Height", REF lngHeight, False, 0)
IF lRC <> 0 THEN
  MessageBox("Error", "inContainer.GetValueLng")
  Return lRC
END IF
sle_height.Text = string(lngHeight)

...The same for Base_Width, Base_Thickness

lRC = inContainer.GetValueStr("Base_Material",REF sBMaterial,False,0)
IF lRC <> 0 THEN
  MessageBox("Error", "inContainer.GetValueStr")
  Return lRC
END IF
sle_material.Text = sBMaterial
```

Figure 62. Retrieving from Input Container

10.3.7 Skip Job

Here is some code that allows us to skip a job.

```
IF curIndex < lastIndex THEN
  curIndex = curIndex + 1
ELSE
  curIndex = 1
END IF
lRC = ShowJobData()
```

Figure 63. Skip Job

10.3.8 Get Job

When the user clicks the Get Job button, the selected row in the listbox is mapped into a specific work item. This work item is checked out, its state changes to Running and access becomes disabled for all other users.

```
...
// Check out the Work Item
lRC = WItemA[curIndex].CheckOut(inContainer)
IF lRC <> 0 THEN
  MessageBox("Error", "... check out Work Item. RC="+ string(lRC))
  ...
END IF
```

Figure 64. Get Job

10.3.9 Job Done

When the user clicks the Job Done button, the selected row in the listbox is mapped into a specific work item. This work item is checked in. The return code is set to 0. The activity has set the exit condition _RC=0. Then, after the Check in operation is issued, the activity ends.

```
OleObject outContainer
Long appRC
...

// Get an Container Object
outContainer = ContainerCtrl1.object.Container
// Check In
appRC = 0
lRC = WItemA[curIndex].CheckIn(REF outContainer, REF appRC)

IF lRC = 0 THEN
  // Refresh the Job List
  cb_refresh.EVENT Clicked()
ELSE
  MessageBox("Error", "...Check In Work Item. RC=" + string(lRC))
END IF
```

Figure 65. Job Done

10.3.10 Cancel Job

When the user clicks the Cancel Job button, the selected row in the listbox is mapped into a specific work item. This work item is Check in. The return code is set to -1. The activity has set the exit condition _RC=0. Then, after the Check in operation is issued, the activity becomes ready again.

```
OleObject outContainer
Long appRC

// Get an Container Object
outContainer = ContainerCtrl1.object.Container
// Check In - RC=-1
appRC = -1
lRC = WItemA[curIndex].CheckIn(REF outContainer, REF appRC)
IF lRC <> 0 THEN
  MessageBox("Error", "...Check In. RC=' + string(lRC))
END IF
```

Figure 66. Cancel Job

10.4 FlowMark Definitions

This example uses the following FlowMark definitions:

Processes

- Manufacture Non Web

Programs

- No Execute

Data Structures

- Design

10.5 Source Files and Installation

Source Files:

- \examples\pbrtc\cutmaterial.pbl

Installation:

1. Run \examples\pbrtc\install\disk1\setup.exe.
2. Select Install Option and press Next.
3. Select 32-bit short/long names and press Next.
4. Select Custom to change the target directory and press Next.
5. Check Power Builder - FlowMark Custom Runtime Client, click on Details button to change the target directory. Answer Yes to the warning message and press Ok. Then, press Next.
6. Select Prompt for each and press Next.
7. Select Install now and press Next.

8. When the Please, insert the following disk (or type the new path) and press Enter message appears, enter the full path of the source installation directory.
9. When the Completed copying files message appears, press Next.
10. Select Yes and press Next to create the Program Group and Icons.
11. Press Finish when the Setup Complete message appears.

Chapter 11. Custom FlowMark Run-time Client - Visual Basic Version

In a previous chapter, we have shown a Powerbuilder-based custom FlowMark Run-time Client. In this chapter, we show how to develop a Visual Basic custom FlowMark Run-time Client that uses the FlowMark ActiveX Toolkit.

11.1 Where This Example is Used

This example is used to implement the ″Engrave Text″ activity, which is part of the Manufacturing Sign Process described in topic 3.2.3, “Manufacturing Steps Sub-Process” on page 11.

11.2 What Techniques This Example Demonstrates

This example demonstrates the integration between:

- IBM FlowMark
- Microsoft Visual Basic
- Access to ODBC database
- FlowMark ActiveX Toolkit:
  - Server control
  - Work List control
  - Container control

11.3 Discussion

In this redbook, we have developed a Visual Basic application that behaves like a FlowMark run-time client without all of the functionality of the standard run-time client. We have included only the functions needed to execute the ″Engrave Text″ activity, which is part of the ″Manufacturing Sign″ process. However, the Flowmark ActiveX Toolkit provides all of the support needed to develop a custom run-time client with the same functionality that the standard one has.

The goals of a custom run-time client are almost the same as those you see in Chapter 10, “Custom FlowMark Run-time Client - Powerbuilder Version” on page 69.

- Allows you to start, execute, and finish tasks or jobs.
- Allows you to filter and group the information of the jobs.

  Note: Remember that the information the application shows is extracted from the input container. This filtering and grouping cannot be done using the standard run-time client.
- Hides the details of activity, work list, process, and so on.
- Provides a new abstraction:
  - A job or task that has to be done.
  - A list of jobs that are waiting to be done.
- Provides mapping between the FlowMark abstraction and the application abstraction to ensure the dynamic of the process.

11.3.1 Setting Up the Visual Basic Development Environment

In order to use FlowMark ActiveX components within a Visual Basic program, they must be referenced in the development environment. To do this, select Project from the menu, and then select References, which brings up a dialog with a list box. Check the following items and press OK:

- FlowMark Redbook Utility Component
- Microsoft DAO 3.5 Object Library

Then, select Project from the menu, and select Components, which brings up a dialog with a list box. Check the following items and press OK:

- FlowMark Server OLE Control module
- FlowMark WorkList OLE Control module
- FlowMark Container OLE Control module

After you make these components available, you have to add an Server control, a WorkList control, and Container control to the Visual Basic form. The first two controls have a GUI. In this sample, we do not want these GUIs to appear. So, using the properties of these controls, set them as not visible. The Container control does not have a GUI.

11.3.2 Server Registration and Logon

To implement our custom run-time client, we need to log on to a FlowMark run-time server, query the work lists and work items, and access the elements of the input container of each work item.

The logon facility is provided by the server control. First, we need to register the server into a special structure of server array, which is maintained by the control itself. After a successful registration, we instantiate a server object and log on to the FlowMark run-time server, using the *Logon* method.

```
If ServerCtrl1.ServerArray.GetSize < 1 Then
  lSrvIndex = ServerCtrl1.ServerArray.Add(sServer, sDBName)
End If
If lSrvIndex < 0 Then
  MsgBox "Error adding server to server array = "+ CStr(lSrvIndex)
  Exit Sub
End If

Set oServer = ServerCtrl1.ServerArray.GetAt(lSrvIndex)
...
lRC = oServer.Logon(General.sUserId, General.sPassword)
If lRC <> 0 Then
  MsgBox "Error from logon =" + CStr(lRC)
  Exit Sub
End If
```

Figure 67. Server Registration and Logon

11.3.3 Work List Setup

To set up the work list, first we have to query the size of the work list array object into the server object. If the size is at least one (there is at least one work list), we instantiate a new work list object with it.

```
Dim lWLIndex as Long
...
lWLIndex = oServer.WorklistArray.GetSize
If lWLIndex > 0 Then
  oServer.WorklistArray.SetAt (lWLIndex - 1), _
                              WorkListCtrl1.Worklist
  WorkListCtrl1.RefreshWorklist
Else
  If lWLIndex = 0 Then
    MsgBox "No work list found."
  Else
    MsgBox "Error returned from WorkListArray.GetSize =" _
           + CStr(lWLIndex)
  End If
End If
```

Figure 68. Work List Setup

11.3.4 Filtering the Work List

In this sample, we are interested only in work items that meet the following conditions:

- They are related with the activity "Engrave Text"
- Their state is "ready".

We used the filters to select the work items that meet both conditions. To set a filter, you must specify:

- The intention: if you want to filter the work list, work items, process instances, or process template.
- The filter specification: the name of the process or activity, the state, category, description, and so on.
- Type of criterion:
  - String
  - Long
  - Date/Time

You can specify more than one filter to be applied to the work list.

```
Set filterArray = WorkListCtrl1.FilterArray
idx = filterArray.Add(4) 'WorkItem=4  WorkList=3
Set WLfilter = filterArray.GetAt(idx)
strValue = ACTIVITY_NAME
filterSpec = 1 'Name
lRC = WLfilter.SpecifyCriterionStr(filterSpec, strValue) '1=Name
If lRC <> 0 Then
  MsgBox "Filter.CriterioStr RC= " + CStr(lRC)
End If
lngValue = Ready
filterSpec = 9 'State
lRC = WLfilter.SpecifyCriterionLong(filterSpec, lngValue)
If lRC <> 0 Then
  MsgBox "Filter.CriterioLong RC= " + CStr(lRC)
End If
```

Figure 69. Filtering the Work List

11.3.5 Container Setup

Before accessing the input container, we have to initialize an Container object.

```
' Get a Container Object
 Set inContainer = ContainerCtrl1.Container
```

Figure 70. Set the InContainer Object

11.3.6 Refreshing the Job List

Each time the user clicks the Refresh Job List button, or a job is terminated, the program refreshes the Job List. To do this, our program must refresh the work list, applying the specified filters. The resulting set of work items is filtered and grouped according to the criteria specified by the user through the GUI. The final set of work items is displayed.

The Job List is implemented as a set of arrays used to store the values of the elements retrieved from the input container of each work item.

For each work item, we use the GetValueLng and GetValueStr functions to retrieve the values from the input container.

```
' Refresh the work list
WorkListCtrl1.RefreshWorklist
lRC = WorkListCtrl1.Worklist.QueryWorkitems(WLfilter)

' Get the number of items
ItemCount = WorkListCtrl1.GetItemCount
lastIndex = -1
If ItemCount > 0 Then

  For i = 1 To ItemCount
    Set WItem = WorkListCtrl1.GetItemAt(i - 1)
    lastIndex = lastIndex + 1
    Set WItemA(lastIndex) = WItem
    lRC = WItem.GetInContainer(inContainer)
    With inContainer
      lRC = .GetValueLng("ID", lngID, False, 0&)
      ONumberA(lastIndex) = lngID
      lRC = .GetValueStr("Base_Material", sBMaterial, False, 0&)
      BMaterialA(lastIndex) = sBMaterial
      lRC = .GetValueStr("Text_Style", sTStyle, False, 0&)
      TStyleA(lastIndex) = sTStyle
      lRC = .GetValueLng("Text_Height", lngHeight, False, 0&)
      THeightA(lastIndex) = lngHeight
      lRC = .GetValueStr("Text", sText, False, 0&)
      TextA(lastIndex) = sText
    End With
    OrderA(lastIndex) = lastIndex
  Next i
End If
```

Figure 71. Refreshing the Job List

11.3.7 Get Job

When the user clicks the Get Job button, the selected row in the listbox is mapped into a specific work item. This work item is checked out, its state changes to Running and it becomes disabled for access by all other users.

```
...
' Check out the work item
curIndex = GetIndex(lstOrderInfo.ListIndex)
lRC = WItemA(curIndex).CheckOut(inContainer)
If lRC <> 0 Then
  MsgBox "Check Out RC= " + CStr(lRC)
  ' Disable Job Done Button and Enable some Controls
  ...
End If
```

Figure 72. Get Job

11.3.8 Job Done

When the user clicks the Job Done button, the selected row in the listbox is mapped into a specific work item. This work item is checked in. The return code is set to 0. The activity has set the exit condition _RC=0. Then, after the Check in operation is issued, the activity ends.

```
Dim outContainer as Object
Dim appRC as Long
...
' Get a Container Object
Set outContainer = ContainerCtrl1.Container
' Check In - appR=0
appRC = 0
lRC = WItemA(curIndex).CheckIn(outContainer, appRC)

If lRC <> 0 Then
  MsgBox "Check In. RC= " + CStr(lRC)
End If
```

Figure 73. Job Done

11.3.9 Job Cancel

When the user clicks the Cancel Job button, the selected row in the listbox is mapped into a specific work item. This work item is checked in. The return code is set to -1. The activity has set the exit condition _RC=0. Then, after the Check in operation is issued, the activity becames ready again.

```
Dim outContainer as Object
Dim appRC as Long
...
' Get a Container Object
Set outContainer = ContainerCtrl1.Container
' Check in
appRC = -1
lRC = WItemA(curIndex).CheckIn(outContainer, appRC)
If lRC <> 0 Then
  MsgBox "Check In. RC= " + CStr(lRC)
End If
```

Figure 74. Job Cancel

11.4 FlowMark Definitions

This example uses the following FlowMark definitions:

Processes

- Manufacture Non Web

Programs

- No Execute

Data structures

- Design

11.5 Source Files and Installation

Source files:

- \examples\vbrtc\Engrave.vbp
- \examples\vbrtc\Engrave.vbw
- \examples\vbrtc\MainForm.frm
- \examples\vbrtc\LogonForm.frm
- \examples\vbrtc\Engrave.bas

Installation:

- Run \examples\vbrtc\install\setup.exe.

Note

The redbook utility component is required to run this example program and should be installed first. Please refer to Chapter 14, “The Redbook Utility ActiveX Component” on page 121 for information about this component and for installation instructions.

Chapter 12. Custom Run-Time Client - Visual C++

The FlowMark C++ API contains the most functionality of all the FlowMark APIs. Just like the APIs from C, Visual Basic, REXX, and COBOL, it can be used by an activity program to access its own input and output containers and to perform process control functions such as start, suspend, resume, terminate, and change activity state. In addition, the C++ API provides a large amount of extra functionality not found in the other APIs. This extra functionality is sufficient to build custom run-time clients. This chapter discusses the construction of a custom run-time client using Visual C++.

12.1 Where This Example is Used

The run-time client described by this chapter can be used to inspect and manipulate the activities of any FlowMark process. In the context of this book's scenario, this run-time client can be used to list, check out, and finish the manufacturing steps of the Manufacture Sign process described in Chapter 3, "FlowMark Model Description" on page 9.

12.2 What Techniques This Example Illustrates

This chapter describes the construction of a custom FlowMark run-time client using the FlowMark C++ API and the Microsoft Visual C++ Version 4.2 development environment. The example shows how to:

- Log onto FlowMark
- Fetch a work list
- Display a work list
- Start, finish, restart, and check out a selected work item
- Suspend, resume, and terminate a process related to a work item
- Inspect the contents of a work item's input container
- Handle FlowMark error conditions

12.3 Designing the User Interface

In this example, we want to build a customized run-time client that can be used to list, check out, and finish the steps of the Manufacture Sign process. We want to provide the user with all the information required to complete a manufacturing step from the work list itself, rather than writing special applications for each step. Also, rather than writing code that is scenario-specific, we want build a run-time client that can be reused in other scenarios.

Our user interface consists of a standard Single Document Interface (SDI) frame window that displays a multi-column list view. The columns of our list view can contain the attributes of an activity or, unlike the standard run-time client, values from the activity's input container.

The **File** pull-down contains normal file operations such as **New**, **Open**, **Save**, and **Save As...**. These options can be used to save the configuration of the list view in a file so that it can be recalled at a later time. The **File** pull-down also contains a **Connection...** option that can be used to invoke a connection dialog

that allows the user to manipulate the state of the connection to the FlowMark server and to select the FlowMark work list from which items are displayed.

The **Activity** pull-down contains options for manipulating the selected activity in the list view. In addition to operations found in the standard run-time client such as **Start**, **Restart**, and **Finish**, there is an option, **Checkout**, that can be used to check out the selected work item to suggest that it is being worked on. On the **Activity** pull-down, there is also a **Process** sub-menu that allows the user to issue operations (**Suspend**, **Resume**, and **Terminate**) against the FlowMark process that contains the selected work item. Finally, there is an **Inspect...** option that can be used to inspect the contents of a work item's input container.

The **View** pull-down contains options that affect the presentation of the view. The **Arrange...** option invokes a dialog that allows the user to configure which columns contain which attributes or input container members and what their labels are. The **Container Data** option is used to show whether container data is fetched automatically when the view is refreshed. By default, this option is disabled. The **Refresh** option discards the current work list and loads a fresh copy from the FlowMark server.

It should be noted that this run-time client lacks many of the features of the standard run-time client. For example, it does not allow multiple selection, displaying or manipulating process lists, specifying filtering or sorting criteria, displaying work item or processing notifications, displaying multiple work lists simultaneously, or displaying a graphic process monitor. In principle, however, all of these features, except for the graphical process monitor, can be carried out by exploiting the functionality of the FlowMark C++ API.

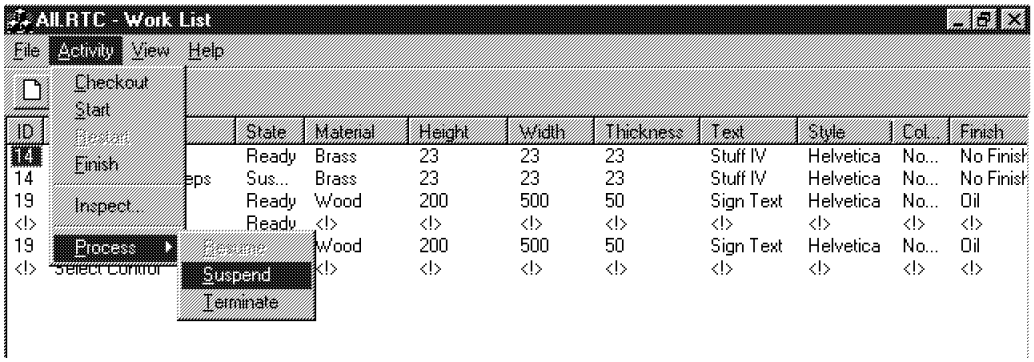


Figure 75. The C++ Custom Run-Time Client

12.4 Designing the Implementation

The custom run-time client described here was built using Microsoft Developer's Studio. The AppWizard tool was used to create a Microsoft Foundation Class (MFC) single document interface skeleton application, which was then edited to add and subtract the desired features. Table 2 on page 87 presents a list of the core classes used along with their responsibilities.

Table 2 (Page 1 of 2). Visual C++ Implementation - Class Responsibilities

Class	Responsibilities
CWorkListApp	This application class creates the document template that relates the document class (CWorkListDoc) to the frame (CWorkListFrame) and view (CWorkListView) classes that are used to display its instances. No customization of the generated skeleton class is required.
CWorkListFrame	This frame window class creates the menu and tool bars used by the application. No customization of the generated skeleton class is required.
CWorkListDoc	Derived from CDocument, an instance of this class acts like a *document* in the MFC document-view framework. It is responsible for managing the definition of the view′s columns and the contents of the view′s rows. CWorkListDoc manages the state of the FlowMark connection on behalf of the view, although it delegates those responsibilities to an instance of the CFlowmarkLogon class. As an actor in the MFC serialization framework, a CWorkListDoc object is also responsible for serializing column definitions to and from persistent storage in response to commands invoked from the **File** pull-down.
CWorkListView	Derived from CListView, an instance of this class acts like a *view* in the MFC document-view framework. It is responsible for managing the presentation of the CWorkListDoc in a multi-column list control and for controlling and responding to user input. There is a one-to-one relationship between items of the list control and CWorkListRow instances and between columns of the list control and CWorkListColumn instances.
CWorkListRow	A CWorkListRow object adapts an instance of the C++ FlowMark API ExmWorkitem class with a simplified interface that can be used by the CWorkListView class. Besides a work item object, it contains a pointer to a cache of the related input container object if it has been accessed. Most of its methods correspond to command options in the user interface.
CWorkListColumn	An instance of the CWorkListColumn remembers the width and label of a view column. Its subclasses (CWorkListInputMember, CWorkListAttribute) know how to query a row object to extract the value of the cell that lies at the intersection of the column and row. The CWorkListAttribute subclass knows how to query a CWorkListRow object to extract the value of a particular attribute of the work item. Supported attributes include: Activity, Process, State, Description, and Staff. The CWorkListInputMember subclass knows how to query a CWorkListRow object to extract the value of a particular member of the related input container. As actors in the MFC serialization framework, CWorkListColumn objects know how to serialize their state with MFC CArchive objects.

Table 2 (Page 2 of 2). Visual C++ Implementation - Class Responsibilities

Class	Responsibilities
CFlowmarkLogon	This dialog class manages the state of a FlowMark connection. When displayed modally, it allows the user to either log on to or off from a FlowMark server, depending on the current connection state. When logged on to a server, this dialog allows the user to select an existing work list for display in the view or to create a new work list with default filtering criteria.
CFlowmarkException	This exception class is used to represent the details of an exceptional processing condition that is met while using the FlowMark API. If an instance of this exception is thrown, and it is not caught by intervening layers, the MFC framework catches it and displays its text in a modal dialog box.
CContainerView	This dialog class displays the contents of a FlowMark container in a multi-column list control. A context menu in this control allows the user to add the selected input member to the columns of the main view.
CArrangeDialog	This dialog class provides an editor that allows the user to add, edit, delete, and re-arrange the columns of the main view.
CEditColumnDlg	This dialog class allows the user to edit the definition of an individual column.

12.5 Logging On to FlowMark

Before the C++ API can be used to perform useful work, it is necessary to log on to the FlowMark server. With our run-time client, logging on is initiated in three ways:

- Selecting the **Connection...** from the **File** pull-down.
- Clicking the left mouse button on the client area when there is no connection.
- Opening a view configuration (.RTC) file when there is no connection.

In response to these actions, a `CFlowmarkLogon` dialog box is displayed and, assuming the user is not already logged on, the user is prompted for the user ID, password, database name, and server name to be used for the connection. These values are then used to assign parameter values for the construction of an `ExmServer` object and a later call to its `Logon()` method.

```
//
//  Called by CFlowmarkLogon::OnLogon()
//
...
void CFlowmarkLogon::Logon()
{

   APIRET rc;

   // logoff an existing connection first

   if (m_Server.IsLoggedOn()) {
      Logoff();
   }

   // construct a new server object

   m_Server = ExmServer((const char *)m_ServerName,
                        (const char *)m_Database);

   // logon to the new server

   rc = m_Server.Logon((const char *)m_Userid,
                       (const char *)m_Password);
   if (rc != EXM_API_OK) {
      throw new CFlowmarkException(...);
   }

   return;
}
```

Figure 76. Logging on to a FlowMark Server with the C++ API

After logging, the ExmServer::QueryWorkLists() method is called to populate the work list combo box control with a list of the user's work lists. If the CFlowmarkLogon::m_WorkList member contains the name of an existing work list, that work list is set as selected in the combo box control. Otherwise, the work list that matches the user's logon ID is selected.

```
void CFlowmarkLogon::FillWorkLists()
{
   //
   // erase the existing list
   //

   m_WorkLists.erase(m_WorkLists.begin(), m_WorkLists.end());

   if (m_Server.IsLoggedOn()) {

      rc = m_Server.QueryWorklists(m_WorkLists);
      if (rc != EXM_API_OK) {
         throw new CFlowmarkException(...);
      }
   }

   ... // fill the combo box with the list of work lists

}
```

Figure 77. Querying the User's List of Work Lists

At the user's option, a new work list may be created by typing a new name into the combo box control and selecting the **Create** push button. In response to this selection, the CFlowmarkLogon class calls the ExmServer::CreateWorklist() method to create a new work list. For simplicity, assume that the name of the work list is also the name of the user ID and that no filtering criteria are required.

```
void CFlowmarkLogon::OnCreateWorkList()
{
...

   if (UpdateData(true)) {

           ExmFilter   filter(ExmFilter::Worklist);     // a default filter
         ExmWorklist   newWorkList;
              APIRET   rc;

      rc = m_Server.CreateWorklist(
              (char const *)m_WorkList, // work list name
              (char const *)m_WorkList, // userid
              filter,
              newWorkList);

      if (rc != EXM_API_OK) {
         throw new CFlowmarkException(...);
      }

      ... // fill the work lists, enable the dialog controls

   }

}
```

Figure 78. Creating a New Work List

12.6 Fetching a Work List

Our run-time client needs to fetch a work list from the host whenever the user logs on, changes the active work list, or selects the **Refresh** option of the **View** pull-down.

The `CWorkListDoc::Refresh()` method populates a document object with the contents of the selected work list. First, we search the list of work lists, by name, for the active work list. Then, after finding the active work list, we use the `ExmWorklist::QueryWorkitems()` method to fetch the contents of that list into a vector of work items. A default filter is used here; however, we can substitute a more restrictive filter to reduce the number of work items retrieved. We then iterate through that vector and use each element to build a corresponding element in the document's array of `CWorkListRow` objects. Finally, we call the `CDocument::UpdateAllViews()` method to notify all the dependent views that their document has been updated. Views use this notification to update the visual presentation of the work list.

```
void CWorkListDoc::Refresh(BOOL fNotifyViews)
{

               // remove existing rows
               // check that we are already logged on.
    ...

                    APIRET   rc;
                 ExmFilter   filter(ExmFilter::Workitem);
       vector<ExmWorkitem>   workItems;
       vector<ExmWorklist> & workLists = m_Connection.m_WorkLists;

   vector<ExmWorklist>::iterator   i;

    for (i = workLists.begin();
         i != workLists.end();
         ++i) {

        ExmWorklist & wl = *i;

        if (wl.Name() == (const char *)m_Connection.m_WorkList) {

            rc = wl.QueryWorkitems(filter, workItems);

            if (rc != EXM_API_OK) {
                throw new CFlowmarkException(...);
            }

        }

    }


   ...
   // fills m_Rows with CWorkListRow objects built with
   // items of m_WorkList, then notify all views of the
   // update ...

}
```

Figure 79. Fetching a Work List

12.7 Displaying a Work List

Whenever the work list in the document object is updated, the document's view is notified through an indirect call to the CWorkListView::OnUpdate(). This notification gives the view an opportunity to update its visual representation of the document. Unless it is given a hint that allows it to perform a more efficient update, the CWorkListView::OnUpdate() method discards all the columns and items in the list control, reads the new column definitions from the document object and then populates the items of the control with the cells of the document's rows.

```
void CWorkListView::OnUpdate(CView*, LPARAM lHint, CObject* pHint)

... // cleanup existing view, check for hints

   for (i = 0; i < doc.GetRowCount(); ++i) {

      CWorkListRow & row = doc.GetRow(i);
               int   x;

      x = ctl.InsertItem(i, row.GetValue(0));

      ctl.SetItemData(x, i);

      for (unsigned int j = 1; j < doc.GetColumnCount(); ++j) {

         ctl.SetItemText(x, j, row.GetValue(j));

      }

   }

... // save column widths, update list of valid actions
```

Figure 80. Inserting Work Items into a Visual Control

Each row contains N cells, where N is the number of columns in the view. To get the value of the jth cell of a row, the view calls the CWorkListRow::GetValue() method with a parameter of j. This method uses the document object to find the jth column and then passes itself as a reference to the GetValue() method of that column. CWorkListColumn::GetValue() is a virtual method, so that the actual method invoked depends on the run-time type of the column object. If it is an instance of CWorkListAttribute, the column calls one of the work item's attribute methods to obtain a value. Otherwise, if it is an instance of CWorkListInputMember, the column fetches the work item's input container and gets one of its members. Figure 81 illustrates this sequence of calls if the column is an attribute column that is bound to the Name() attribute of a work item.

```
VIEW            DOC              ROW              COLUMN      WORK ITEM

  ---------------- GetValue(j) -->

                  <--GetColumn(j)--

                                    --GetValue(row)-->

                                    <--GetWorkItem()--

                                                        --Name()-->
```

Figure 81. Call Trace for CWorkListRow::GetValue() Method

12.8 Performing Actions on the Work Item

The **Activity** pull-down or context menu allows the user to initiate an action on the selected work item. This section describes how the FlowMark C++ API is used to determine which menu selections are available at a given point in time and how those selections are processed when the user makes them.

The view keeps track of the selected row by tracking mouse clicks and key presses. After each of these events, it calls `CWorkListRow::GetActions()` to determine which actions the selected row currently supports. The row examines the state of its related work item using `ExmWorkitem::State()` and answers an unsigned integer that contains a set bit for every action that is currently valid. These bits are used by the view to enable or disable menu items, as appropriate, just before the display of the **Activity** pull-down or context menus.

```
//
// called on key down or mouse click events to track
// available actions of currently selected item...
//

void CWorkListView::UpdateActions()
{
   CWorkListRow * row = GetCurrentRow();
   if (row) {
      m_ValidActions = row->GetActions();
   } else {
      m_ValidActions = 0;
   }
}

//
// answers the set of currently available actions...
//

unsigned int CWorkListRow::GetActions() const
{
    unsigned int rv = 0;

    switch (m_WorkItem.State()) {

        case ExmWorkitem::running:
            rv = actRestartActivity|actFinishActivity
                 |actSuspendProcess|actTerminateProcess;
            break;

        ... // other states

    }

    return rv;

}

//
// called by the framework, just before displaying the menu
// allow us to enable/disable menu items appropriately.
//

void CWorkListView::OnUpdateActStart(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_ValidActions & CWorkListRow::actStartActivity);
}
```

Figure 82. Enabling Menu Options

Once the user has selected an item from a menu, the appropriate handler method such as CWorkListView::OnActStart() of the view class is called. The handler method uses information associated with the selected item to locate the corresponding document row, and then calls the appropriate action method of the row. The row delegates the implementation of this method to its related work item object and then refreshes itself with a call to its own CWorkListRow::Refresh() method.

```
void CWorkListView::OnActStart()
{
   SaveColumnWidths();

   CWorkListRow * row = GetCurrentRow();
   if (row) {
      row->StartActivity();
   }

}

void CWorkListRow::StartActivity()
{
    APIRET rc = m_WorkItem.Start();
    Refresh();
    if (rc != EXM_API_OK) {
       throw new CFlowmarkException(...);
    }
}
```

Figure 83. Invoking an Action

The call to the Refresh() method is necessary because action methods such as Start() may alter the state of the work item object, perhaps even deleting it, and such changes in state need to be reflected in the related row object, the containing document, and any dependent views.

To refresh a row, we first call the ExmWorkitem::Refresh() method. If the return code shows that the work item no longer exists, we ask the document to refresh itself. This forces the active work list to be refreshed. The reason to perform a refresh here is that not only do we want to delete the current work item from our document and its views, but also we want to insert any other work items that may have appeared as the result of the current work item's change of state.

Otherwise, if the Refresh() method returns with a normal return code, we call the document's UpdateAllViews() method with a hint that suggests that the view should refresh the presentation of just this particular row.

```
void CWorkListRow::Refresh()
{
    APIRET rc = m_WorkItem.Refresh();
    switch(rc) {

        case EXM_API_ERROR_DOES_NOT_EXIST:
            m_Doc->Refresh();
        break;

        case EXM_API_OK:
            m_Doc->UpdateAllViews(0, CWorkListDoc::RowUpdated, this);
        break;

        default:
            throw new CFlowmarkException(...);
    }
}
```

Figure 84. Invoking an Action

12.9 Using the ExmWorkItem::CheckOut Method

The **Check Out** menu item is not available in the standard run-time client. In our client, we provide this option so that we can demonstrate using the ExmWorkitem::Checkout() method.

The ExmWorkitem::CheckOut() method is similar to the Start() method in that it changes the state of the receiving work item to ″Running″. However, unlike the Start() method, the CheckOut() method does not result in invocation of an activity program by a FlowMark program execution client. Instead, the caller of the CheckOut() method assumes responsibility for eventually restarting, finishing, or checking in the work item.

In our scenario, we want to use our run-time client during the manufacturing steps of the Manufacture Sign process to check out a work item to show that it is being worked on. When a manufacturing step is complete, we simply use the **Finish** action to terminate the work item and so let the work flow go to the next step. In our case, we do not need to use the corresponding CheckIn() API because there is no data being passed between the steps of the manufacturing process.

Since our run-time client allows members of a work item′s input container to be displayed in separate columns alongside other attributes of the work item, we exploit the fact that the CheckOut method populates a container variable with the contents of the activity′s input container. As a result, after checking out a work item, the user can immediately see on the work list any relevant input data associated with the work item.[5]

[5] Of course, a similar effect could be achieved significantly more efficiently with the standard run-time client by including parameter markers within the activity′s description.

```
void CWorkListRow::CheckOutActivity()
{
    ... // initialize m_Container if it hasn't been already

    rc = m_WorkItem.CheckOut(*m_Container);
    Refresh();
    if (rc != EXM_API_OK) {
       throw new CFlowmarkException(...);
    }
}
```

Figure 85. Calling the ExmWorkitem::CheckOut() Method

12.10 Performing Actions on the Related FlowMark Process

The **Activity** menu of our run-time client contains a **Process** sub-menu that conveniently allows the user to suspend, resume, or terminate the related FlowMark process. Processing these actions is similar to processing actions that are applied directly to a work item. The only difference is that for the latter actions we need to navigate from the work item to the related process instance using the ExmWorkitem::GetProcessInstance() method.

```
void CWorkListRow::SuspendProcess()
{
                APIRET   rc;
    ExmProcessInstance   p;

    rc = m_WorkItem.GetProcessInstance(p);
    if (rc != EXM_API_OK) {
        throw new CFlowmarkException(...);
    }

    rc = p.Suspend(false);
    Refresh();
    if (rc != EXM_API_OK) {
        throw new CFlowmarkException(...);
     }

}
```

Figure 86. Suspending a Process Instance

12.11 Inspecting the Contents of an Input Container

The standard FlowMark run-time client does not allow the user to inspect, in an ad hoc manner, the contents of any activity's input container. Such a feature is not usually desirable, since it can make sensitive process data available to any user equipped with the standard run-time client and a suitably privileged FlowMark user ID.

From a FlowMark developer's standpoint, however, this restriction can sometimes be a hindrance to debugging process flows and activity implementations. We chose to alleviate this restriction in our run-time client by

adding the **Inspect...** option to the **Activity** menu. If you select this option, the contents of the selected work item's input container are displayed in a two-column list box.

This shows how to traverse the structure of the input container to visit all of the leaf nodes using a recursive method called `CContainerView::InsertElement()`. It should be noted that we did not use the `Leaves()` method of the `ExmContainer` class. While this method populates an `ExmContainerElement` vector with all the leaf nodes of the container, it is not subsequently possible to obtain the fully-qualified names of each element of the vector. Since we need to preserve the fully-qualified name of each element, the parameters of our traversal function include a `prefix` parameter. This parameter is used to form the fully-qualified name of each element.

```
void CContainerView::InsertElement(CString const & prefix,
                                   ExmContainerElement const & ce)
{

    if (ce.IsStruct()) {

                vector<ExmContainerElement>   members;
        vector<ExmContainerElement>::iterator   i;

        ce.StructMembers(members);
        for (i = members.begin(); i != members.end(); ++i) {

            InsertElement(prefix+CString(ce.Name())+CString("."),*i);

        }

    } else if (ce.IsArray()) {

                vector<ExmContainerElement>   elements;
        vector<ExmContainerElement>::iterator   i;

        ce.ArrayElements(elements);
        for (i = elements.begin(); i != elements.end(); ++i) {
            InsertElement(prefix+CString(ce.Name())+CString("."),*i);
        }


    } else {

        CString name = prefix+CString(ce.Name());
        CString value;

        GetStringValue(ce, value);

        //
        // insert name, value pair into list control...
        //

        ...
    }
}

BOOL CContainerView::OnInitDialog()
{
    ... // insert member and view columns into list control

    m_Container->StructMembers(members);
    for (i = members.begin(); i != members.end(); ++i) {
        InsertElement(CString(), *i);
    }

    ... // other initialization...

    return TRUE;
}
```

Figure 87. Traversing the Input Container

12.12 Handling FlowMark Error Conditions with C++ Exceptions

Throughout the implementation of our run-time client, we use the idiom of testing the return code from a FlowMark method invocation against expected return codes such as EXM_API_OK and, if there is no match, throwing an instance of the CFlowmarkException class.

For diagnostic purposes, the constructor of the CFlowmarkException class forces the creator of the exception to record the class and method names of the failing FlowMark method, a string of indicative parameter data, and the return code received.

```
throw new CFlowmarkException(
            "ExmContainerElement",
            "GetValue",
            ce.Name(),
            rc);
```

Figure 88. Throwing a C++ Exception

The default response of the MFC application framework to an exception that is not caught by intervening layers is to build and display a modal message box containing a description of the exception. Dismissing the message box usually leaves the application in a state that is similar to the state the application was in before the user initiated the action that led to the reported error condition.

The message box displayed by the MFC framework is populated with a string returned by the virtual GetErrorMessage() method of the CException class. In our case, our CFlowmarkException class provides an implementation of this method that simply formats the diagnostic data collected by the exception's constructor. A more sophisticated implementation looks up a table of strings using the return code as a key and displays an informative message that explains the meaning of the return code.

```
BOOL CFlowmarkException::GetErrorMessage(
        LPTSTR   lpszError,
          UINT   nMaxError,
         PUINT )
{
    ostrstream   buffer(lpszError, nMaxError);

    buffer
       << m_ClassName
       << "::"
       << m_MethodName
       << "(" << m_Param
       << ") failed with rc = "
       << m_RC
       << (char)0;

    return (TRUE);
}
```

Figure 89. Describing a FlowMark Error Condition

12.13 Discussion

In this chapter, we have described the construction of a custom run-time client using the FlowMark C++ API and Microsoft Visual C++ 4.2. In this section, we evaluate the implementation and discuss its limitations and possible improvements

12.13.1 Using a Multiple Document Interface (MDI) Model

The run-time client that is the focus of this chapter uses the MFC Single Document Interface (SDI) model as the framework for implementing the application. If you want to create a more fully functioned run-time client, you might consider using the MFC Multiple Document Interface (MDI) framework instead. Most of the source code and resources from this example can be imported into a new MDI skeleton application with few changes, although you probably want to change how the FlowMark connection state is managed.

12.13.2 Performance Considerations

One advantage of our technique of displaying input container data along with other attributes of a work item is that we can specify, in an ad hoc manner, which container members to view. We can, in principle, use a container member as a sort key. The disadvantage of this technique is that a performance penalty is associated with retrieving a work item′s container from the server. If you are planning to use this technique widely, you must consider carefully the performance implications of doing so, particularly if you expect your work lists to be large.

12.13.3 Implementing CheckOut without CheckIn

In our run-time client, we provided access to the `ExmWorkitem::CheckOut()` method but we did not provide access to the complementary `ExmWorkItem::CheckIn()` method. This is appropriate in our scenario, because the activities for which we use the run-time client to complete do not pass data in their output containers to following activities. In the general case, this is not a valid assumption and so this limits the utility of our run-time client as a general purpose activity ″completer″.

We can enhance our run-time client to add a generic check-in facility that allows the user to type values directly into a activity′s output container. While this is in fact a useful tool for development and testing purposes, this generic solution has few uses in a production environment because of its limited ability to present helpful context information or perform any required input validation.

12.14 Source Files and Installation

Source files:

- \examples\cpprtc*.*

Installation:

- Copy \examples\cpprtc\install* into a directory of your path.

Chapter 13. Web-Enabled Activities

The use of local and wide area networks is now common in the business world. The ability to maintain centralized data and control while permitting and exploiting the abilities of local workstations has made them a permanent fixture in the office. But more and more, there is an effort to directly include those users who are outside the office; salespersons, field engineers, and telecommuters are a few examples.

One technology that has emerged as a possible solution is the browser program, that works equally well on the Internet and on an intranet. This program interprets data sent from a server and displays the results in the browser window. The server does not need to know about the client computer, so a platform-independent solution is automatically achieved.

The IBM Internet Connection for Flowmark allows you to execute, manage, and track FlowMark processes using an intranet or the Internet. Of particular interest is the ability for Web browser users to take part in a process without knowing about FlowMark or understanding workflow concepts.

The Web-enabled activity program is a generic activity program that enables a FlowMark activity to send activity data to and receive updated data back from a Web browser user. The Web-enabled activity program is the key to including mobile agents in the workflow process, and its use is the focus of this chapter.

13.1 Where This Example is Used

The examples discussed in this chapter are part of a sample FlowMark scenario described in Chapter 2, “Scenario Description” on page 7. The FlowMark implementation of this scenario is discussed in Chapter 3, “FlowMark Model Description” on page 9. The FlowMark process model details of the sample activity programs presented here are described in:

- Topic 3.4, “Manufacture Web Process” on page 13.

The Web-enabled activity program is carried out for each of the manufacturing steps in our scenario. These are within the Manufacture block of the Manufacture Sign process. An HTML page for connecting to FlowMark and a simple HTML work list are shown to start the activities. A separate sample process with sample HTML documents is supplied as part of the IBM Internet Connection for Flowmark software.

13.2 What Techniques This Example Demonstrates

This example demonstrates:

- The IBM Internet Connection for Flowmark
- A simple browser logon window
- A custom browser work list
- HTML activity program templates

13.3 Web-Enabled versus Web-Initiated Activities

The IBM Internet Connection for Flowmark allows the Web browser user to start two types of program activities, Web-enabled activities and Web-initiated activities.

Web-initiated activities are any normal FlowMark program activities that refer to a program registration that runs a program at a specific location. This specific location is the TCP/IP or APPC address of any machine running the FlowMark Program Execution Client and Telepath Server. So, when a Web browser user selects and starts a Web-initiated activity program, the program executes at the location specified by the program registration. The Web browser user does not have any interaction with the activity program. This type of activity program is not specific to a Web browser environment and is not discussed in this chapter.

Web-enabled activities are FlowMark program activities that refer to the FlowMark Common Gateway Interface (CGI) program, and that allow Web browser user interaction with the program activity. The FlowMark CGI program, also called the Web-enabled activity program, can display any HTML document. This document becomes the Web browser user's interface to the program activity. Input data container information can be displayed and output data container data can be modified, just as in a local FlowMark run-time client program activity. This type of activity program is specific to a Web browser environment and is described in detail in this chapter.

13.4 Setting Up for Web-Enabled Activities

There are a couple of requirements for using a Web browser based activity within your FlowMark process, but they are simple. The requirements are:

- Web-enabled activities should only be started from a Web browser client. They can be started from a FlowMark run-time client, but return an error. Therefore, the exit condition for all Web-enabled program activities should check the value of the predefined data member _RC. The values returned by the Web-enabled activity program are documented in the *IBM Internet Connection for Flowmark* manual.
- The activity name must start with the Web-enabled prefix defined for your installation. The default prefix is "www_".
- The activity must be defined as a manual start activity.
- The activity must refer to a program registration of the Web-enabled activity program, EXMP5WAP.EXE.

If data from an activity is to be displayed in the Web browser, you must define this data in the input container for the activity. The HTML template must include variables for the input container data to be used. The Web-enabled activity program replaces the variables in the template with the actual container data values when it sends the HTML page to the Web browser.

The Web-enabled activity program must be notified of the HTML template to be completed and sent to the Web browser. This can be accomplished by two methods. The first is to specify the template name as a parameter in the program registration. The second is to include a data member in the input container named html_template_name and to set its value to the HTML template name. This data member can be set by a previous activity program, or by

specifying a default value for the container member in the FlowMark process model.

Each method has advantages and disadvantages. The first method allows you to see in FlowMark Buildtime Client all the Web browser activity HTML programs you have defined, but they are all identical except for the parameter. The second method keeps the FlowMark Buildtime Programs folder cleaner and allows the HTML template to be specified while defining activities in the process model editor. We used the second method, with default values, in our sample scenario process.

For the program registration of the Web-enabled activity program, we specified the program to run unattended at a specific IP address, that of the Web server.

13.5 Signing On to FlowMark

The first HTML document that the FlowMark Web client accesses serves as a type of logon for the user. A key value must be entered and, if the server is protected, a valid Web user ID and Web password must be entered. The key serves to link a Web user ID to a FlowMark user ID. This is described in detail in the documentation for the IBM Internet Connection for Flowmark. The HTML source for this page is shown in Figure 90.

```
<html><! AgentPage.html>
<head>
<title>SignCo Agent Page</title>
</head>

<body>
<center><h2>SignCo Agent Login</h2></center>
<hr>
<p><h4>Enter key and press button to view work list.</h4>

<strong>Key:</strong>
<form action="/cgi-prot/exmp5cgi.exe" method="POST">
<input type="text" name="wf-fmig-key" size=64>
<br>
<br>
<hr>
<input type="submit" name="Submit" value="Display Worklist">
<input type="hidden" name="wf-cgi-submit" value="0">
<input type="hidden" name="wf-cgi-html" value="/sg242184/html/Worklist.html">
</form>
</body>
</html>
```

Figure 90. HTML for FlowMark Logon

Everything is standard HTML up until the <form> tag. Here we define the action to run a common gateway interface (CGI) program when the submit button is pressed, and to send the data from the form through the POST method. This definition for the <form> tag is common to all the HTML documents we discuss. Next, we have a Submit button defined as is usual for HTML forms. The following two lines are specific to FlowMark and are required for this logon window.

```
...
<input type="hidden" name="wf-cgi-submit" value="0">
<input type="hidden" name="wf-cgi-html" value="/sg242184/html/Worklist.html">
...
```

Figure 91. Required Form Elements

The first line defines a hidden form element to contain the value "0". The value for the variable wf-cgi-submit can range from "0" to "14", and informs the CGI program what action is to be taken. These values are defined in the IBM Internet Connection for Flowmark documentation. The second line sets a FlowMark variable to the next HTML template page to be displayed when the key is verified. Here, the next page is /sg242184/html/Worklist.html, which is described in the next section.

The FlowMark logon page is shown in Figure 92.

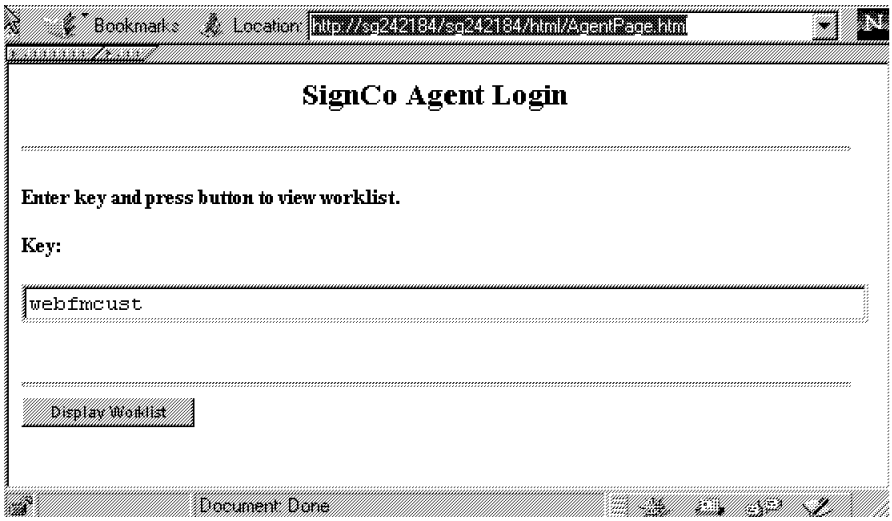


Figure 92. FlowMark Agent Logon

13.6 Displaying a Work List

The work list HTML document is a template for the FlowMark CGI program. The CGI program reads the template file, modifies it, and sends it to the Web browser. The FlowMark CGI program modifies the HTML page in three ways by:

- Replacing variables with FlowMark data.
- Expanding repeated groups of HTML.
- Adding hidden form elements to the HTML.

As we look at the HTML for the client work list, we highlight which modifications are made by FlowMark and why they are made.

First, Figure 93 on page 107 shows the <head> section of the work list HTML page.

```
<html>
<head>
<title>Custom Workflow Client for Web Users</title>

<script language="JavaScript">
<!-- hide

function Refocus()
{
   document.form1.submit.focus();
}

function VerifyInput()
{
   var ok = false;
   for(var i = 0; i < document.form1.length; ++i) {
      if (document.form1.elements[i].type == "radio") {
         if (document.form1.elements[i].checked) {
            ok = true;
            break;
         }
      }
   }
   if (!ok) {
      alert("Please select a work item to start.");
   }
   return ok;
}
// -->
</script>
</head>
```

Figure 93. Work List HTML Heading

Notice the use of two simple JavaScript functions. We have included these to show that JavaScript can be used with FlowMark Web browser pages to add functionality such as form entry verification, simple calculations, and enhanced navigation. The first function in our example, `Refocus()`, is used to move the focus of the page from the work list elements to the push buttons when a work item is selected. This may be helpful when the work list becomes large. The second function shown, `VerifyInput()`, checks that a work item has been selected when the form is submitted. If a work item has not been selected, the function alerts the user and cancels the submission of the form.

The next section shown in Figure 94 on page 108 starts the main form, `form1`, and specifies the same action and method as before. Notice that the `onSubmit()` handler is used to call our verification function.

```
...
<body>
<center><h2>SignCo Agent FlowMark Worklist</h2></center>
<hr>
<p><h3>Select a work item:</h3>

<form name="form1" action="/cgi-prot/exmp5cgi.exe" method="POST"
      onSubmit="return VerifyInput()">
...
```

Figure 94. Work List HTML Form Definition

Now we define the actual work list itself, formatted within a table structure. It begins with the appropriate headings for each column, and then displays a row with three cells. The first cell contains a radio button so that the work item can be selected.

```
...
<center>
<table border=1 cellpadding=2 cellspacing=0 width=80%>
  <tr>
    <th>Description</th>
    <th>Activity</th>
    <th>Process</th>
  </tr>
  <!--"wf-cgi-rbegin"-->
  <tr>
    <td valign=middle><input type="radio" name="wf-api-item"
                        value="wf-api-item-id" onClick="Refocus()">
        "wf-api-item-descrip" </td>
    <td>"wf-api-item-name"</td>
    <td>"wf-api-item-procinst"</td>
  </tr>
  <!--"wf-cgi-rend"-->
</table>
...
```

Figure 95. Work List HTML for Displaying Work Items

Within this code fragment, you see several items in quotes that begin with "wf-...". These are the repeat group and replacement variables that were mentioned earlier. First, let's look at the repeat group replacement variables:

```
  ...
  <!--"wf-cgi-rbegin"-->
  ...
  ...
  <!--"wf-cgi-rend"-->
  ...
```

Figure 96. FlowMark CGI Repeat Group Variables

These are formatted like HTML comments with string literals as the comment. The repeat group variables define to the CGI program which sections of the HTML should be expanded to display the multiple entries in the list. The list in our example is of work items because the contained replacement variables are

of the type ″wf-api-item-...″. The three possible types of variables within a repeat group are:

- ″wf-api-item-...″ - for work list items
- ″wf-api-proc-...″ - for process templates
- :wf-api-inst-...″ - for process instances

The replacement variables within the repeat group are replaced with literal values when the FlowMark CGI program processes the HTML template. The work list item variables specify information contained within the normal work list such as description, activity name, and so on.

Note: When examining the format of replacement variables, you notice that some are contained within HTML comments, <!-- ... -->, and some are not. When a replacement variable is contained within comments and there is no value to put in its place, nothing is displayed in the browser. When the replacement variable is not contained within HTML comments, and there is no value, the replacement variable name itself is displayed. This is similar to what occurs when %MemberName% replacement variables are used in activity descriptions to show container information.

The remainder of the HTML template defines two push buttons for the user to select the next action. The first button is pressed to start the selected work item, and the second is pressed to refresh the work list. They are presented in a table just for formatting purposes; this is not required by FlowMark. The rest of the HTML template is shown in Figure 97.

```
...
<table>
<tr>
  <td>
    <input type=″submit″ name=″submit″ value=″Start Work Item″>
    <input type=″hidden″ name=″wf-fmig-logoff″ value=″0″>
    <input type=″hidden″ name=″wf-cgi-submit″ value=″3″>
    <input type=″hidden″ name=″wf-cgi-html″
           value=″/sg242184/html/Worklist.html″>
    </form>
  </td>
  <td>
    <form name=″form2″ action=″/cgi-prot/exmp5cgi.exe″ method=″POST″>
    <input type=″submit″ name=″refresh″ value=″Refresh Worklist″>
    <input type=″hidden″ name=″wf-fmig-logoff″ value=″0″>
    <input type=″hidden″ name=″wf-cgi-submit″ value=″0″>
    <input type=″hidden″ name=″wf-cgi-html″
           value=″/sg242184/html/Worklist.html″>
    </form>
  </td>
</tr>
</table>
...
```

Figure 97. Work List HTML Push Button Actions and Form Element Values

Each push button is contained within its own form, even though the submit action and method are identical. This is required because when the form is submitted, all inputs for that form and the corresponding values are sent to the CGI program specified in the action. The values for these inputs can be different,

and we can see in this example that the value of "wf-cgi-submit" is "3" for the "Start Work Item" button and "0" for the "Refresh Worklist" button.

One thing that may be a little confusing is the value for "wf-cgi-html". We have defined it to show the current page when "Refresh Worklist" is pressed, which makes sense, but also when the "Start Work Item" button is pressed. Don't we want the work item window to display? Actually, the CGI program stores the specified template name to display it after the Web-enabled activity is completed. The page to be displayed when the work item is started is defined in the process model, either by setting the data container member html_template_name or by specifying the template name as a parameter in the program registration. On the other hand, for a Web-initiated activity discussed earlier, there is no HTML user interface and the specified HTML template is shown after the "Start Work Item" is pressed.

The FlowMark work list page is shown in Figure 98.

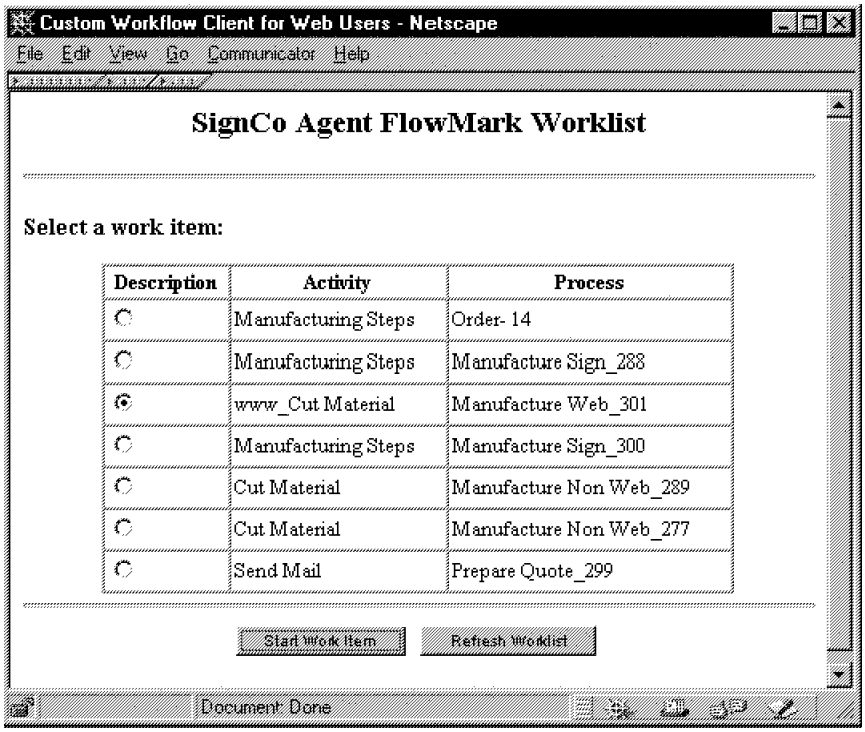


Figure 98. FlowMark Agent Work List

13.7 The Activity Program HTML Template

Generally, each Web-enabled program activity has a unique HTML template file. Within these activities, you have access to the input and output container data, just as in a non-Web-enabled activity. This sample file shows how data container information can be accessed and used within a simple JavaScript, and how output container data can be set. Again, we begin by looking at the header section of the HTML template shown in Figure 99 on page 111.

```
<html>
<head>
<title>Finish FlowMark Activity</title>

<script language="JavaScript">
<!-- hide

function SurfaceArea()
{
   var area = 0;

   area = document.formDesign.height.value    *
          document.formDesign.width.value     * 2 +
          document.formDesign.height.value    *
          document.formDesign.thickness.value * 2 +
          document.formDesign.width.value     *
          document.formDesign.thickness.value * 2;

   document.form1.area.value = area/1000000;
}

// -->
</script>
</head>
```

Figure 99. Activity HTML Heading and JavaScript

We have a simple title and one JavaScript function for calculating the surface area of an object given the height, width, and thickness. In our scenario, if the sign is to be painted or varnished, it might be helpful to know how much area is to be covered so that the correct amount of finish is ready. This is included to show how FlowMark data container information can be used within an HTML form. The function uses the values from three form elements defined later in the document in the form whose name attribute is set to formDesign. We will look at that form in detail when we get to that section of the HTML.

Next, Figure 100 shows the beginning of the body of the page and the declaration of the first form within the page:

```
...
<body onLoad="SurfaceArea()">

<center>
<h2>Surface Finishing for Sign</h2>
<hr>

<form name="form1" action="/cgi-prot/exmp5cgi.exe" method="POST">
...
```

Figure 100. Calling the JavaScript Function and Defining the Main Form

Notice that our JavaScript function is executed when the form has completed loading. Since we know the function is to use container data, we can see that all the container data is ready and available when the form is loaded. Again, the same action and method are defined for this form as for the other HTML documents we have seen.

Now it is time to show some information to the Web browser user. In this example, the sign design details relevant to the surface finishing of the sign are displayed in a table. The replacement variables that represent the input container members are named "wf-act-in-MemberName", where MemberName is replaced with the full dot notation name of the container data member. Figure 101 shows the HTML to display the sign details.

```
...
<h3>Design Details</h3>
<table border=1 cellpadding=2 cellspacing=0 width=80%>
  <tr>
    <th>Attribute</th>
    <th>Value</th>
  </tr>
  <tr>
    <td><b>Order Number</b></td>
    <td>"wf-act-in-Design.Order_ID"</td>
  </tr>
  <tr>
    <td><b>Base Finish</b></td>
    <td>"wf-act-in-Design.Base_Finish"</td>
  </tr>
  ...
  ...
  <tr>
    <td><b>Surface Area (sq m)</b></td>
    <td><input type="text" name="area" value=""></td>
  </tr>
</table>
...
```

Figure 101. Displaying FlowMark Container Information

Remember that when the template is processed by the FlowMark CGI program, the replacement variables are replaced with the literal values of the container members. The last element in the table is a text field to display the surface area of the sign, and is set by the JavaScript function previously defined.

Now all the information the user needs to complete the activity is displayed; all that remains to be done is to show whether the task was actually completed or not. As we did with the work list page, we provide push buttons for the user to press; one for the task being completed, and one for the task being canceled. This is reflected in the workflow by the value of the Finished output data container member. It is set to "YES" if the user presses complete, or "NO" if cancel is pressed. Figure 102 on page 113 shows the HTML to accomplish this; the table is not required and is only for formatting.

```
...
<table>
<tr>
  <td>
    <input type="submit" value="Task Completed">
    <input type="hidden" name="wf-cgi-submit" value="13">
    <input type="hidden" name="wf-act-out-Finished" value="YES">
    </form>
  </td>
  <td>
    <form name="form2" action="/cgi-prot/exmp5cgi.exe" method="POST">
    <input type="submit" name="cancel" value="Task Canceled">
    <input type="hidden" name="wf-cgi-submit" value="13">
    <input type="hidden" name="wf-act-out-Finished" value="NO">
    </form>
  </td>
</tr>
</table>
...
```

Figure 102. Activity HTML Push Button Actions and Form Element Values

The value for "wf-cgi-submit" is set to "13", which indicates that data is to be passed back to the activity. As in the work list example, two forms are used so that the correct data is sent to the CGI program through the POST method.

There are also some standard replacement variables that can be used in any of the HTML templates, whether they are used for displaying process templates, process instances, or work items. These are values that are valid only in the context of the CGI program, not for FlowMark in general. Two of these are displayed in Figure 103.

```
...
<p><!--"wf-cgi-datetime"--> <!--"wf-cgi-webuser"-->
...
```

Figure 103. FlowMark CGI Program Replacement Variables

Notice that since they were displayed within HTML comment tags, nothing is displayed if they cannot be replaced for some reason. These two just show the date and time the HTML template was sent to the browser, and the user ID that was provided during logon.

We have not yet seen how the input data container information was made available to the JavaScript function. One method is to create a form consisting only of hidden fields and to set the values to the data container member values. Then the JavaScript can access any value needed by the input name attribute. The definition of this invisible form is shown next in Figure 104 on page 114.

```
...
<form name="formDesign">
<input type="hidden" name="height" value="wf-act-in-Design.Base_Height">
<input type="hidden" name="width" value="wf-act-in-Design.Base_Width">
<input type="hidden" name="thickness"
       value="wf-act-in-Design.Base_Thickness">
</form>
</body>
</html>
```

Figure 104. Making Container Data Available to JavaScript

The FlowMark activity program page is shown in Figure 105.

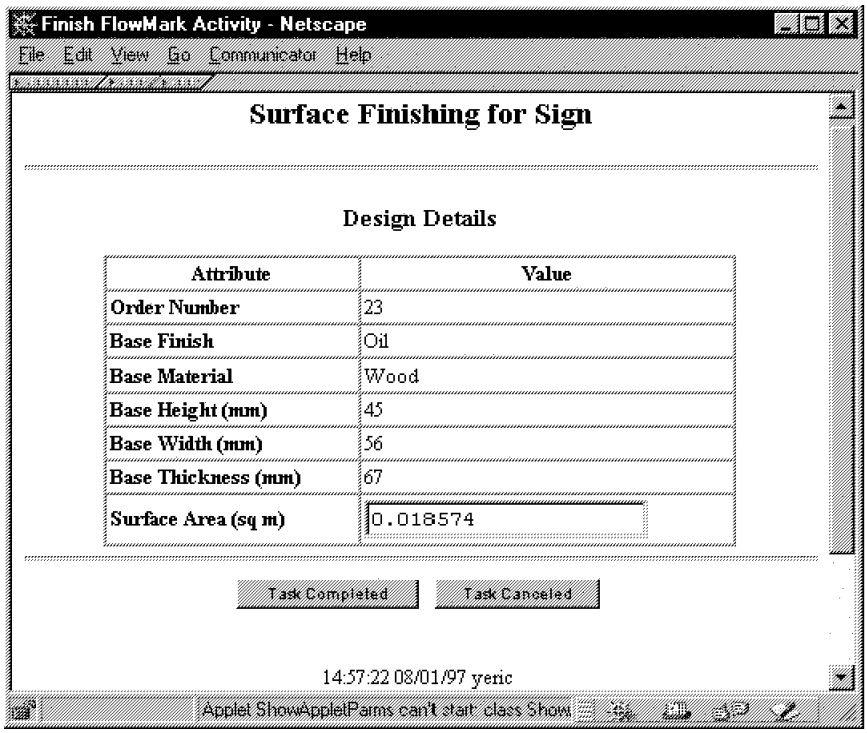


Figure 105. FlowMark Activity Web Page

13.8 Additional Information

The sample HTML documents provided show how to carry out many of the features that are available with the IBM Internet Connection for Flowmark when working with Web-enabled activities. It may be helpful to mention some aspects of using Web-enabled activity HTML templates that are not discussed as part of the sample scenario.

13.8.1 Using Input Container Data as HTML Form Input Values

There may be occasions when you need to set the value of an HTML form input element to a value contained in the input container. In our example, we used certain data members in a calculation, and made them available to JavaScript by putting them in hidden form elements shown in Figure 106 on page 115.

```
...
<input type="hidden" name="height" value="wf-act-in-Design.Base_Height">
<input type="hidden" name="width" value="wf-act-in-Design.Base_Width">
...
```

Figure 106. A Potential Replacement Variable Error

This HTML code should work fine for our example because each of these data container members is defined as `long` and should always contain a value. A problem might arise if you need to copy data from the input container to the output container and some of these members are of type `string` and contain spaces. Consider the example in Figure 107.

```
...
<input type="hidden" name="wf-act-out-Name" value="wf-act-in-Name">
...
```

Figure 107. HTML before Variable Replacement

This HTML template is processed by the Web-enabled activity program, that replaces the input container variables with data container information. Figure 108 shows a possible section of HTML sent to the Web browser.

```
...
<input type="hidden" name="wf-act-out-Name" value=George Jetson>
...
```

Figure 108. HTML after Variable Replacement

Most Web browsers do not like this format. To fix this potential error, the problematic HTML should be rewritten with two sets of quotes around the replacement variable, as shown in Figure 109.

```
...
<input type="hidden" name="wf-act-out-Name" value='"wf-act-in-Name"'>
...
```

Figure 109. Corrected HTML before Variable Replacement

This corrects the potential error because the replaced text is still within single quotes, thus setting the form value to a value that it understands.

13.8.2 JavaScript and FlowMark Input Container Data: an Alternative Method

Previously, we showed how a dummy form consisting of only hidden form input elements can be used to make data container information available to a JavaScript function. For simple functions such as the one in our example, direct use of the replacement variables may be sufficient. Our new function is similar to Figure 110 on page 116.

```
function SurfaceArea()
{
   var area = 0;

   area = "wf-act-in-Design.Base_Height"    *
          "wf-act-in-Design.Base_Width"     * 2 +
          "wf-act-in-Design.Base_Height"    *
          "wf-act-in-Design.Base_Thickness" * 2 +
          "wf-act-in-Design.Base_Width"     *
          "wf-act-in-Design.Base_Thickness" * 2;

   document.form1.area.value = area/1000000;
}
```

Figure 110. Using Direct Variable Substitution with JavaScript

This technique has the advantage of a smaller HTML file and requires less work to be done by the browser, but is not flexible. If for instance, JavaScript is to be used for form validation, you want a parameter-driven function that is more generic. Using direct variable substitution is not a good solution for that task.

13.8.3 JavaScript and FlowMark Output Container Data

So far we have seen how to access and use FlowMark input container data, but we have not seen how JavaScript can be used to set output data container members. To set the value of an output container form element, we must use the array notation method of accessing form elements in an HTML document. This is necessary because JavaScript does not accept variable names that include hyphens. That is, `wf-act-in-PhoneNo` appears to JavaScript as a subtraction expression, not a variable name.

To demonstrate how this can be done, let us take a simple task such as changing a phone number. We have a phone number member in both the input and output data containers, and the JavaScript performs some verification on the entry. When the Submit push button is pressed, our function is called. If the entry field is blank, an alert box is displayed. Otherwise, the phone number is formatted and placed back in the form element associated with the output container member. Our HTML might look like Figure 111 on page 117.

```
...
<script language="JavaScript">
<!-- hide
function VerifyPhoneNumber()
{
   if (document.form1.elements[0].value == "") {
      alert("A phone number must be entered.");
      document.form1.elements[0].focus();
   } else {
      var formattedNumber = ""
      // format phone number ...
      ...
      // set container value with formatted string
      document.form1.elements[0].value = formattedNumber;
   }
}
// -->
</script>
...
<form name="form1" action="/cgi-prot/exmp5cgi.exe" method="POST">
<input type="text" name="wf-act-out-PhoneNo" value='"wf-act-in-PhoneNo"'>
...
<input type="submit" value="Task Completed" onClick="VerifyPhoneNumber()">
...
</form>
...
```

Figure 111. Setting Output Container Data with JavaScript

This particular example assumes that the text input element for the phone number is the first input element defined for form1, therefore, it is at index 0 in the elements[ ] array of the form object. When using more complex or multiple forms, keeping track of the index of each form elements can become tedious. Also, the FlowMark CGI program adds some hidden elements to the HTML document when it is sent to the Web browser. These hidden elements are appended at the end of the form, so they do not affect us in our example. However, care should be taken when using this method, and the HTML documents should be viewed from the Web browser to verify where the hidden elements are being placed in relation to the elements you may be accessing through JavaScript.

13.8.4 Java Applets and FlowMark Container Data

Using Java applets with your HTML templates can greatly increase the functionality available to your system's Web browser clients. Complex graphical user interfaces, animation, and database access through JDBC can be used to create applications for the Web browser that are similar to local applications. We do not describe Java applet programming here, but we discuss how FlowMark container data can be used by Java applets. For an applet to access input container data, we use the same replacement variables that we used to set form input elements. The only difference is that the variables are used within the applet tag. This is shown in Figure 112 on page 118.

```
...
<applet codebase="/sg242184/java/"
        code="SomeApplet.class"
        width=400 height=200>
<param name=height value="wf-act-in-Design.Base_Height">
<param name=width  value="wf-act-in-Design.Base_Width">
<param name=finish value='"wf-act-in-Design.Base_Finish"'>
</applet>
...
```

Figure 112. Applet HTML Tag with Container Data as Parameters

In this example, the Java applet has access to the three parameters shown. Notice that the third parameter value, which we know is a string type, is enclosed in single and double quotes. As described earlier, this is necessary because after variable replacement, the value may contain spaces. These parameters are accessed within the applet by using the `getParameter()` function. Once the data is retrieved, any processing can be done. We can, for example, use a Java applet instead of a JavaScript function in our sample template to calculate and display the surface area of the sign.

What about setting output container data members? As of this writing, a way to accomplish this does not exist. To set the values of output members, we need to POST the submit value, data member values, the key value, and the handle back to the FlowMark CGI program. It is possible to simulate a CGI POST from within a Java applet, but we do not have access to the key and handle values. Unlike the input container replacement variables, `"wf-fmig-key"` and `"wf-fmig-handle"` are only appended at the end of the form tag; they are not replaceable values.

13.9 FlowMark Definitions

This example uses the following FlowMark definitions:

Processes

- Manufacture Web

Programs

- WebEnabledActivity

Data Structures

- Design Web
- Step Status

13.10 Source Files and Installation

Source files:

- \examples\webact\AgentPage.html
- \examples\webact\WorkList.html
- \examples\webact\CutMaterial.html
- \examples\webact\EngraveText.html
- \examples\webact\Filling.html

- \examples\webact\Finish.html

Installing the HTML templates:

- Copy all HTML files to the appropriate HTML directory of your Web server.

Chapter 14. The Redbook Utility ActiveX Component

One aspect of a FlowMark implementation that becomes evident early is the number of applications that must be created or modified for just one process. Each step in the process is generally a stand-alone application that uses container data and external information to complete the task. Each program activity may also be required to interact with other applications, whether on the local machine or on a host computer.

Many times, the systems to be used are common throughout the flow although the programs and data required may be different. The availability of reusable code components can make the FlowMark application development task much easier on the programmer. Of course, there are many other advantages to object-oriented programming, but the benefit of code reuse is particularly important for our tasks.

There are several methods and strategies for component development, but the most exciting today are JavaBeans and ActiveX. FlowMark does not yet have a Java application programming interface, so we chose to do a simple utility component with ActiveX.

14.1 Where This Example is Used

The examples discussed in this chapter are part of a sample FlowMark scenario described in Chapter 2, “Scenario Description” on page 7. The FlowMark implementation of this scenario is discussed in Chapter 3, “FlowMark Model Description” on page 9. The FlowMark process model details of the sample activity programs presented here are described in:

- Topic 3.1.1, “Calculate Price Activity” on page 10

This utility component is used in the Calculate Price activity program and in the Microsoft Visual Basic process starter program. In the Calculate Price activity program, the component′s functionality is used to retrieve information from the Windows registry of the local machine, to add data to a spreadsheet application, to retrieve a calculated value from the same spreadsheet, and to update information in an ODBC database. The process starter application also accesses the same spreadsheet to generate a price for the customer.

As an ActiveX component, the utility code can be used from any development environment that supports ActiveX. If we expand our process in the future, or have a related process, we can use tools such as Visual C++, Powerbuilder, Delphi, Visual J++, or IBM VisualAge for Basic.

14.2 What Techniques This Example Demonstrates

This example demonstrates:

- Object-oriented development with Microsoft Visual Basic
- ActiveX component development
- OLE automation of Microsoft Excel
- ODBC database access

- Using the Windows registry
- Component use by a FlowMark activity program

14.3 What the Utility Component Provides

The redbook utility component provides some encapsulated functionality for actions that are common within our process scenario. The component can be expanded to encapsulate other common actions such as representing the database objects, handling all database interactions, and automating the other office applications used in the demo. The classes defined in the utility component are simple, and serve as a good introduction to the design of reusable components.

The classes defined within the utility component are RBRegistry and RBExcel.

14.4 The RBRegistry Class

There are several configuration values used by the applications in our scenario. This helps to prevent too much hard-coding of information, and makes it easier for you to get the samples running correctly. A good place for storing configuration values is the Windows registry, which has a well-documented structure and stores all information in a common location. This is an improvement over the old .INI file technique, where dozens of files are spread across the local machine.

To use the Windows registry, we must use the Windows API, which can get tedious and complex. A class that provides a simple interface to these APIs,tailored to our applications, is helpful. RBRegistry is intended to be such a class.

14.4.1 RBRegistry Attributes

Creating a class in Microsoft Visual Basic is easy. We will not go into the details of setting up the project files, but we will review the code to describe the functionality. First, we need a place to store the registry information upon retrieval; these are the private data members of our class. In Microsoft Visual Basic, private data members are defined as shown in Figure 113.

```
Option Explicit
Private m_strODBCUserName As String
Private m_strODBCUserPassword As String
Private m_strODBCDataSource As String
Private m_strExcelFile As String
Private m_strExcelSheet As String
Private m_strDesignTable As String
Private m_strOrderTable As String
Private m_strAddressTable As String
Private m_strShipmentTable As String
Private m_strCustomerTable As String
Private m_strStringTable As String
```

Figure 113. RBRegistry Private Data Members

Private data members are variables that can be accessed only by the methods within the class. We do not want applications that make use of this class to be able to change these values, since they no longer represent the values in the registry. Thus we define them as private. To use these values, we create functions that return only the value of the variables. These functions create attributes in Microsoft Visual Basic, and are defined as shown in Figure 114.

```
...
Public Property Get ODBCDataSource()
    ODBCDataSource = m_strODBCDataSource
End Property

Public Property Get ExcelFile()
    ExcelFile = m_strExcelFile
End Property
...
```

Figure 114. RBRegistry Read-Only Attributes

The code fragment in Figure 114 defines functions to return the values of private data members and defines two public attributes. The application accesses the value of the attribute ODBCDataSource and receives the value stored in the private data member m_strODBCDataSource.

Now we have seen where the data is stored within the application and how the application can access it. Next we show how the information is retrieved from the Windows registry.

14.4.2 RBRegistry Methods

The RBRegistry class has one public method that can be called from any application with an instantiated object of the class. This method is used for setting the individual registry entries. SetValue() stores a value for the specified entry. Figure 115 shows the code for SetValue().

```
Public Function SetValue(strName As String, _
                         strValue As String) As Long
    SetValue = AddRegKeyStr(strName, strValue)
    Call Class_Initialize
End Function
```

Figure 115. Setting a Registry Value with RBRegistry

Notice that this function takes two arguments of type String, and returns a value of type Long. Looking at this code fragment does not really show what is happening, since we are just passing the arguments to another function. The called function, AddRegKeyStr(), actually does all the work by using the Windows APIs, and is shown in Figure 116 on page 124.

```
Public Function AddRegKeyStr(keyName As String, keyValue As String) As Long
    Dim rc As Long
    Dim hRegKey As Long
    Dim lngReserved As Long

    lngReserved = 0
    rc = RegCreateKey(HKEY_LOCAL_MACHINE, _
                      "SOFTWARE\IBM\FlowMark\RedbookSG242184", _
                      hRegKey)
    rc = RegSetValueStr(hRegKey, keyName, lngReserved, REG_SZ, _
                        keyValue, Len(keyValue))
    rc = RegCloseKey(hRegKey)

    AddRegKeyStr = rc
End Function
```

Figure 116. Setting a Value with the Windows API

Even though this method is declared as Public, it is actually contained within a module file used by the component, and is not exposed as part of the public interface of the class. It cannot be called directly. The call to RegCreateKey() creates the entry if it does not exist; otherwise, it opens the existing entry. We have specified the base location of the entries in a typical place for software configuration items. Therefore, objects of this class can only set values that should be relevant to the redbook sample programs.

Two private methods are also defined as part of the RBRegistry class, GetValue() and Class_Initialize(). The first method is related to the public method just described, and uses the same Windows API format. Figure 117 shows the definition of the GetValue() method.

```
Private Function GetValue(strValue As String) As String
    GetValue = ReadRegKeyStr(HKEY_LOCAL_MACHINE, _
                             "SOFTWARE\IBM\FlowMark\RedbookSG242184", _
                             strValue)
End Function
```

Figure 117. Retrieving a Registry Value

Again, we are wrapping a call to another function, this time to simplify the internal interface to the function. We only want to retrieve configuration values for our application, so the location is preset within the call. Only the name of the registry entry is required to return its value.

The second private class method is Class_Initialize, which is analogous to a constructor in the CC++ programming language. This subroutine is called when an object of the class is instantiated, and is used to initialize any data members of the class instance. The RBRegistry class uses this method to load all the configuration values from the registry if they already exist. This is shown in Figure 118 on page 125.

```
Private Sub Class_Initialize()
    ...
    m_strODBCDataSource = GetValue("ODBCDataSource")
    m_strExcelFile = GetValue("ExcelFile")
    m_strExcelSheet = GetValue("ExcelSheet")
    m_strDesignTable = GetValue("DesignTable")
    ...
End Sub
```

Figure 118. RBRegistry Class_Initialize() Subroutine

14.5 The RBExcel Class

In our sample scenario, more than one activity needs to access information and logic from an Excel spreadsheet. Therefore, common functionality required by the activity programs is made available by the utility component. The RBExcel class does not contain any public attributes, but makes available four functions to assist in integration with Excel. The four public methods provide these capabilities:

- Starting Microsoft Excel and adding data to the spreadsheet elements.
- Retrieving calculated data from the spreadsheet.
- Quitting Excel and saving or discarding the changes.
- Updating the order table with the new calculated price.

These four public methods, and the supporting private methods and data are described in the following sections.

14.5.1 RBExcel Data Members

The private data members of this class are used to maintain connection, configuration, and state information necessary for communication from our applications to Microsoft Excel and an ODBC database. Several classes are used to make our job easier. These are:

- Workspace - for local storage of database information
- Connection - for communication to an ODBC database
- Excel - for automating Excel through OLE
- RBRegistry - to retrieve configuration values

The Class_Initialize() subroutine is also used for this class to retrieve the configuration information and to set two state variables, as shown in Figure 119.

```
Private Sub Class_Initialize()
    Set m_rbr = New RBRegistry
    m_blnExcelActive = False
    m_blnConnected = False
End Sub
```

Figure 119. RBExcel Class_Initialize() Subroutine

Remember, by instantiating an object of the registry class, all the configuration values are read from the registry and stored as attributes of the object.

14.5.2 Starting Excel and Adding Data

In our scenario, the application program that uses this utility class already has the information needed to calculate the customer price. The problem is that the formulas are stored in Excel, and the calculation spreadsheet may also be used for reports and other purposes. To use these formulas, we need to start the spreadsheet program and insert our data for the formulas. The public subroutine AddOrder() handles these tasks for us. Invoking the method with the data as parameters is all that needs to be done. Let's step through this subroutine and describe the process required. First, Figure 120 shows the declaration of the subroutine with the required arguments and local variables.

```
Public Sub AddOrder(intOrderNum As Integer, _
                    datOrderDate As Date, _
                    intBaseHeight As Double, _
                    intBaseWidth As Double, _
                    intBaseThickness As Double, _
                    strBaseMaterial As String, _
                    strBaseFinish As String, _
                    intTextHeight As Double, _
                    intTextLength As Double, _
                    strTextColor As String)

    Dim lastRow As Integer
    Dim strRange1 As String
    Dim strRange2 As String
    ...
```

Figure 120. AddOrder() Subroutine Declaration

The local variables are used to maintain the position of the last row in the spreadsheet, and to specify the range of elements to be copied, which is shown later. The initial action of this routine is to load Excel and open the correct spreadsheet. This is accomplished with a call to the private subroutine LoadExcelSheet(), shown in Figure 121.

```
Private Sub LoadExcelSheet()
    If Not m_blnExcelActive Then
        'Set workbook and worksheet objects
        Set m_xlBook = GetObject(m_rbr.ExcelFile)
        Set m_xlSheet = m_xlBook.Worksheets(m_rbr.ExcelSheet)
        m_blnExcelActive = True
    End If
End Sub
```

Figure 121. Loading an Excel Worksheet

If Excel has not already been loaded, the specified workbook file is loaded, which automatically launches Excel. Next, the worksheet is specified, which becomes the active worksheet in the spreadsheet. The state variable is then set to reflect that Excel has been loaded.

The next step in adding the order to the spreadsheet is to find the last row of the spreadsheet, insert a row before it, and copy the last row back into the new second to last row. This may sound a little confusing at first, but realize that the elements contain formulas we want to maintain. By copying an existing row, we maintain the formulas within the cells. The code to perform this copy is displayed in Figure 122.

```
    ...
    'Get and update the last used row number in the spreadsheet
    lastRow = m_xlSheet.Range("$R$2").Value
    lastRow = lastRow + 1
    m_xlSheet.Range("$R$2").Value = lastRow

    'Set up the copy from and to range strings
    strRange1 = "A" & Format(lastRow - 1) & ":O" & Format(lastRow - 1)
    strRange2 = "A" & Format(lastRow) & ":O" & Format(lastRow)

    With m_xlSheet
        'Insert a blank row for the new data
        .Rows(lastRow - 1).Insert shift:=xlShiftDown

        'Copy an existing row into the new row to retain formulas
        .Range(strRange2).Copy Destination:=.Range(strRange1)
        ...
    End With
    ...
```

Figure 122. Inserting a Row and Copying a Range in Excel

Finally, we add our data to the newly added last row by setting the values of specific elements with our data. Figure 123 shows how this is accomplished.

```
    ...
    With m_xlSheet
        ...
        'Update the entries with the current data
        .Range("A" & Format(lastRow)).Value = intOrderNum
        .Range("B" & Format(lastRow)).Value = datOrderDate
        .Range("C" & Format(lastRow)).Value = intBaseHeight
        .Range("D" & Format(lastRow)).Value = intBaseWidth
        .Range("E" & Format(lastRow)).Value = intBaseThickness
        .Range("F" & Format(lastRow)).Value = intTextHeight
        .Range("G" & Format(lastRow)).Value = intTextLength
        .Range("H" & Format(lastRow)).Value = strBaseMaterial
        .Range("J" & Format(lastRow)).Value = strTextColor
        .Range("L" & Format(lastRow)).Value = strBaseFinish
    End With
End Sub
```

Figure 123. Updating Cell Values

14.5.3 Retrieving Data from Excel

As the design information is inserted into the spreadsheet, the customer price is calculated by the formula of the customer price cell. All we need to do is access the value of the particular cell containing the price and return it to the caller. To make the function more general, and more flexible for use by other applications, the order number is required as an argument, and the correct price for the specified order is returned. The declaration and initialization of the function is shown in Figure 124.

```
Public Function GetPrice(intOrderNum As Integer) As Currency
    Dim intRow As Integer
    Dim intRowOrder As Integer
    Dim varCellData As Variant

    'Load Excel and our worksheet
    Call LoadExcelSheet
    ...
```

Figure 124. GetPrice() Function Declaration

Notice that this function begins with a familiar call to the LoadExcelSheet() subroutine to start Excel and load the worksheet if needed.

Next, we must search through the rows of the spreadsheet to find the one containing the customer price for the specified order number. The code used to search through the spreadsheet is shown in Figure 125.

```
    ...
    intRow = 5
    Do
        intRow = intRow + 1
        varCellData = m_xlSheet.Cells(intRow, 1).Value
        If IsNumeric(varCellData) And Not IsEmpty(varCellData) Then
            intRowOrder = CInt(varCellData)
        Else
            'Order number not found
            GetPrice = 0
            Exit Function
        End If
    Loop Until (intRowOrder = intOrderNum) Or (intRowOrder = 0)
    ...
```

Figure 125. Searching Cells in Excel

For each row that contains order data, the value is retrieved into a local variable of type variant. We used a variant type because we do not know what kind of data is in the element. We expect that it is a number, but the user may enter comments or other text in the spreadsheet. The conditional expression checks if the value represents a numeric value and that it is not empty. This is necessary because an empty cell value with number formatting represents the value 0, which is numeric. The logic of the loop is that the loop ends when the current row order number is equal to the target order number, or when a non-numeric or zero value is found. Notice the use of the Cells and Value attributes to access data in the spreadsheet.

A final check is made to see if the loop terminated because of a zero value and a price is returned to the caller. This is shown in Figure 126 on page 129.

```
    ...
    If intRowOrder > 0 Then
        GetPrice = m_xlSheet.Range("$O$" & CStr(intRow)).Value
    Else
        'Order number not found
        GetPrice = 0
    End If
End Function
```

Figure 126. Retrieving an Excel Cell Value

14.5.4 Saving and Discarding Changes

When the application is finished using the Excel spreadsheet, the changes should be saved or discarded, and Excel should be unloaded. This is accomplished by the subroutine displayed in Figure 127.

```
Public Sub Quit(Optional blnSave As Boolean = True)
    If m_blnExcelActive Then
        If blnSave Then
            m_xlBook.Save
        End If
        m_xlBook.Saved = True  'prevents popup if not saved
        m_xlBook.Application.Quit
        m_blnExcelActive = False
    End If
    Set m_xlSheet = Nothing
    Set m_xlBook = Nothing
End Sub
```

Figure 127. Saving Changes and Ending Excel

Notice that the single argument, `blnSave`, is defined as optional and is set to a default value. That indicates that this function can be called without specifying an argument, and that so doing results in a default value of True for the argument. First, the subroutine checks to see if Excel is even active for this object. If so, the workbook is saved if the argument is true. Next the attribute `Saved` is set to True; this prevents a dialog box from appearing that asks the user about saving the workbook. Finally, the `Quit()` method is called to unload Excel, and our state variable is set to False to show Excel is not loaded. The last two lines destroy the Excel objects that are no longer required.

14.5.5 Updating an ODBC Database

The last public method of the `RBExcel` class is used to update the customer price in the order table in our ODBC database. This subroutine is declared with one required argument, the order number, and one optional argument, the customer price.

```
Public Sub UpdateOrderTable(intOrderNum As Integer, _
                            Optional curPrice As Currency)
    Dim rs As Recordset

    ' If price not specified as argument, get from Excel
    If IsMissing(curPrice) Then
        curPrice = GetPrice(intOrderNum)
    End If
    ...
```

Figure 128. UpdateOrderTable() Subroutine Declaration

Unlike the previous example, a default value for the optional argument cannot be specified. Therefore, if the customer price was not specified, GetPrice() is called to retrieve the price from the spreadsheet. Remember, GetPrice(), in turn, uses our other methods to load Excel if needed.

The next step is to establish a connection to the database. This is done with the call shown in Figure 129. Please view the source code supplied to see the details of this function.

```
    ...
    Call ConnectToDataSource
    ...
```

Figure 129. Connecting to the ODBC Data Source

Once we have connected to the data source, all we need to do is create a record set, edit the customer price field, and update the ODBC table. We then finish by destroying the record set object. The connection is maintained until our utility object is destroyed, permitting more than one operation on the database without reconnecting. The update code is now shown in Figure 130.

```
    ...
    'Update customer price field
    Set rs = m_cnn.OpenRecordset("select * from " & m_rbr.OrderTable & _
                                 " where ID=" & intOrderNum, dbOpenDynaset,
                                 0, dbOptimisticBatch)
    rs.Edit
    rs!BILLABLE_AMOUNT = curPrice
    rs.Update
    rs.Update dbUpdateBatch
    Set rs = Nothing
    ...
```

Figure 130. Creating and Updating a Recordset

14.5.6 Cleanup

When the utility object is no longer needed, the application can set the reference variable to Nothing to destroy the object. When this occurs, the terminate event is sent to the object and the subroutine shown in Figure 131 on page 131 is executed:

```
Private Sub Class_Terminate()
    If m_blnConnected Then
        m_cnn.Close
    End If
    Set m_rbr = Nothing
    Set m_cnn = Nothing
    Set m_ws = Nothing
    Call Quit(False)
End Sub
```

Figure 131. Cleaning Up and Destroying Objects

This closes the ODBC database connection, quits Excel, and destroys all objects used by the utility component object.

14.6 The Registry Setup Program

As previously described, the redbook utility component uses the Windows registry to store configuration values necessary for the example programs to run. These values must be set initially, and probably need to be updated as you get each part of the system working with your environment. A simple application is provided to set and update these registry values, and must be run at least once before the example programs that use the utility component are run. The user interface for this program is shown in Figure 132.

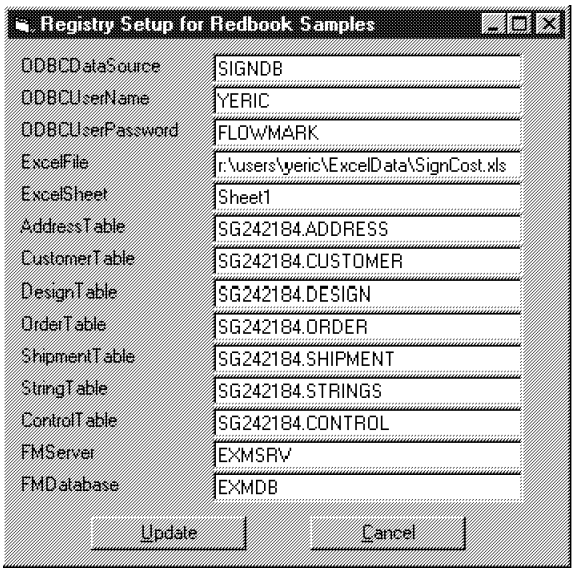


Figure 132. Registry Setup Program

14.7 Source Files and Installation

Source files:

- Redbook Utility Component
  - \examples\rbutil\RedbookUtility.vbw
  - \examples\rbutil\RedbookUtility.vbp

  - \examples\rbutil\RBRegistry.cls
  - \examples\rbutil\RBExcel.cls
  - \examples\rbutil\modReg.bas
- Registry Setup Program
  - \examples\rbsetup\RegistrySetup.vbw
  - \examples\rbsetup\RegistrySetup.vbp
  - \examples\rbsetup\frmRegSetup.frm

Installing the Redbook Utility component:

1. Run `\examples\rbutil\install\Setup.exe`.
2. Run `regsvr32 c:\winnt\system32\RedbookUtility.dll` to register the component with the system. The path shown may be different for your particular system configuration, but you can search for the DLL and use the correct path.
3. Run `\examples\rbsetup\install\RegistrySetup.exe` to configure the sample programs for your system. This saves the necessary registry entries required to run the demo programs. This setup program can be run whenever you need to update the registry values.

Appendix A. Contents of the Example CDROM

A.1 Disclaimer

We have provided you with a compact disk containing our examples, attached to the back cover of this redbook. However, we do have a disclaimer about the book and CDROM:

Disclaimer: The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

A.2 Finding an Example

The source code and installation files for all the examples described by the chapters of this book can be found in sub-directories of the \examples directory of the accompanying CDROM. Table 3 lists the chapters of the book, the directories that contain the related source code, and an indication of the execution environments they require. Where individual components have separate installation packages, these packages are located in the installation sub-directories of the respective component directories.

Table 3. Finding an Example

Chapter	Directory	OS/2	NT	Database	Web
3	fdl	√			
4	webstrtr	√			√
5	vbstrtr		√	√	
6	execsql	√		√	
7	vbexcel		√	√	
8	vbword		√	√	
9	javasmtp	√	√		
10	pbrtc		√		
11	vbrtc		√	√	
12	cpprtc		√		
13	webact		√		√
14	rbutil		√	√	
8	dispatch		√		
	nortc		√		

A.3 Using the Examples

To use any of the examples described by this book, you must first install an OS/2 server and at least one Windows NT client per the instructions in Appendix B, “Setting Up the OS/2 Server and Windows NT Clients” on page 135. Installation instructions for individual examples are given at the end of each chapter in a section called ″Source Files and Installation″.

Appendix B. Setting Up the OS/2 Server and Windows NT Clients

This appendix provides an overview of the steps required to set up an OS/2 server and Windows NT clients to allow you to run the examples described in this book.

B.1 Description of OS/2 Server Used in Examples

This section describes the hardware and software installed on the OS/2 server that was used to build and test the examples described in this book. Alternative configurations are possible. Note that in a production system, you may distribute the software servers over multiple hardware servers rather than using just one hardware server for all software servers.

B.1.1 Hardware Description

The server used to build and test the examples described in this box was configured as follows:

- IBM Personal Computer 750 (6887-XAC)
- 48MB RAM
- 1.2GB SCSI HDD
  - OS/2 Boot Manager - 7MB
  - C: - 258MB (HPFS, OS/2 BOOT)
  - D: - 949MB (HPFS, OS/2 DATA)
- SCSI CD-ROM
- IBM Streamer Family Adapter (Token Ring)

B.1.2 Software Description

Table 4. Software Description - OS/2 Server

Software	Notes
IBM OS/2 Warp Version 4.0	• No Service Applied • TCP/IP Services • File and Print Sharing Services
IBM Flowmark for OS/2 Version 2.3	• Fix Pack 2 applied. Fix Packs available from: ftp://ftp.software.ibm.com/ps/products/flowmark/fixes/
IBM Database 2 for OS/2 Version 4.0	• Fix Packs US8122, US8132 applied. Fix Packs available from: http://www.software.ibm.com/data/db2/db2tech/dbserveros2v4.html
IBM Internet Connection Secure Server v4.1	• No service applied
IBM Internet Connection for FlowMark	• Available from: http://www.software.ibm.com/ad/flowmark/exmn0b21.htm.
IBM Net.Data	• No service applied. For more information, visit: http://www.software.ibm.com/data/net.data/

B.1.3 Steps Required to Install and Configure OS/2 Server

1. Install IBM OS/2 Warp Version 4.0.
2. Install IBM Flowmark for OS/2 Version 2.3 and apply Fix Pack 2.
3. Import the FlowMark process model:

   Refer to Chapter 3, “FlowMark Model Description” on page 9 for information about importing the Flowmark process model.
4. Set up the example database:

   Refer to Appendix C, “Setting Up the Example Database” on page 139 for information about setting up the example database.
5. Set up the Web server:

   Refer to Appendix D, “Setting Up the Web Server” on page 143 for information about setting up the web server.
6. Set up sendmail:

   This step is required only if you want to use the OS/2 server as an SMTP server for testing the examples described in Chapter 9, “Sending E-mail with Java” on page 63. Otherwise, you can use another SMTP server instead.

B.2 Description of Windows NT Clients Used in Examples

This section describes the hardware and software installed on the NT clients that were used to build and test the examples described in this book. Alternative configurations are possible.

B.2.1 Hardware Description

The clients used to build and test the examples described in this book were configured as follows:

- IBM ThinkPad 760 EL
- 40MB RAM
- 2GB SCSI HDD
  - OS/2 Boot Manager - 1MB
  - 541MB (NTFS, WinNT BOOT)
  - 1465MB (HPFS, OS/2)
- CD-ROM
- IBM Auto 16/4 PCMCIA token-ring adapter

B.2.2 Software Description

Table 5. Software Description - Windows NT Clients

Software	Notes
Windows NT Version 4.0	• Service Pack 2 applied.
IBM Flowmark for Windows NT Version 2.3	• Fix Pack 2 applied. Fix Packs available from: ftp://ftp.software.ibm.com/ps/products/flowmark/fixes/
IBM Database CAE V2.1.1 for Windows 95 and NT	• Fix Packs US8116 and US8126 applied. See http://www.software.ibm.com/data/db2/db2tech/db2nt95v21.html
IBM FlowMark ActiveX Toolkit	• To get the ActiveX Toolkit contact the FlowMark team: http://www.software.ibm.com/ad/flowmark

B.2.3 Steps Required to Install and Configure Windows NT Clients

This section describes the steps performed to install and configure the Windows NT clients.

1. Install Windows NT Workstation V4.0 and apply Service Pack 2.
2. Install IBM FlowMark for Windows NT V2.3 and apply Fix Pack 2.
3. Set up the DB2 ODBC client for Windows NT:

   Refer to C.5, “Setting Up a Windows NT ODBC Client” on page 141 for information about setting up the DB2 ODBC client.
4. Install the IBM FlowMark ActiveX Toolkit
5. Install the Redbook Utility component:

   Refer to Chapter 14, “The Redbook Utility ActiveX Component” on page 121 for information about setting up the Redbook Utility component.

Appendix C. Setting Up the Example Database

This appendix describes the customer database used for the examples in this book and provides an overview of the steps required to set it up in an OS/2 database server and connect to it with a Windows NT client.

C.1 Description of the Example Database

The database is used to store details of the orders that are processed by the sign company. Table 6 describes the purpose and contents of each table.

Table 6. Summary of Database Schema

Table	Description
ORDER	Represents an order for the manufacture and delivery of a sign to a customer. The attributes of an order describe its price, production status, and the related FlowMark process. The foreign keys of an order relate the order to its customer, design, and shipment details.
CUSTOMER	Represents the contact details of a customer. The attributes of a customer detail the customer′s name, e-mail address, and phone number. The foreign keys of a customer relate the customer to an address representing the customer′s billing address.
DESIGN	Represents the design of a sign. The attributes of a design describe the size, material, and finish of the sign′s base material, and the size, color, and contents of the sign′s text.
ADDRESS	Represents an address. The attributes of an address describe its street address, locality, region, country, and postal code.
SHIPMENT	Represents a shipment of an order to an address. The attributes of a shipment describe its delivery instructions and status. The foreign keys of a shipment relate the shipment to an address representing shipment′s delivery address.
STRINGS	Contains strings that describe, with text, the meaning of small integers used elsewhere in the database. Strings of class 1 describe the small integers used as class identifiers in the STRINGS table itself.
CONTROL	Contains the next identifiers to be used for each of the ORDER, DESIGN, ADDRESS, and CUSTOMER tables. DB2 triggers are used to increment these identifiers after every successful insert. Clients that want to insert a new row into one of these tables should first issue a select from the CONTROL table to obtain the identifiers to be used, issue the required inserts, and then commit the entire transaction.
Note: The tables named here are fully qualified by the name SG242184.	

C.2 Setting Up the OS/2 Database Server

If you have not already done so, you must install and configure a database server to contain the example database. These instructions provide an overview of the steps required to install the DB2 on an OS/2 server using TCP/IP as the communications protocol between the database client and server. Refer to DB2 for OS/2 Installation and Operation Guide, S20H-4785 for more detailed instructions about installing and configuring an OS/2 DB2 server.

- Install IBM DB2 Server for OS/2 V4.0.
- Edit CONFIG.SYS to include a DB2COMM statement:

  ```
  SET DB2COMM=TCPIP
  ```

- Edit the %ETC%\services file to include the DB2 service names:[6]

  ```
  db2inst1c  3700/tcp  # DB2 main connection
  db2inst1i  3701/tcp  # DB2 interrupt connection
  ```

- Update the DB2 database manager configuration

  ```
  DB2 UPDATE DATABASE MANAGER CONFIGURATION svcename USING db2inst1c
  ```

- Stop and restart DB2

  ```
  DB2 STOP DATABASE MANAGER
  DB2 START DATABASE MANAGER
  ```

C.3 Creating the Example Database

To create a DB2 database to contain the example tables, open an OS/2 window at the DB2 server and execute the following commands:

```
DB2 CREATE DATABASE SIGNDB ON D:
```

where D: is the drive where you want to install the example database.

C.4 Loading the Example Database Schema and Sample Data

The schema and sample data for the database is included in files of SQL statements in the \examples\setupdb directory. To load the schema and sample data into your database, open an OS/2 window at the DB2 server and execute the following commands:

```
cd \examples\setupdb
runsql SIGNDB drop.sql     /* only required to cleanup previous loads */
runsql SIGNDB schema.sql
runsql SIGNDB data.sql
```

[6] The service names (db2inst1c, db2inst1i) and the port numbers (3700, 3701) used here are arbitrary but must be used consistently in all following configuration steps for servers and clients. The values chosen here are identical to the examples included in the DB2 documentation.

C.5 Setting Up a Windows NT ODBC Client

To enable access between a Windows NT client and the OS/2 database server, you must install and configure the DB2 ODBC drivers for Windows NT. These instructions provide an overview of the steps required to do this. For more detailed instructions, refer to DB2 for Windows NT V2.1 Installation and Operation Guide, S33H-0312.

- Configure TCP/IP services file:

```
# ...
# extract of services file, typically located at
#    \WINNT\SYSTEM32\DRIVERS\ETC\SERVICES
db2inst1c   3700/tcp   # db2 main connection
db2inst1i   3701/tcp   # db2 interrupt connection
# ...
```

- Install IBM DB2 Client Application Enable for Windows NT and 95 V2.1.1.

  Note: Do not attempt to perform any database configuration until after you have completed installing the requisite service packs as described in the following step.

- Apply IBM DB2 CAE service packs US8116 and US8126:

  These service packs can be obtained from the following URL:
  http://www.software.ibm.com/data/db2/db2tech/db2nt95v21.html

- Configure a TCP/IP connection to the example database:

  Use the DB2 Client Setup tool to configure a connection to the example database. During this process you are prompted for several values. The following table provides hints about the values to use when you are prompted.

Parameter	Hint
Server Name	**DBSERVER** A name used to represent the database server. This name is used in two places: first, when creating a new node; second, when cataloging a new database.
Server Description	**Sign Company Example Database** An arbitrary string that helps to describe the database.
Protocol	**TCPIP**
Host name	**dbserver.somewhere.com** A valid host name or IP address of the OS/2 database server containing your database.
Service Name	**db2inst1c** The name of the TCP/IP service used by the OS/2 database server as configured during the installation of your OS/2 database server.
Database	**SIGNDB** The name of the database as it is known at the OS/2 database server.
Alias	**SIGNDB** The name of the database as it is known at your DB2 client.

- Configure an ODBC database source:
  - Open the Windows NT control panel.

- Open the ODBC icon.
- Select the **System DSN** page.
- Select **Add...**.
- Select **IBM DB2 Data Source**.
- Select **SIGNDB**.
- Close the notebook.

Appendix D. Setting Up the Web Server

To use the examples described in Chapter 4, “Starting a FlowMark Process from the Web” on page 17 and Chapter 13, “Web-Enabled Activities” on page 103 you must install and configure a web server, IBM Internet Connection for FlowMark and IBM Net.Data. It is not necessary to use IBM Internet Connection Secure Server v4.1 as your web server. However, the instructions described by this appendix assume that you are using this software. If you are using a different web server, you can use these instructions as a guide to the steps you must take to configure your web server appropriately.

D.1 Installing and Configuring Software for the Web Server

- Install IBM Internet Connection Secure Server for OS/2 v4.1

  After installing the web server, edit the server configuration file and insert the following line:

  ```
  Pass /sg242184/html/* d:\redbook\html\*
  ```

  before a line similar to:

  ```
  Pass /* c:\www\html\*
  ```

  where `d:\redbook\html` is the name of a directory on your web server into which the HTML source for the examples in this book will be copied.

- Create a web server password file

  Use the htadm tool provided with the web server to create a new password file to be used by the web server for authentication purposes per the example in Figure 133.

```
htadm -create c:\mptn\etc\cgiprot.pwd
```

Figure 133. Creating a Web Server Password File

- Install IBM Internet Connection for FlowMark

  This software can be obtained from:

  `http://www.software.ibm.com/ad/flowmark/exmn0b21.htm.`

  Follow the installation procedure in the accompanying documentation and test the supplied sample application. In particular, ensure that the web server configuration file contains a **Protection** directive similar to the one described by Figure 134 on page 144. This directive is used with **Protect** directives to force web clients to be authenticated when they access particular resources on the web server.

```
Protection  PROT-CGI  {
      PasswdFile   C:\MPTN\ETC\CGIPROT.PWD
      Mask         All@(*)
      PostMask     All@(*)
      PutMask      All@(*)
      GetMask      All@(*)
      AuthType     Basic
      ServerID     Private_Authorization
}
```

Figure 134. Protection Directive: PROT-CGI

- Install IBM Net.Data

  Install IBM Net.Data using that software's documentation for assistance, if required. After installing IBM Net.Data, note the directory specified by the MACRO_PATH directive in the Net.Data configuration file, db2www.ini.[7] The Net.Data macro file used by the examples described in Chapter 4, "Starting a FlowMark Process from the Web" on page 17 will be copied into this directory.

- Add Protection To /cgi-bin/db2www/*

  Add the line in Figure 135 to the web server configuration file. This line is necessary to ensure that web clients are authenticated when using the examples described in Chapter 4, "Starting a FlowMark Process from the Web" on page 17. Authentication is necessary for these examples because we need to use the web user's id to perform a query against the CUSTOMER table. If authentication is not enabled, this query will fail.

```
Protect /cgi-bin/db2www/* PROT-CGI
```

*Figure 135. Adding Protection to /cgi-bin/db2www/**

- Create passwords and keys for 3 users

  Follow the steps described by D.2, "Adding Additional Users" to add passwords and keys to the web server password file and the IBM Internet Connection for Flowmark key database for three users: ws, jmf and bj.

  Note: You should not insert rows into the database for these users because this is done for you by the database load procedure described in C.4, "Loading the Example Database Schema and Sample Data" on page 140.

D.2 Adding Additional Users

To add a user to the system, you must complete the following steps:

- Create the new web user

  Use the htadm tool to create a new user id and password in the web server password file per Figure 136 on page 145.

[7] This file is normally located in the web server's document root directory.

```
htadm -adduser c:\mptn\etc\cgiprot.pwd userid password "real name"

where:
   userid is the user ID of new web user
   password  is the password of the new web user
   "real name" is the real name of the new web user
```

Figure 136. Creating a New Web User

- Create a IBM Internet Connection for Flowmark key

  Use the exmp5adm tool to create a key for the user in the IBM Internet Connection for Flowmark key database per Figure 137. These instructions assume that the FlowMark user id, FMCUST, exists in the FlowMark database EXMDB, that there is a FlowMark runtime server called EXMSRV and that a password for FMCUST exists in the IBM Internet Connection for Flowmark key database.

```
exmp5adm -add_key userid userid FMCUST EXMSRV EXMDB

where:
   userid is the id of both the web user and the key created for that user
```

Figure 137. Creating a Key in the IBM Internet Connection for Flowmark Key Database

- Insert a new customer record into the database

  The SQL shown in Figure 138 can be used to obtain unique values for the ID columns of new CUSTOMER and ADDRESS rows.

```
SELECT NEXT_ADDRESS, NEXT_CUSTOMER
FROM SG242184.CONTROL
```

Figure 138. Obtaining Identifiers for New Customer Rows

  To insert a new customer into the database, you must use SQL similar to that shown in Figure 139 on page 146. If you intend to test the example described by Chapter 9, "Sending E-mail with Java" on page 63, you should set the EMAIL column of the customer row to be equal to a real e-mail address.

```
INSERT INTO SG242184.ADDRESS
   (ID, STREET, LOCALITY, REGION, COUNTRY, POSTCODE)
VALUES
   (<addr-id>, '999 Some Street',
    'Some Locality', 'Some Region', 1, 'Some Post Code');

INSERT INTO SG242184.CUSTOMER
   (ID, BILLTO_REF, NAME, EMAIL, PHONE, WEBUSER)
VALUES
   (<cust-id>, <addr-id>, 'Some Name',
    <email>, 'Some Phone Number', <user>);

where:
   <addr-id> is the identifier of the new address row
   <cust-id> is the identifier of the new customer row
   <email> is the e-mail address associated with the customer
   <user> is the web user id associated with the customer
```

Figure 139. Inserting a New Customer into the Database

Appendix E. Special Notices

This publication is intended to help workflow programmers and consultants produce FlowMark applications by providing sample code and programming tips. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM FlowMark. See the PUBLICATIONS section of the IBM Programming Announcement for IBM FlowMark for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

DB2®	DB2®/2
FlowMark®	IBM®
MQSeries	Net Data
OS/2®	Streamer®
ThinkPad®	VisualAge®

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Lotus Notes is a trademark of Lotus Development Corporation.

Microsoft, Windows, Windows NT, ActiveX, Excel, Visual Basic, Word, Office, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix F. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

F.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 151.

- *Integrating VisualAge Generator and FlowMark*, SG24-4239
- *FlowMark V2.2 Design Guidelines*, SG24-4613
- *FlowMark Installation and Administration*, SG24-4614
- *FlowMark and VisualInfo with Windows*, SG24-4712
- *Integrating IBM FlowMark with Lotus Notes*, SG24-4851

F.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

F.3 Other Publications

These publications are also relevant as further information sources:

- *IBM FlowMark: Programming Guide*, SH19-8240
- *IBM FlowMark: Application Integration Guide*, SH12-6267
- *IBM FlowMark: Installation and Maintenance*, SH12-6260
- *IBM FlowMark: Modeling Workflow*, SH19-8241
- *IBM FlowMark: Managing Your Workflow*, SH19-8243
- *IBM FlowMark: Diagnosis Guide*, SH19-8239

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO: type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Web Site on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications Publications Customer Support P.O. Box 29570 Raleigh, NC 27626-0570 USA	IBM Publications 144-4th Avenue, S.W. Calgary, Alberta T2P 3N5 Canada	IBM Direct Services Sortemosevej 21 DK-3450 Allerød Denmark

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Terms and Abbreviations

Note: This glossary defines terms and abbreviations used in this manual. For information about terms not defined here, refer to the respective publications as listed in Appendix F, “Related Publications” on page 149.

A

activity. A unit of work that is performed by one person in one place and at one time. An activity consumes time and resources and has a defined duration with an explicit start and end time.
In FlowMark: an activity is one of the steps that make up a process. See also *program activity* and *process activity*.

activity block. In FlowMark: a modeling construct that enables the grouping of related activities into a lower-level diagram. It also enables the modeling of loops and bundles. See also *activity bundle*.

activity bundle. In FlowMark: a type of activity block that supports multiple instances of a single program or process activity during run time. The number of instances of an activity is determined during run time by a special program activity called the planning activity. See also *planning activity*.

AIR. See *assumptions/issues/recommendations*.

AIX. Advanced Interactive Executive (UNIX)

animation. A facility for dynamically verifying workflow models. Animating a workflow model lets the user simulate the flow of work through its activities.

API. See *application program interface*.

APPC. Advanced program-to-program communication. A commonly used protocol for various network environments, such as Internet, Host communications, and LANs. FlowMark uses either APPC or TCP/IP. See also *TCP/IP*.

application development models. Process and data models required for application systems development.

application program interface (API). An interface provided by the workflow manager that enables programs to be started, processes to be controlled, and operators to work with data containers.

attribute. The characteristic or property that defines an entity, such as the attribute of a unit of information. See also *representative attribute*.

audit trail. A facility for recording events that occur when process instances are run.

automation manager. The automation manager is responsible for connection of remote clients to OLE servers.

B

bar code. Industry standard pattern of vertical lines; You can use bar codes to indicate the beginning of a new folder, the beginning of a new document, or to provide a value to be used in indexing the folder or the document.

base product. The product that provides the functionality required for the operation, for example, FlowMark, Lotus Notes. This is the product called via the Service Broker Manager.

benchmarking. Comparing something with a standard, like comparing the performance of a business process with another process of the same kind.

best of breed (BOB). A company that offers the same or similar products and services as other companies in that category, but at higher levels of performance in one or more area of competition (for example, price, quality, competence, customer service, and so on).

block. See *activity block*.

bottom-up. Starting the modeling or design of business processes from their lowest level of abstraction and detail and then integrating lower-level models or designs into a higher-level whole. See also *top-down*.

BPM. See *business process management* and *business process modeler*.

BPR. See *business process re-engineering*.

build time. In FlowMark: the component used to define processes. See also *run time*.

bundle. In FlowMark: a type of block that supports multiple instances of a single program or process activity at run time. The number of instances of the activity is determined at run time by a special program activity called the planning activity. See also *bundle activity, pattern activity*, and *planning activity*.

bundle activity. In FlowMark: one of the multiple instances of the pattern activity created for an activity bundle during run time. The number of instances is determined by the input to the planning activity. See also *planning activity* and *pattern activity*.

bus. A term borrowed from electrical engineering (or computer design), which signifies a continuum with concrete contact (start and exit) points. In this manual, it represents a continuum of repetitive and unpredictable processes, a set of sequential data stores, or a continuum for capturing critical process quality measurement points.

business. An entity that engages in commerce. A business produces or sells goods and services, has goals, processes, and personnel.

business area. Part of an enterprise implementing a group or processes that support one aspect of furthering the mission of the enterprise.

business process. See *process*.

business process management (BPM). The ongoing management of business processes, from day-to-day operational process management to radical business process re-engineering.

business process modeler (BPM). An IBM business modeling tool that is based on IBM LOVEM. Short name: ProModeler.

business process re-engineering (BPR). A disciplined approach to radically changing business processes.

C

call flow management. The automated management of telephone calls; especially applicable for call center applications, where phone calls are treated as units of work and are tracked and measured.

cardinality. In data modeling: an attribute of a relationship that describes the membership quantity. There are four types of cardinality:

1. One-to-one
2. One-to-many
3. Many-to-many
4. Many-to-one.

CASE. Computer Aided Software Engineering.

child organization. In FlowMark: an organization within the hierarchy of administrative units of an enterprise that has a parent organization. Each child organization can have one parent organization and several child organizations. The parent is one level above in the hierarchy. See also *parent organization*.

CD. compact disk

CDROM. compact disk read only memory

CGI. Common Gateway Interface (programs that provide services on the WWW)

COBOL. common business oriented language

CGI. FlowMark Common Gateway Interface program, or Web-enabled activity program. It can display an HTML document.

computer data store. A business object representing computer files, databases, or other media that store information.

condition. In FlowMark: an expression that determines the flow of control through a process instance. See also *start condition, exit condition*, and *transition condition*.

connector. An arrow drawn between two nodes in a process diagram to signify the flow of control or data. See also *data flow, information flow, material flow, control flow, control connector, data connector*, and *default connector*.

constrained. Representative of the factors that define *how* a process path or job is performed and by whom (for example, time, money, organization, and technology are constraining factors).

container. See *data container*.

control connector. In FlowMark: the graphical representation of the flow of control from one activity or block to another. See also *control flow, data connector*, and *default connector* and *transition condition*.

critical measurement point. Any point on the process path or within a job that is of critical importance, and where a measurement should be taken. Measurements can be in units of time or quantity, for example, time to answer a customer inquiry, cycle time, number of invoices per unit of time, error rate, and so on.

critical path. Taking into account all the dependencies and processing requirements for achieving a major goal or target, the critical path is that sequence of events that takes the longest time to reach the final goal.

critical success factor (CSF). A qualitative or quantitative measure that defines the quality or performance of an enterprise, a process path, or a job, such as the required skills of employees to perform a certain task. A measurable internal or external business result that has a major influence on whether an enterprise achieves its goals. CSFs can be assigned at the following levels:

- The entire industry
- The enterprise
- An organizational unit, such as a business area, department, or job
- Individuals, such as managers or employees.

CSD. Corrective Service Diskette (software updates for OS/2 programs).

CUA. Common User Access.

customer. A person or business that acquires products or services from an enterprise.

customer activity. In physical models or blueprints, it depicts the activities performed by customers. A customer activity can start or end a chain of activities.

customer expectation. A description of what a customer needs or wants. This can be in terms of products, services, or the performance of an activity or a system.

customer process. On logical models or blueprints, it depicts the actions or processes carried out by customers.

customer satisfaction. The goal of a customer-oriented business. Customer satisfaction occurs when a customer receives as much as, or more than, expected from a product or service. It is usually measured as the number of satisfied customers as a percentage of the total number of customers.

customer service encounter. See *service encounter*.

cycle time. The time it takes to complete a process path. For example, for the order fulfillment process, the cycle time is the time it takes to fulfill an order (from when a customer places an order to when the customer receives the product).

D

DAO. document architecture operations (CCITT)

DASD. Direct Access Storage Device; A device in which access time is effectively independent of the location of the data

data component. A packet of data with which a process deals.

data connector. In FlowMark: the graphical representation of the flow of data in a process diagram. See also *data flow, information flow, control connector*, and *default connector*.

data container. In FlowMark: storage for the input and output of an activity, a block, or a process. See also *input container* and *output container*.

data dependency. Data that a process requires to proceed. An *upstream data dependency* is data required from the preceding process. A *downstream data dependency* is data required by the next process.

data entity. See *entity*.

data flow. A packet of data in motion. It can consist of data groups, data entities, and data elements. A data flow identifies the data that is either generated from or required by, the customer or internal processes to which it is connected. See also *information flow*.

data flow diagram (DFD). A modeling technique for representing the static properties of business processes and data flows. Its major use is the systematic refinement of business objects.

data store. A logical set of data or a physical place where you can keep information (desks, filing cabinets, databases, and personal computer files are examples of *physical* data stores).

DB2/2. IBM Database 2 for OS/2, a relational database manager.

DDE. Dynamic data exchange. A OS/2 or MS Windows feature that enables data exchange between applications.

decomposition. The process of breaking a large entity into smaller components. For example, a process can be decomposed into sub-processes, or an activity into tasks.

department. A subdivision of an enterprise that shows reporting lines and performs one or more activities.

design point. The primary focus of a design; for example, the customer, or a workflow solution.

designing. The creative process used to plan, sketch, and model the new business processes, process paths, and jobs in order to create a set of detailed blueprints from which the new design can be implemented.

DFD. See *data flow diagram*.

DLL. Dynamic link library. A module containing a routine that is linked at load time or run time. FlowMark uses the *.DLL filetype under OS/2 as well as under MS Windows.

document. A transmission medium for, or depository of, information, such as a report or an invoice.

document flow. The flow of documents through an organizations. This can be through a document management system in digitized form or in hardcopy form.

document storage. A physical data store where hard-copy documents are stored.

downstream. Subsequent processes or activities, for example the *distribute* process is downstream from the *order* process. See also *data dependency* and *upstream*.

E

EDI. See *electronic data interchange*.

electronic data interchange (EDI). A computer-to-computer connection between two companies. The transaction flow for EDI transaction is regulated through international protocol.

e-mail. electronic mail

empower. To provide employees with the authority to make decisions.

enterprise. An organization, usually consisting of of several lines of business, whose purpose it is to perform a mission and to achieve goals by working with customers and suppliers.

enterprise process. The actions or processes performed by the enterprise.

entity. A thing or object of importance to a business about which the business wants to keep information, such as customer or product.

event flow. In process-based applications, including FlowMark, an event flow is part of the control flow. It triggers the continuation of activities that are in a wait status. See also *control flow, workflow*, and *task flow*.

exit condition. A control setting for an activity that determines when an activity is complete and control is passed to the next activity. It also determines when a process path or workflow is completed. See also *start condition, start criteria*, and *exit criteria*.

exit criteria. The conditions that determine when an activity has completed its actions. See also *start criteria*.

external organization. An organization unit, such as a government agency, a bank, or a supplier, that is part of a process path, but outside the organization of the enterprise.

F

FDL. See *FlowMark Definition Language*.

FlowMark. An IBM program product that manages and controls the execution of a process path or workflow.

FlowMark Definition Language (FDL). An external format for defining staff, programs, data structures, and workflow models in a flat file. The definitions in the FDL file can then be imported into a FlowMark database.

FRL. FlowMark Run-time Language; An external format for templates, instances,and work items in a flat file. The export utility program EXMPFREX.EXE located on the FlowMark database server exports the run-time database to a FRL file. Using the import utility program EXMPFRIM.EXE, a FRL file can be imported into a FlowMark run-time database.

fulfillment process. The sequential processes and data required to fulfill a customer order from the time of first contact to the complete satisfaction of the customer.

G

goal. The statement of an enterprise′s medium or long-range objectives or direction. A business target that is to be met within a given time. Goals can be qualitative, such as ***to become the best-of-breed***, or quantitative, such as ***to increase revenue by 20% over the next 12 months***. Goals exist at the following organizational levels:

- The enterprise
- An organizational unit, such as a business area, department, or job
- Individuals, such as managers or employees.

GUI. graphical user interface. This is a menu or screen that interacts with the end-user, providing information and collecting user input or choices.

H

hand-off. The passing of a product, information, or other materials from one department or workstation to another.

hierarchical structure diagram (HSD). A graphical modeling technique, which shows the hierarchical structure of organizations, processes, activities, goals, CSFs, and systems. Its major use is the systematic refinement of objects.

HLLAPI. High-level language application program interface. HLLAPI is used by application programs for host communication. FlowMark makes use of the HLLAPI building block under OS/2 or MS Windows.

HSD. See *hierarchical structure diagram*.

HTML. Hypertext Markup Language. A script language to format screens for web-based applications on the Internet or an intranet.

HTTP. Hypertext Transfer Protocol

I

INI. Initialization file. A place on workstations for programs to store configuration and usage data.

IP. internet protocol (ISO)

L

IBM LOVEM. An IBM business process engineering or re-engineering methodology, which can be applied at the following levels:

- Enterprise architecture
- Logical process path model or blueprint
- Physical process path model or blueprint
- Job model or blueprint.

LOB. See *line of business*.

location. A physical place where activities are performed or where information or materials are stored.

logical. The abstract or generic nature of business processes or data before any physical constraints are applied. Logical defines *what* process or data is required not *how* it is implemented. See also *unconstrained*.

logical model or blueprint. The depiction of the effectiveness of a process: *doing the right thing*. See also *logical, model*, and *blueprint*.

logical-to-physical transformation. The translation of a logical process into its physical implementation components, such as manual activities or systems functions. See also *logical transformation list*.

logical transformation list (LTL). A technique for transforming logical processes into physical implementation scenarios. For example, a logical process can be transformed into any number of manual activities, any number of systems functions, or both.

loop. A loop is an iteration of activities. There are two sets of exit criteria for a loop:

1. One set contains the criteria for exiting the loop through the normal flow when the exit conditions are met.
2. The other set contains the criteria for how often the flow can go through the loop before terminating it if the first criteria are not met. This set also has to describe where the flow continues in case of an abnormal termination.

loop connector. A symbol that points back to the starting point of a loop. The loop connector contains the short and long names of the activity, where the loop starts.

M

material. A commodity that is of value for the business process. Materials are used or worked on by an activity or system and are transported between activities and systems. See also *material flow*.

material flow. Any tangible product or document that is generated by an activity or system.

measurement. The extent, quality, or size of an object. For example, the measurements of the object *box* can be expressed by *volume, height, weight*, and the measurement of the object *process* can be expressed by *effectiveness, cost, duration, or maturity*. Measurements can be used for benchmarking. See also *benchmarking*.

measurement point. A designated point in a process path where measurements are to be taken. See also *critical measurement point*.

methodology. A collection of related techniques and notations based on a common philosophy to solve a series of major tasks. See also *IBM LOVEM*.

metrics. The definition and description of a procedure for taking measurements. Metrics can be assigned to activities as *actual, target, or ultimate* values.

MFC. Microsoft foundation class, used in Visual C++ programming.

mission. A general statement of the purpose and nature of an enterprise.

model. The graphical and written representation of observations and predictions of how a design could or should be implemented. Models are usually built at various levels of abstraction and detail. For example, a business model depicts a defined business area that is important to the enterprise; it can be shown as different views of the same business area, such as:

- Process or process path
- Organization
- Performance

modeling. Part of the design process used to create alternatives or *what if* scenarios before committing to the final design. See also *model*.

moment of truth. Your customer's perception resulting from any contact that your company has with that customer either in person or through a document, product, or system. See also *service encounter*.

MQSeries. Message Queue Series: a communications layer, which establishes connection between two systems. With MQSeries, messages can be sent and received through queues.

MDI. multiple document interface

N

navigation. The movement from a completed activity to downstream activities. The paths followed are determined by control parameters, their associated transition conditions, and by the start conditions of activities. See also *control connector, start condition, exit condition*, and *transition condition*.

navigation evaluation. The process FlowMark uses to decide which path to take. See navigation.

node. A point at which one or more functional units connect. In a process diagram, nodes are the symbols or objects that can be joined by connectors or flows, such as activities, systems, blocks, sources, and sinks.

NT. Microsoft Windows NT (New Technology)

NULL. empty, of no value

O

ODBC. open database connectivity (Microsoft Corp., based on X-Open CLI)

OLE. object linking and embedding (Microsoft Windows)

opportunity area (OA). A point in a process or process path where possibilities, advantages, or other positive factors can help an enterprise to meet its goals.

organization. An administrative unit of an enterprise. In FlowMark: organization is one of the criteria that can be used to dynamically assign activities to people. See also *role, child organization* and *parent organization*.

organization unit. An administrative subdivision with reporting lines that implement processes, for example, a department.

OS/2. IBM′s Operating System/2 for workstations.

output container. In FlowMark: storage for data produced by an activity, process, or block for use by other activities. It can also be used for evaluation of conditions. See also *sink*.

outsource. To buy services under contract from another company; for example, a company could buy all data processing services from a data processing company on an ongoing basis.

P

packet of data. A grouping of related data.

packet of information. A grouping of related information.

parent organization. In FlowMark: an organization within the hierarchy of administrative units of an enterprise that has one or more child organizations. A child is one level below its parent in the hierarchy. See also *child organization*.

pattern activity. In FlowMark: the single program or process activity in a bundle from which multiple instances, called bundle activities, are created. See also *bundle activity*.

PC. Personal computer or workstation.

physical. The concrete, specific, or constrained nature of business processes and data. Physical defines the *how, where, when*, or *by whom* a process is performed or data is used. See also *constrained*.

physical constraints. See *constrained*.

physical model or blueprint. The depiction of the efficiency of a process: *doing the thing right*. The physical model or blueprint shows the *how, where, when*, or *by whom* aspects of an enterprise, such as the resources required to perform a process or process path. See also *model, blueprint*, and *physical*.

planning activity. In FlowMark: a special program activity that creates, at run time, the required number of bundle activities for a specific bundle. The planning activity must use a program that refers, in its registration, to the bundle-planning tool supplied with the FlowMark product. See also *program activity, program registration*, and *bundle activity*.

platform. The operating system environment in which a program runs. FlowMark is a distributed, cross-platform application, which can run on OS/2, AIX, and Windows.

policy. A principle, plan, or course of action pursued by an enterprise.

problem. An obstacle to meeting a goal or fulfilling a CSF. A problem can also be a situation or issue that is uncertain, complicated, or difficult to deal with.

problem area (PA). A point in a process or process path where difficulties, constraints, or other negative factors prevent the enterprise from meeting its goals or CSFs.

procedure. A series of steps or activities required to perform a process.

process. A business function or operation that achieves results for customers with input from suppliers. A process transforms the nature, status, or composition of input to produce output according to business rules and policies. A process is a means to achieving the goals and strategies of an enterprise. See also *subprocess*.
In FlowMark: a process is a set of activities that must be completed to accomplish a given task.

process activity. In FlowMark: an activity to which a separate process is assigned. Starting this activity creates an instance of the referenced process and starts it. See also *program activity*.

process administrator. In FlowMark: the person responsible for the execution of a process instance. A process administrator can be specified in the workflow model; otherwise it is the person who starts the process.

process category. In FlowMark: an attribute that a process modeler specify for a process. Only users, who are authorized for this category can start and control instances of the process as top-level. See also *top-level process*.

process cycle time. The elapsed time required to receive, process, and forward a transaction.

process diagram. A graphical representation of a process or process path that shows all its components.

process instance. In FlowMark: an executable copy of a process template in run time

process management. In FlowMark: the run-time tasks associated with process instances, such as creating, starting, suspending, resuming, terminating, restarting, and deleting process instances. See also *business process management (BPM)* and *process path management*.

process manager. A manager, who has the delegated responsibility from a process owner to manage the day-to-day operations of a process or process path. See also *process owner*.

process maturity. A measure of how well and how complete a process or process path has been defined and is being managed. See also *maturity rating*.

process owner. A senior manager, who is responsible for managing all aspects of a business process or process path. A process owner usually delegates the operational management to process managers. See also *process manager*.

process path. A sequence of processes or activities and flows of data or information that produce a specific product or service. A process path usually starts and ends with a customer service encounter. See also *service encounter*.

process path management. The management of a business across the traditional vertical processes or organizations; for example, in a traditional order fulfillment company, the vertical processes are *sell, order, supply, distribute, settle*.

process status. In FlowMark: the status of a process instance. The status can be one of the following:

- Ready
- Pending
- Running
- Suspended
- Terminated
- Finished.

process template. In FlowMark: the translated form of a workflow model in run time.

program. In FlowMark: a computer-based application that supports the work to be done in an activity. Program activities reference executable programs using the logical names associated with the programs in FlowMark program registrations. See also *program registration*.

program activity. In FlowMark: an activity to which a registered program is assigned. Starting this activity invokes the program. See also *process activity*.

program registration. In FlowMark: identification of a program to a FlowMark database, so that it can be assigned to a program activity in a workflow model. See also *program activity*.

Q

quantum leap. A radical change in direction and results. For example, radical changes to your company should mean quantum leap improvements in your business results.

R

RAL. See *representative attributes list*.

recommendation. Advice or suggestion on how to meet a goal, solve a problem, evaluate a CSF, or carry out a strategy or policy.

refinement. A standard modeling technique used to view the parts of a whole in increasing amounts of detail. See also *hierarchical structure diagram (HSD)*.

relationship. A descriptive association between two data entities or relationships in a data model.

report. Formatted text and graphics, usually generated by a system.

report layout. The design and specifications for the format of a printed report. See also *screen layout*.

repository. An organized, shared collection of data or information that supports business process re-engineering, application development, or business or systems management. It is usually automated and is implemented as a database.

representative attribute. An attribute that defines a data or information flow. For example, representative attributes for the data flow *customer data* could be customer name, customer address, or customer number.

representative attributes list (RAL). A list of representative attributes that define a data or information flow.

REXX. Restructured extended executor language. A procedures language.

RISC. Reduced instruction set computer (runs AIX).

role. In FlowMark: a responsibility that is defined for staff members. Role is one of the criteria that can be used to dynamically assign activities to people. See also *organization*.

root cause analysis. The analytical determination of the cause of the symptom of a problem.

run time. In FlowMark: the component used to execute process instances. The run-time component consists of:

- Run-time server
- Program execution client
- Bundle server
- Notification service
- Delivery server
- Run-time client

See also *Buildtime* and *run-time client*.

run-time client. In FlowMark: the user interface for working with process templates, process instances, work lists, and work items. See also *run time*.

S

SCSI. small computer system interface

screen layout. The design and specifications of the image that the user sees on the screen of a system. See also *report layout*.

SDI. single document interface

service encounter. Any point of contact with your customer. See also *moment of truth*.

shredder. A machine for the destruction of documents.

simulation. A mock-up version or prototype of the new process and job design used to test assumptions before final implementation.

sink. In FlowMark: the symbol that represents the output container of an activity, process, or block. See also *output container* and *source*.

skill. An ability, proficiency, expertness of a person that comes from training, practice, and experience.

SMTP. simple mail transfer protocol (Ethernet, based on TCP/IP)

source. In FlowMark: the symbol that represents the input container of an activity, process, or block. See also *input container* and *sink*.

SQL. structured query language

staff. In FlowMark: the people and their roles, organizations, and levels, who execute the process instances. Staff is defined in a FlowMark database. See also *role* and *organization*.

staff resolution. The process FlowMark uses to decide which work lists to add a process to.

start activity. In FlowMark: an activity that has no incoming control connector. A start activity becomes ready when the process or block that contains it starts. There can be more than one start activity in a process or block.

start condition. A control setting that determines when an activity with incoming control connectors can start. It also determines when a process path or workflow can start. See also *condition, exit condition*, and *transition condition*.

STDERR. standard error

strategy. A pattern of goals, policies, and plans that specify how an enterprise is to function over a given period of time. A strategy can specify areas for product development and marketing as well as techniques for responding to competition.

subject expert. A specialist in a particular area of expertise, such as workflow management.

subprocess. A lower level process. Processes can be refined into sub-processes through the HSD modeling techniques. See also *hierarchical structure diagram (HSD)* and *process*.
In FlowMark: a process instance that is started by a process activity.

substitute. In FlowMark: the person to whom an activity is automatically transferred if the person to whom the activity is assigned is flagged as absent.

symbol. A graphical representation of an object or thing, which may be abstract in nature; for example, a line with an arrow is the symbol for a data or information flow.

system. A set of processes with a common aim that act on data or information using input and producing output. Usually used in the context of *information system* or *data processing system*.

system development. The design, code, test, implementation, and maintenance of an information system. Can also denote a business function, which performs systems development.

system function. A component or module of a system. A system can be refined into system functions using the HSD modeling technique. See also *hierarchical structure diagram (HSD)* and *system*.

T

task. The lowest level of activity or unit of work. Tasks belong to the physical business models. There is no implied sequence or order in performing tasks within an activity. Activities can be refined into tasks using the HSD modeling technique. See also *hierarchical structure diagram (HSD)* and *activity*.

task flow. In process-based applications, a task flow is part of a control flow. See also *control flow, workflow, event flow*, *and document flow*.

TCP/IP. Transmission control protocol / Internet protocol. A commonly used protocol for various network environments, such as Internet, Host communications, and LANs. FlowMark uses either TCP/IP or APPC. See also *APPC*.

technique. A procedure for doing anything in an orderly way; a method.

top-down. Modeling or designing business processes from their most abstract level down to their most concrete and constrained levels of detail. See also *bottom-up*.

top-level process. In FlowMark: a process that is started from a user′s process list or from an application program.

transition condition. In FlowMark: a logical expression associated with a control connector. If specified, it must be **true** for control to flow along the associated control connector. See also *control connector, default connector, condition, start condition*, and *exit condition*.

trigger. An event or condition that start or ends an activity, a process, or process path. See also *start condition* and *exit condition*.

U

unit of measure. A standard dimension, extent, or quantity, such as days, hours, or minutes, dollars of cents, or meters or centimeters. A unit of measure is used for measurement purposes.

upstream. Preceding processes or activities, for example the *order* process is upstream from the *distribute* process. See also *downstream*.

URL. Uniform (or Universal) Resource Locator. A location on the World Wide Web.

V

VHLPI. VisualInfo high-level programming interface. The service broker for VisualInfo. It can be used to integrate FlowMark with VisualInfo for document management.

VisualBasic. A programming language under MS Windows.

VisualInfo. An IBM product for document management.

V2R1. Version 2 Release 1.

V2R2. Version 2 Release 2.

V2R3. Version 2 Release 3 (also V2.3).

W

Web. World Wide Web (Internet)

workflow. A sequence of activities (units of work). A movement of units of work through a process. In process-based applications, it can be part of a control flow. See also *control flow, task flow*, and *event flow*.

workflow management. The art of controlling or administering a sequence of activities.

workflow model. In FlowMark: a complete representation of a process. It consists of the process diagram and settings and the definition of staff, programs, and data structures that are associated with the activities of a process.

work list. In FlowMark: a list of work items assigned to a staff member.

work item. In FlowMark: a single item or task on a work list.

World Wide Web. A collection of servers containing various types of information, all connected together by a set of TCP/IP communications links known as the Internet.

www. World Wide Web (Internet)

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

ITSO Redbook Evaluation

Image and Workflow Library:Sample Applications for FlowMark
SG24-2184-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

_____

_____

_____

_____

_____

This soft copy for use by IBM employees only.

Printed in U.S.A.