SG24-4712-00

Workflow and Image Library: FlowMark and VisualInfo with Windows

August 1996

International Technical Support Organization
Rochester Center

International Technical Support Organization

SG24-4712-00

Workflow and Image Library: FlowMark and VisualInfo with Windows

August 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special Notices” on page 117.

First Edition (August 1996)

This edition applies to Version 2, Release 2 of IBM FlowMark, Program Number 5622-615 for use with the IBM OS/2 Operating System and to Version 1, Release 2 of IBM ImagePlus VisualInfo, Program Numbers 5655-036, 5622-231 and 5621-326 for use with the IBM OS/2 Operating System.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

Contents

Figures

Tables

Preface

This redbook shows how a successful pilot in workflow and document management led to a complete system project for a large bank. It provides information about the integration of FlowMark, VisualInfo, and the Windows client. It contains many examples of sample code, applicable to clients using Windows or OS/2.

This book was written for consultants, system architects, solution architects and programmers.

Some knowledge of FlowMark, VisualInfo and programming techniques is assumed.

(150 pages)

How This Redbook Is Organized

This redbook is organized as follows:

- Chapter 1, "Workflow and Document Management"

  This provides an overview of considerations when implementing a document management and workflow system, and provides advice on managing a project of this type.

- Chapter 2, "Customer Project Overview"

  This describes a pilot system that was developed for three banks.

- Chapter 3, "FlowMark - the Workflow Manager"

  This introduces you to the IBM FlowMark workflow software.

- Chapter 4, "VisualInfo - the Document Manager"

  This introduces you to the VisualInfo document/imaging software.

- Chapter 5, "Implementing an Integrated Solution"

  This provides tips on using FlowMark and VisualInfo in an integrated solution.

- Chapter 6, "Integration Techniques"

  This provides techniques to create a pilot system using the VisualBasic, VisualREXX, and C languages.

- Chapter 7, "Hints and Tips"

  This provides additional hints and tips for the solution.

- Appendix A, "FlowMark C++ API Sample"

  This appendix includes sample code that describes how FlowMark WorkFlow Client C++ APIs can be used to add more function to your workflow application.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Rochester Center.

Guido Auberger is a technical image and workflow specialist in Germany. He has two years of experience in the document and workflow management field. His areas of expertise include design and implementation of document and workflow management solutions in the OS/2 and OS/400 environments. He holds a degree in Computer Science from the BA Mannheim. After working three years at IBM in Mainz, Germany he joined SerCon in 1994.

Unae Choi is a technical workflow specialist in Austria. She has over two years of experience in the workflow management field. She holds a degree in Computer Science from the Technical University of Vienna. Her areas of expertise include implementation of workflow, imaging, and document management solutions in a client/server environment.

This redbook is published by the International Technical Support Organization.

Mike Ebbers is a Senior Product Support Representative at the International Technical Support Organization, Rochester Center. He has 22 years with IBM. He produces redbooks on workflow and image. Mike previously developed and taught courses on imaging and printing.

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

Christian Arnoux
IBM Switzerland

Francois Baud
IBM Switzerland

Jean Francois Chavannes
IBM Switzerland

Serge Pilet
IBM Switzerland

Werner Ruppert
IBM Germany

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. Workflow and Document Management

This redbook is based a project which successfully integrated document management and workflow products into a cohesive solution. The pilot described in this book has been subsequently approved by the client, resulting in an order for a production system.

1.1 Integration of Workflow and Document Management

This section gives an overview of the considerations for implementing a workflow and document management system, and references other books which give valuable information in this area.

We will introduce the concept of workflow modeling as it relates to the design of application programs for use with the IBM FlowMark workflow manager.

We will then discuss some points to consider when introducing a document management system to your customer.

1.1.1 General Requirements

The main benefits of an integrated workflow and document management system are:

- Eliminating paper dependence
- Improving the management of work processes
- Integrating seamless applications
- Using consistent and efficient processes

With a document management system, you are able to handle various forms of information such as image, voice, fax, EDI, and word processing.

- Flexible capture, retrieval techniques, and user interfaces meet the customers′ varying business requirements.
- Users can always view document folders and images in the system, independent of their physical location.
- Multiple users can work on the same document images, therefore no copies are required.
- All users are able to view the latest version of a document.
- The search of documents is independent of the document′s main index, so that any index field can be used to find a document.
- Images need less space for archiving than the original paper documents.
- The underlying workflow system allows graphical definition and consistent documentation of business processes.
- Organizations can easily adapt the system to changes in their business processes.
- Execution of processes can be controlled and monitored at all times.
- Eliminate entry and re-entry of the same data.

- The system assists the users in their daily work, by automating the flow and providing all of the necessary information and tools.

1.1.2 Workflow Design and Definition

A workflow management system manages people, processes, programs, and data according to user-defined rules, conditions and routing, and allows management reporting and tracking of work-in-progress.

The need for workflow usually originates when searching for a technology that improves customer service, responsiveness, and efficiency.

The IBM **FlowMark** program product provides a way to model a process, then assign applications to activities in the resulting workflow model. This allows the workflow manager to automate the control of activities and the flow of data.

Work is routed to the person who does the activity. A tool or application program that is required to perform an activity can be designed to be started automatically when a user starts an activity from a worklist.

In a typical workflow project, the solutions architect and the workflow specialist define the workflow models, the application programs, and the data structures needed to support activities. The workflow specialist not only is able to define workflow models using the FlowMark buildtime components, but also has a general knowledge of various programming and integration techniques and how these pieces can fit together. The workflow specialist may take part in the actual coding of the application integration parts, but this is usually done by the programmers. The following sections apply to solution architects, workflow specialists, and programmers of workflow systems.

Note: The word ″application″ is used interchangeably with the word ″program″.

Always remember to view the workflow system from the perspective of the actual business processes and the work being performed as part of these processes. A workflow system is designed to help users perform their part of the work in processes involving several people or several departments in an enterprise. That is, the purpose of the workflow system is to let users spend most of their time performing activities that have add-on value to the business. This is achieved by providing the users with all of the data and tools needed to perform their work with an easy-to-use ″to do list″ interface called **work list**.

A workflow system not only takes away from end users the burden of searching and starting the right applications to perform some work and re-entering the same data multiple times in different applications, but also provides a way of controlling and administering the status of complex and long lasting business processes, and makes sure that business rules are being correctly followed.

The main advantage of workflow systems over large applications, where business rules are hard-coded, is that you can quickly adapt to changes in business processes without having to change, recompile, and reinstall your entire application. This can be achieved because in workflow systems, control and flow logic are separated from the applications, and the applications are functionally decomposed in several reusable modules.

So in designing and implementing applications to be integrated with a workflow manager such as FlowMark, you need to separate the control flow and data flow from the applications, and move the start and exit conditions to the workflow model.

Most applications include many diverse functions can support several different activities in different stages of a process. Output produced by one function of a program can be used as input by another function of the same program. Therefore, the same application can be used to support many different program activities in a workflow model.

For detailed information on designing processes in FlowMark to meet performance and capacity requirements, refer to the *FlowMark V2.2 Design Guidelines, SG24-4613*.

For detailed information on FlowMark modeling concepts, refer to *FlowMark V2.1 Modeling Workflow, SH19-8241-00*.

For detailed information on FlowMark APIs and integration concepts, refer to *IBM FlowMark: V2.1 Programming Guide, SH19-8240* and *IBM FlowMark Application Integration V2R2, SH12-6267*.

For hints and tips on installing and maintaining workflow systems based on FlowMark and VisualInfo, refer to *FlowMark Installation and Administration, SG24-4614*.

1.1.3 Document Management Design and Definition

A document management system allows documents to be electronically captured, stored, retrieved, and displayed. A document management system can handle various forms of enterprise information such as image, voice, fax, EDI, word processing, and spreadsheets. This reduces the amount of paperwork handled by businesses. It improves productivity by storing images of paper documents and routing them electronically to where they are needed.

In a document management system, electronic documents are indexed, routed, and added to business folders just the same as normal paper documents. Unlike paper, the electronic documents can be accessed by multiple users at the same time and can be stored and administered centrally.

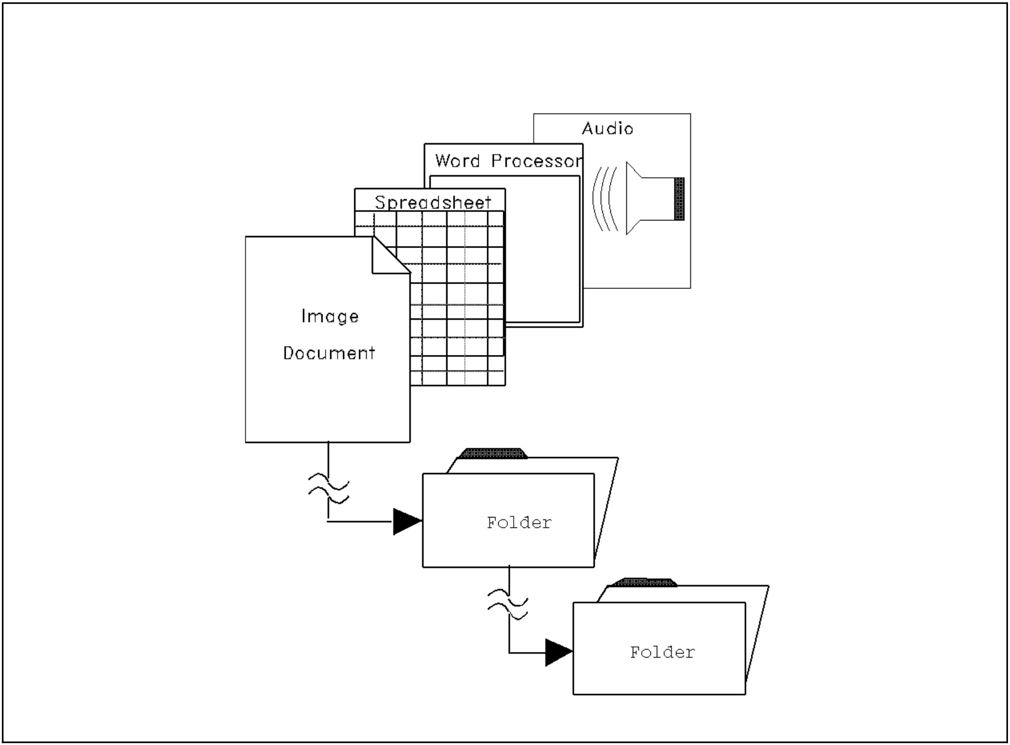


Figure 1. Electronical Folders and Documents in a Document Management System

To manage your documents with **VisualInfo**, you have to consider the following:

1. Scanning and indexing (document capture process) of the archives with folders and documents. Your main focus is the different types of documents, but you also consider the daily occurrence of such document classes.
2. Storage and retrieval in the VisualInfo system.
3. Document management structures under VisualInfo Windows client.

The single most expensive part of the total pilot system is certainly the initial capital investment of scanners, PCs, rooms, and environment that you need, and the on-going labor cost of operating scanning and indexing stations (approximately 80 percent of the total cost). This is the reason it is worthwhile to invest a reasonable amount of time and money in the analysis and optimization of the current and future document capture process in relationship to the total business process. The document capture process is composed of five separate modules:

1. Preparation of document classes, batches, documents, and pages.
2. Scan and rescan.
3. Image processing with cleanup, OCR, bar code, and patch code.
4. Indexing with key entries and index validation.
5. Release, commit images, indexes, and full text to the permanent storage or workflow.

The process of setting up document classes is the basis for the first step. The following questions have to be answered:

- What type of information is going to be used for retrieving these imaged documents? Which index fields are used in each document class?

- How are documents of this type to be processed? Which queues are used for each type?
- What type of physical documents are being dealt with?
- What are the volumes in total?

The more carefully the **batch**es in the document capture process are prepared, the easier the following steps are going to be. Often scan tools have predefined batch types. These batch types were created based on many different types of processing that have been encountered. This allows you to quickly select a batch definition that matches your requirements for processing the given documents.

One of the difficult concepts of the document capture process is understanding batch processing. The contents of a batch can vary from one business to another based on its individual needs for paper handling and storage. A batch can consist of several documents that make up a folder or just a unit of identical forms that are processed together.

Batch preparation includes sorting the documents, identifying the types of forms and documents, and removing the staples. With proper preparation accomplished, scan operators do not need to know any of the characteristics you defined for the batch; they just have to choose the right batch type by name.

Scanning is accomplished in two different ways: single document scanning and batch scanning. In single scanning, the user manually specifies when the last page of the document has been scanned. Batch scanning consists of loading a stack of documents into an automatic document feeder and starting the scan process, where the documents are separated by bar codes, patch codes, blank pages, or a fixed count of pages per document. Post-scan activities would include cleaning up the images and de-skewing them. The Kofax AscentCapture product, for example, was created to help with the batch scanning process.

Scanning a document converts the information on the paper into pixels using a raster scan process. If the image is not acceptable because the contrast or brightness should be adjusted, then the document is **rescanned** after the appropriate adjustments have been made to the scanner.

During the **image processing**, the skewing of the scanned documents is corrected, and black borders and unnecessary parts of the layout are removed (forms dropout). Then, optical character recognition (OCR), optical mark recognition (OMR), and reading of patch codes and bar codes may be performed in this step.

Each document or folder must be indexed into the system to ensure that it is easily retrieved later. Indexing can be performed manually or automatically. **Manual indexing** is needed when it is not possible to read the index data from the scanned document (such as a formless letter or unstructured document):

1. Index before scan:

   Operators enter index information on their workstations when they view the paper documents and receive a temporary ID. This ID is an alphanumerical string or bar code; it is needed when the paper document is scanned later.

2. Index after scan:

Scanned documents are displayed on a workstation and operators have to enter the index information that they can read on the image.

Automatic indexing is a fast way to index your images. To use automatic indexing, your documents have to be well defined forms with areas on them where the information can be retrieved from:

1. OCR and OMR:

   Using optical character or mark recognition, indexing information is read from the image itself. If you use simple and monospaced fonts, you get pretty good results. Advanced techniques such as forms dropout and skew correction improve the OCR success rate. Misread information must be corrected by operators viewing the image.

2. Bar Code Recognition.

3. Indexing Data provided with the object. This data can be checked against a database before or after scanning, which allows verification of fields and retrieval of data so that other fields can be filled in. **Index verification** is a second index pass, so that the indexing information can be verified and ″bad″ pages can be rescanned.

   OCR full text scan can be included in your processing. This results in a character-based document which is machine-readable and can be used for subsequent full-text searches.

   Batch release is when all documents in a batch are imported into VisualInfo.

1.2 Setting Up a Workflow and Document Management Project

This section provides some general advice on managing a workflow and document management project. We provide tips on how to build the project team, the skills required and a few things to consider when planning and implementing the various project phases.

1.2.1 Defining the Project Goal

General goals may be to:

- Decrease turnaround time.
- Increase productivity.
- Decrease cost.
- Improve quality.

However, when implementing a workflow and document management system to be rolled out for a production environment, you should specify the goals more precisely and make sure that they are measurable, achievable, and ranked. Most of all, the goals should not be conflicting.

Examples of goals may then be to:

- Reduce the cycle-time of a process from 1 month to 2 weeks.
- Achieve 70% increase in rate of business processes completed.
- Achieve 20% decrease in cost.
- Achieve 20% decrease in customer complaints.

1.2.2 Building the Project Team

To cover a workflow and document management project, your project team should consist of members with at least the following roles:

Table 1. Project Members

Role	Description
Project Manager	Coordinate project team and contact to customer.
Solutions Marketing Representative	Primary customer contact.
Solutions Architect	Design overall solution; develop test scenarios.
Industry Advisor	Interface to customer end-user requirements.
Technical Support	Install and support HW/SW.
Workflow Specialist	FlowMark technical support; verify business process model requirements.
Image Specialist	Network and client/server support; help with network considerations such as configuration, network load and accessing distributed resources. verify requirements concerning document types and quality.
Network Specialist	VisualInfo technical support; verify requirements concerning document types and quality.
Programmer	Code tools and programs assigned to the workflow activities.
Tester	Test according to the test scenario.

This table lists the minimum requirements to execute and successfully complete projects of this type.

The **project manager** is responsible for the coordination of the entire project team and their communications. The manager also does the administrative work such as creating and updating the project plan. Further, we recommend that the manager keep in contact with a defined representative of the customer. The project manager should make sure that all internal and external meetings and their results are protocolled.

The first customer contact is usually established by a **solutions marketing representative**. It is the representative′s job to convince the customer that your solution is the best for this application. Thus the representative has to provide the customer with all the information about products and services that would satisfy the customer′s needs.

The **solutions architect** designs the overall solution and creates a design document. The architect needs input from the workflow and image specialists, the industry advisor and the technical support persons. The architect has to make sure that the individual components and the complete system can cope with the capacity requirements that were negotiated in the contract with the customer. Therefore, the architect has to verify the feasibility of all single requirements, such as, amount of documents (scanned, retrieved, and processed in the workflow), texture and size of the documents, the different types of documents, the number of process instances and their duration, number of users, and general platform specific requirements.

Special industry knowledge is supplied by the **industry advisor**. The advisor knows the terms the customer is talking in, and is able to ″translate″ them to the other team members. Therefore, the industry advisor is an interface between a representative of the customer′s end-users or experts and the project team. The advisor must also provide input relating to the customer′s needs for the project plan.

Hardware and software installation and configuration according to the design document is done by one or more **technical support** persons. It is recommended that they coordinate and document all of their activities. The result of their work must be a unified and reproducible environment.

The **workflow specialist** has two major jobs to do. First, the specialist has to verify the business process model and assist the solutions architect in designing the overall solution. Second, the specialist must help define the business process model in FlowMark. The industry advisor gives the specialist input about the customer′s organization, staff, roles, and their relationships.

The document′s type and quality is analyzed by **image specialists** and is reported to the solutions architect. They have to check and customize the hardware and software for image input and output. Therefore, they specify the peripheral hardware (scanner, monitor, and printer) and VisualInfo definitions such as volumes, storage groups, index classes, and access lists. To do so, they have to understand the current customer business process, distinguish the document classes (logical documents), and understand the components of a document class. They need to be able to note the key lookup fields that need to be included for indexing, and to determine the necessary processing requirements for a document class.

Workflow activities receive processing, logic and decision-making ability from programs attached to them. Therefore one or more **programmers** must be involved. The information that is necessary for this job comes from the design document. It is necessary that all programmers are able to ″understand″ and fix the code of others; therefore a unified documentation of the code and a version management is needed.

The **testers** have to verify that the system functions as described in the test scenarios developed by the solutions architect. They are responsible for documenting abnormal system behaviors in a detailed and reproducible way.

1.2.3 Technical Skills Required

To carry out a workflow and document management system using FlowMark and VisualInfo, you need at least the following technical skills:

1. Product Skills

   - **Target Operating System**

     It is very important to know the target platform. The success of the solution depends on the operating system and on the applications used.

   - **Database**

     Database design skills will help you to optimize capacity and performance. Distributed database programming skills are becoming increasingly important in the client/server environment.

   - **Client/server**

     Advanced client/server design and programming skills (such as RPC, APPN/APPC, and CICS) may be needed for your project.

   - **FlowMark and Service Broker**

You need to know the benefits of your workflow application. Questions such as the following should be answered: What network protocols does the workflow application support, and what tools or building blocks are available to integrate desktop and host applications with it? Does the workflow application always require a database or can you just use it optionally to store huge amounts of data? Is it possible to document, model,and backup your workflow models easily?

- **VisualInfo**

  The document management system you choose, is as important for your solution as the workflow application is. You need to know supported environments, image and file formats and so on. Also it is interesting to know how the document archive works. How do you define or change index classes and what kind of search methods are supported? To be sure of having no problems with storage, you should check the availability of storage devices and their limits.

- **VisualInfo Windows client**

  The Client Application is used to access VisualInfo from a Windows-based environment. Not all of VisualInfo′s functions may be accessible through the VisualInfo Windows client, VisualView, or other client applications; therefore, you must check for conflicts with the customer′s requirements.

2. Programming Skills

   - **C, C + +**

     C++ programming skills are needed to use the FlowMark Workflow Client APIs and C programming skills to use the VisualInfo, VisualInfo Client Application, and FlowMark C API functions.

   - **REXX**

     On the OS/2 platform, Visual REXX can be used to rapidly do the user interface and the integration with FlowMark.

   - **Visual Basic**

     On Windows, the equivalent of REXX programs can be carried out with Visual Basic.

   - **DDE, OLE**

     To integrate applications and exchange data between different applications, DDE and OLE techniques can be used.

   - **HLLAPI**

     For the integration of existing host applications, HLLAPI programming may be required.

3. Document Management and Workflow Skills

   To have a basis for sizing the entire project, data and facts have to be collected and analyzed by persons with document management and workflow skills.

   The persons with document management skills should understand the components of the scan queue and be able to set up its features (bar code labels, OCR, patch code, batch totals, and acceleration cards). They also have to know the components of the OCR index and zoning queue, the index verification queue, and the release queue, and be able to set up the OCR index and zoning queue features (registration zone, manual zones, and OCR zones). They also must know the features of the index verification and release queues.

They need to understand the components of the scan module, know how to set up its features, and create and modify batches (scanner selection, scanner set up such as resolution, contrast, imprinting, and bar code). Further, it is necessary to understand the components of the batch manager, the OCR module, the index and index verification module, the rescan module, and the release module.

The persons with document management skills need to know how to design preprocessing and postprocessing scripts and macros (document class validation scripts and index filed template macros). They have to know the functions, capabilities, and features of viewers: What graphic formats are the viewers able to display, and are they able to work with COLD data?

The workflow specialists have to understand the customer′s current business processes in detail, and also be able to find areas of improvement. They need to analyze the processes to identify the activities, their relationships and dependencies. With their FlowMark modeling skills, they should know which modeling concepts (bundles, manual checklists, subprocesses, notifications, blocks, and support tools) can be used to carry out the processes and activities. They have to decide which data is application specific and which is for workflow control, then design the data flow accordingly. The organizations, roles, people and their relationships also need to be defined.

For the performance and capacity planning, the workflow specialists need to gather information on the different types of processes (number of templates), the number and frequency of process instances started, number of concurrent and total users, and the usage of data containers.

1.2.4 Planning the Project Phases

Generally in a workflow and document management project, you should perform the following steps to successfully develop a pilot application:

1. Get customer requirements.
2. Create design documents (prototype).
3. Review prototype design.
4. Implement and test the prototype.
5. Update customer requirements.
6. Create design documents (pilot).
7. Review pilot design.
8. Implement and test the pilot.
9. Roll-out of pilot.

You should be aware that the preceding steps may overlap and contain several iterations. The **customer requirements** can change continuously during your project. You have to define from the start how the initial and additional requirements are gathered and documented. This should always be documented in agreement with the customer. For every requirement that is added, changed or refined, you have make sure that the customer knows the impact it has on the complete project. Even a seemingly small requirement may lead to critical side effects, which then results in unexpected time and resource problems. It is essential that you and the customer have the same understanding of the scope of the project.

From the beginning, you need to put a project review process in place internally for your project team, and externally for the customer. For a successful workflow project, you have to filter those business processes of the customer that have the most added value and where the benefits are most visible. When defining the solution concept, you have to consider how the solution fits into the customer′s Information Technology (IT) architecture.

The **design documents** define the implementation concept, the standard applications used and integrated, and what needs to be developed. The purpose of this step is to clarify the complexity and risks of the solution, and show the feasibility of the solution in the given cost and time frames. The requirements specifications and the design documents must ensure that your project team understands the customer′s processes.

The design documents should be reviewed thoroughly by an internal team which at least includes some of the authors of the design documents, someone with the industry knowledge of the customer′s business area, and a quality assurer. Following this, the customer must also **review the documents**. Any changes and open points resulting from the reviews should be discussed and incorporated in the design documents. This is an ongoing process that ends only when all of the participants agree to the designed solution in its various levels of detail.

Before the **implementation phase**, you have to set up the development environment, which includes acquiring and installing all of the required software and hardware, and extending the project team with the appropriate technical skills. During implementation, you should keep an ongoing contact with the customer, hold regular project status meetings, verify that the defined project check points are still on time, and document the project progress. To finish the implementation phase in the planned time frame, it is essential to have a reliable support structure to help you with hardware and software problems. Parallel to the implementation of the prototype, you can start the testing according to the test cases described in the design documents.

The finished prototype must fulfill the expectations of the customer concerning the technical feasibility and benefits of the new technology in their business processes. The solution not only must correspond to the customer′s needs, but also should be presented in a way that is optically attractive and intuitive. Even if the customer is satisfied with the prototype as is, it is most certain that they will **refine the requirements**, which leads to an improvement of the prototype to be suitable for productions environment. This then is the beginning of the pilot project phase.

The **pilot design documents** are similar to the ones of the prototype, but they also define the operational concept and are based on the refined customer requirements. When designing the pilot workflow solution, it is vital to consider the implications of the solution on the existing systems and applications, the company′s organizational structure, the security requirements, the scalability in a distributed environment, and the system management. The pilot solution may be far different from the prototype as you consider the production values such as the peak and average number of concurrent users, workload, network traffic, server load, time and place dependencies, usability, and performance.

The **review process** of the pilot design documents is identical to the prototype design documents, except that the end-users are more involved. We recommended that you set up your pilot project team with members from the prototyping phase. They are not only familiar with the environment, but know the particular problems that may occur, understand the customer′s terms and processes, and are already a well-adjusted team.

The **implementation and test of the pilot system** has to ensure a smooth porting of the system to the customer′s production environment, and that the pilot meets the stability criteria according to the design documents. Besides the normal test cases, performance and stress tests with realistic scenarios and quantities have to be performed. Once you have completed and tested the system, and provided technical and user documentations to the customer, you are ready for the final **roll-out of the pilot system**.

For details on roll-out and maintenance of such systems, refer to *FlowMark Installation and Administration*, SG24-4614.

Chapter 2. Customer Project Overview

This chapter covers the pilot workflow and document management system that was developed by IBM and a service provider for several banks in Switzerland. We describe the customer requirements and objectives, the IBM solution and the current project status.

2.1 Project Overview

The project was initiated in August 1995 when the customer, a service provider to several banks in Switzerland, listed requirements from the banks and sent a request for information. A detailed response was sent back by an IBM marketing specialist, containing precise facts on IBM′s standard workflow and imaging products and their benefits to the customer′s needs.

Also, a series of demonstrations was held, where base products were shown individually and in integrated scenarios. Through this, the customer was able to see the entire range of IBM′s workgroup software solutions. So the number of vendors was reduced to three.

IBM and the two other vendors were requested to design and carry out a prototype for a credit management application. The customer′s expectations of the prototype were presented by a specialist from one of the banks in meetings with the project members. The result was a rough overview of the workflow process with approximately ten activities and a few roles.

Based on this information, a design document for the prototype was written during mid-October. It described the activities, programs, responsible people, data structures and the flow in detail. The implementation of the prototype, which integrated workflow, image, host, and desktop applications on the OS/2 platform, lasted less than a month.

The prototypes of the selected vendors were shown at end of November, of which the IBM solution was preferred. However, IBM was expected to develop a more specialized prototype on an alternative platform before a final decision could be made. Following further discussions in January 1996, the project to do the prototype system was started in February. IBM was given six weeks to do the prototype.

At the end of March, a final decision was made whether to continue with IBM to carry out and deploy the pilot system, or to return to the other two vendors on the waiting list if IBM did not prove that it could provide the same solution on the Windows platform.

The objective of the project being described in the following sections was to provide a proof-of-concept workflow and document the management system, whereby the workflow process to be implemented was a credit process for financing of properties. If a final deployment of the system takes place, all banks involved in the project will use the same system. However, the input for this project (the description and requirements of the credit process), was provided by one banks, which is referred to in the following chapters as Customer A (or Bank A).

Here is a time line which shows the progession of steps for this project.

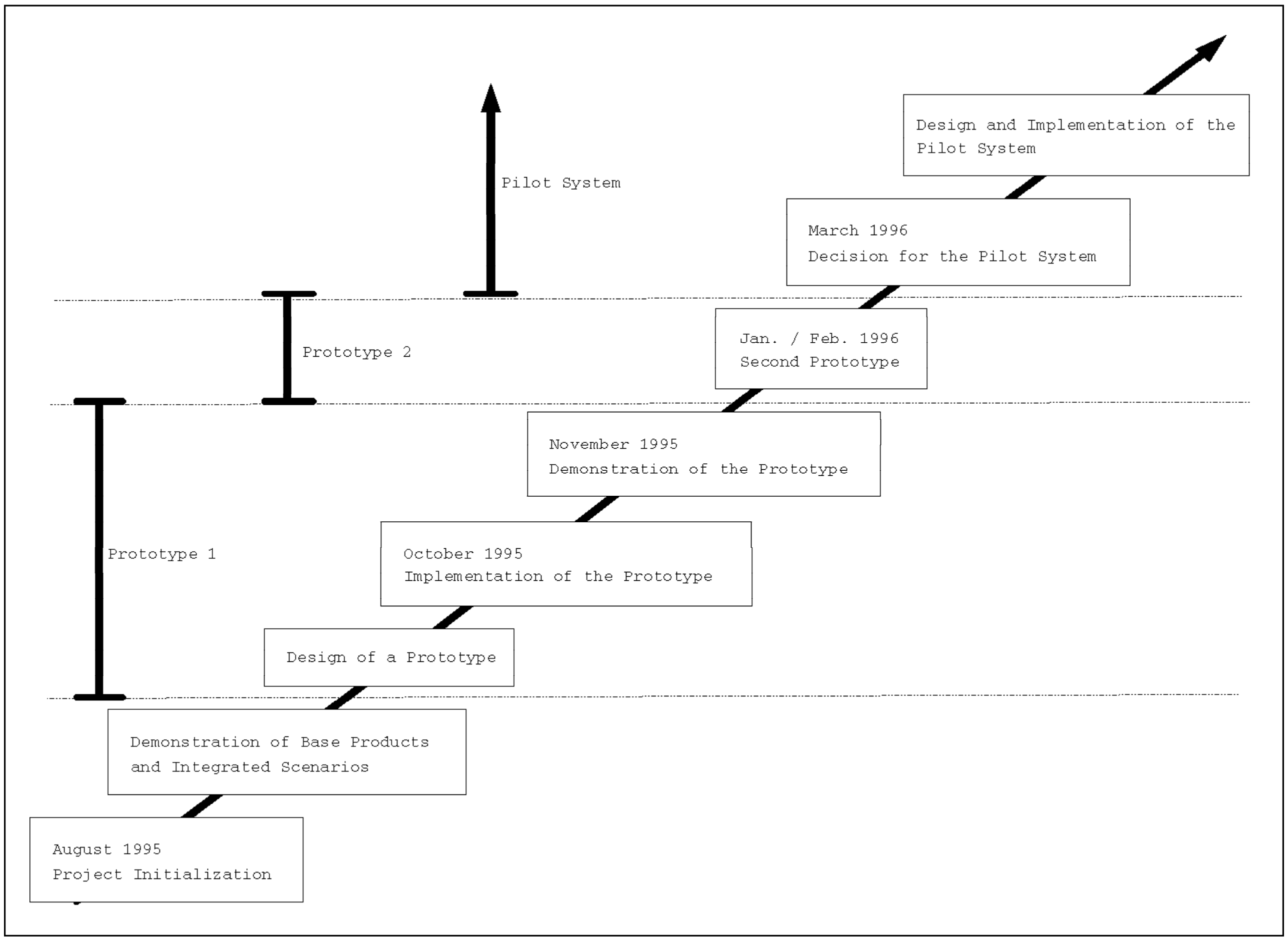

Design and Implementation of the Pilot System
Pilot System
March 1996
Decision for the Pilot System
Prototype 2
Jan. / Feb. 1996
Second Prototype
November 1995
Demonstration of the Prototype
Prototype 1
October 1995
Implementation of the Prototype
Design of a Prototype
Demonstration of Base Products and Integrated Scenarios
August 1995
Project Initialization

Figure 2. Timeline for Project Steps

The next section describes briefly the situation of Bank A and their requirements for a workflow solution.

2.2 Customer Requirements

Customer A is a large bank in Switzerland which is the result of a merger of four banks during the last two years. Since the merge, the bank faced several problems because of the different information systems being used, and different processes being followed. The most pressing business problems were identified in the management of credits and loans. Although a unified credit management process had been defined and enforced in all departments recently, it was still a paper-intensive manual system which had not brought much improvement to the process.

The customer faced problems such as:

- Critical workload in administrative tasks such as text processing and multiple data entry.
- No consistency in current workflow.
- Business rules not being followed correctly by employees.
- No possibility to control and track the status of a credit request.

To solve these problems and control processes more effectively, the bank engaged their Information Technology service provider to find the right technology innovation for their credit applications.

The service provider gathered the user requirements. They collaborated with IBM to design and do a prototype system using workflow (WF) and electronic document management (EDM).

Among others, the final workflow system had to fulfill the following requirements:

1. Allow the users to enter data related to the credit request at any point and time in the process.
2. Allow the credit request process for a specific customer to be stopped and all necessary data to be saved at any point and time in the process.
3. Allow the user to continue with the process, even if all of the defined conditions have not been met.
4. Allow the asynchronous starting and interaction of subprocesses.
5. Allow multiple authorized users to view the information concerning a specific credit request, not just the user currently working on the request.

2.3 Project Goals and Objectives

The goal of this project was to prove that our solution was ″at least as good″ as the major competitor, and to prove that the team of IBM and the service provider was the best choice for the banks involved.

The objective of this project was to implement a workflow and document management system:

- Design and develop FlowMark processes for credit applications that integrate host applications, document management, and common desktop applications.
- Implement a tight integration of VisualInfo Client Application and IBM FlowMark on the Windows platform, using IBM VisualInfo for OS/2 and IBM FlowMark for OS/2 servers.

At the start of the project for the implementation of the prototype, a document describing all of the conditions and terms, responsibilities, specifications, acceptance criteria, milestones, and objectives was created by all parties involved.

The following pages contain excerpts from these documents. This is just for information only. Not all sections of the documents are included and many details have been left out.

Project Document

Objectives

Electronic Document Management

For all end users of the electronic document management (EDM) system, we provide a user-friendly and intuitive graphical interface that allows you to:

- Index documents according to defined rules.
- Search and retrieve a specific document, a folder, a part of a folder, a group of folders, or documents of a specific document type.
- View a document and view the contents of a folder.
- Send electronically a document, a folder, a part of a folder, or a group of folders containing documents of a specific document type to other users of the system.
- Reproduce a document.
- Extract all or parts of a document to be used in other applications, always guaranteeing that the original is not modified.
- Add a document (text, spreadsheet, graphic, image, or fax) in one or more folders.

For all users who are in charge of administering the EDM system, we provide a user-friendly and intuitive graphical user interface that allows you to:

- Organize the hierarchy of folders.
- Define the document types and their characteristics.
- Define the users of the system and their access privileges.

<u>Document Capturing</u>

For this, we provide a report on:

- The organization of work at a document capturing station and the processing cycle.
  - The profile of persons assigned to preparing documents, indexing, controlling and validating.
  - The specifications of an index before scan or an index after scan.
- The specifications of the scanners such as:
  - Speed
  - Single or double-sided
  - Multi-papers
  - Multi-formats
- The specifications of the automatic recognition of forms and OCR.
- The specifications for usage of bar codes and patch codes.

<u>Electronic Workflow Management</u>

For all users who are authorized to work at workplaces of the electronic document management (EDM) and workflow (WF) system, we provide a user friendly and intuitive graphical interface that allows you to:

- Initiate a flow of activities (process).
- Execute programs attached to an activity in a process.
- Call support tools provided for an activity in a process.
- View and start activities that need to be worked on (work list).
- Monitor and provide statistics of a process.

For all users who are responsible for developing/modeling the workflow processes, we provide a user-friendly and intuitive graphical user interface that allows you to define the workflow for processing credit requests.:

- Subprocesses
- Activities
- Data flow between activities
- Programs called
- Data sent and received from programs
- Support tools provided
- Start and exit conditions of an activity

All controls of folders must conform to rules described in the ″Rules of control″.

Responsibilities

IBM

IBM must:

- Write an application to manage electronic folders of client documents that will be deployed in the banks involved in the project, and that conforms to the defined document classification rules.
- Carry out a process for credit requests that is based on the specifications of Bank A.
- Produce a deployment plan for the banks.
- Coordinate all of the activities and resources provided by IBM with the responsible project members of the service provider.

IBM provides the following software:

- OS/2 Warp Connect
- VisualInfo (server OS/2, client windows)
- FlowMark (server OS/2, client windows)
- DB2/2
- 3270 emulation for OS/2 (Communications Manager)
- TCP/IP for OS/2

IBM is responsible for the following installation and configuration:

- Transformation of Windows 3.11 workstations provided by the service provider to OS/2 Warp workstations.
- Installation and configuration of VisualInfo and FlowMark.

Service Provider

The service provider must:

- Provide all of the hardware needed by IBM to develop the system.
- Provide the specifications for the system.
- Write the test scenario for the electronic folder management.

- Coordinate with the IBM project manager for all activities and resources provided by the service provider.
- Coordinate with the end users of the banks.
- Coordinate the acceptance of the results.

The service provider must provide a project room with:

- Five working places.
- Access to the customer host information system.
- Development platform:
  1. **Server:** 1 OS/2 Warp Connect station
     - Pentium 100, 48MB RAM, 2 x 1GB HD
     - 17″ color monitor, Quadspeed CD-ROM Drive, Token-Ring card
  2. **Clients:**
     - One Windows NT (32 MB memory)
     - One Windows 95(16 MB memory)
     - One Windows for Workgroups (16 MB memory)

       Note: Hardware for client workstations is similar to server.
  3. **One OS/2 Warp Connect station:**
     - Pentium 100, 32MB RAM, 1GB HD
     - 17″ color monitor, Quadspeed CD-ROM Drive, Token-Ring card
  4. **Scanner**

     A scanner of the characteristics: A4, tdpm, 200-300dpi, SCSI
  5. **Juke-box**

     For the current project phase, this is not necessary. This is simulated by using different partitions of one disk of the OS/2 server.
  6. **Telephones**

     One international, One local, One fax-modem international

The following software is provided and installed by the service provider:

- Microsoft Office
- Lotus Smartsuite
- EXTRA! - a 3270 Emulation for Windows

The Banks

Bank A provides a user of ″credit″ and one organization who participates in the definition of the processes for granting loans for financing properties.

Bank A writes a test scenario for the flow of activities performed in the preceding process.

Bank A writes a framework of documents to be created in the activities in the preceding process.

Acceptance Criteria

The scenario for the presentation of this application conforms to the specification defined in document X. The demonstration will be presented before the end of March 1996.

At a minimum, the application to be presented consists of the following elements:

- A complete definition of the workflow process.
- A complete user interface.
- At least one integration with the electronic document management system.
- At least one automatic creation of a document.
- At least one integration with the host system.

The following elements are not part of the acceptance criteria, but need to be developed if time permits:

- Integration of the workflow with the electronic document management system VisualInfo using OLE.
- Creation of all the generated documents automatically.

End of Project Document

2.4 The Credit Process

This section describes the workflow process for managing credit requests as it is being handled today, and as it has been defined and modeled in the prototype workflow application.

2.4.1 Logical View of the People and Roles

The following diagram illustrates the people involved in the credit process:

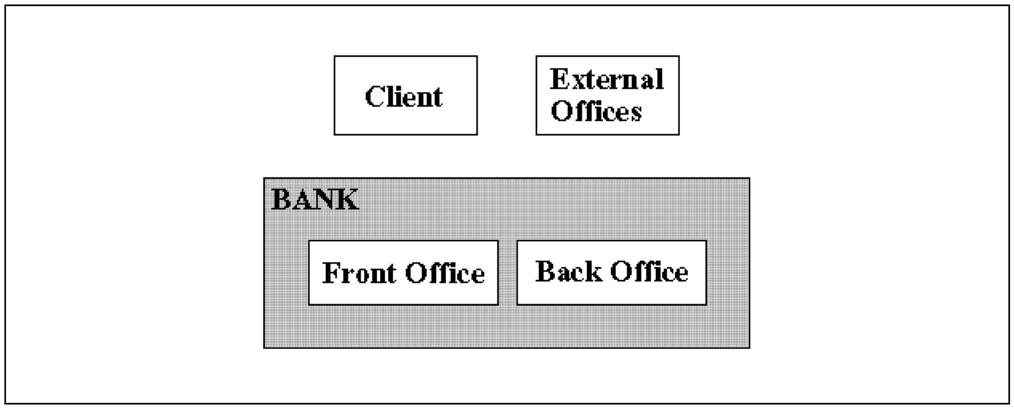


Figure 3. Logical View of the People and Roles

- **Client**

  The client is the person or company that makes a request to the bank for a loan.

- **Front-Office**

The front-office describes anyone in the branch office of a ″Credit″ organization who is responsible for collecting all of the necessary information and documents from the client.

- **Competence**

  The competence can be one person (for example, the team leader in the front-office) or a group of people who can make the decisions to accept or deny the credit.

- **Back-Office**

  The back-office describes anyone in the branch office of a ″Credit″ organization who is in charge of managing the files.

- **External Offices**

  External offices can be official organizations, notaries or anyone else who has to provide the bank with information regarding a customer.

2.4.2 The Current Credit Process

The management of credit requests in Bank A was done with paper folders and documents without any possibility of controlling the process. It was in the bank′s best interest to re-engineer their credit process so it is highly visible to their customers.

However, there was no efficient way to control or track the status of credit requests. The need to re-engineer the way the bank handles credits was not only felt by the executives in the bank, but also by the employees. Although the employees of the credit departments are aware of problems and feel the need for change, there was still a feeling of distrust for electronic document management.

The following gives a brief overview of the credit process as it was before the new IBM solution was piloted:

1. A **client** requesting credit contacted the **front-office** and provided all documents to the front-office.
2. The front-office made copies of the original documents and kept both the original and the copies.
3. The front-office sent a document to the back-office asking for further information on the client, such as land register.
4. The **back-office** routed this document to the appropriate **external offices** such as the land registry.
5. The external offices sent back the necessary information to the back-office.
6. The back-office copied the received documents and sent the copies to the front-office.
7. Based on all of the information gathered, the **front-office** wrote a credit proposal and presented this to the **competence**.
8. The competence made the decision to approve or reject the credit based on all of the information received from the front-office.
9. After a positive decision from the competence, the **front-office** wrote a formal offer for credit and sends the contract to the **client**.

10. The front-office made copies of some documents in the client′s folder and sends the folder with all of the original documents to the **back-office**. The front-office kept the copies for other purposes.
11. The client signed and returned the contract to the **back-office**.
12. The back-office sent the contract to the **front-office** for checking the client′s signature.
13. The front-office checked the signature and, if OK, added the signed contract to the client′s folder.
14. When the client′s folder was received by the **back-office**, the back-office controlled the protocols and documents with the bank rules.
15. If the back-office decided that the client′s folder was ″incomplete″, then the folder was sent back to the **front-office**.
16. The front-office contacted the **client** for the missing documents.
17. When the front-office received all of the missing documents from the client, the front-office added the documents to the client folder and sent the folder back to the **back-office**.

    The client folder may be sent back and forth between the front-office and back-office several times before the back-office decided that the folder was ″complete″.
18. Completed client folders were archived by the back-office.

2.4.3 Problems in the Current Process

- Ideally, only original copies of the documents should exist. However, many copies of the documents were generated and distributed during the entire process.
- One quarter of all approved client folders arriving at the back-office were ″incomplete″.
- The controlling of the client folder was performed too late in the process because the bank rules were not being followed by the employees.
- For ″incomplete″ folders, the folder was physically sent back and forth between the front-office and the back-office. This resulted in unnecessary loss of time as the front-office and the back-office are usually in different locations.
- Whenever the back-office asked the front-office for missing documents from the client, the front-office had to contact the client and ask for these missing documents. This resulted in a loss of the bank′s image to the client, as it was incomprehensible to the client why the bank did not require all of these documents right at the beginning instead of asking for them after the credit agreement had already been signed.

2.4.4 The New Credit Process

In the new Credit Process, activities with no added value have been eliminated such as the routing of the signed contract from the back-office to the front-office for checking the signature. This is now done by the back-office. Most importantly, the control of whether the client folder is complete is now done by the workflow system.

The process can be divided into four large phases:

- The collecting of input information from the client.
- The building of the client folder, its analysis, and the decision by the bank.
- The proposal phase where the contract is sent to the client and the notary public.
- The signing of the contract by the client and the receiving of the letter from the notary.

The new credit process will be executed in the following sequence:

1. The process starts when a **client** contacts the **front -office** and requests credit or a loan.
2. The front-office needs to collect all of the information regarding the client to estimate the financial stability of the client.
3. To do this, the front-office contacts **external offices** (such as the land registry office or real estate expert) for information
4. Once all of the necessary information has been collected from the client and the external offices, the front-office prepares the credit proposal consisting of the terms and conditions and presents it to the **competence** for a decision. This proposal includes, among others things. the client′s requested credit and information on their financial situation.
5. If the competence approves the credit, the **front-office** creates a formal offer and sends the contract to the client and to the notary. Then the competence sends the entire client folder to the **back-office**.
6. If the **client** accepts the proposal, they sign and return the contract.
7. The **back-office** receives the signed contract from the client and the notary, and does the following:
   - Checks the client′s signature.
   - Adds the signed contract to the client′s folder.
   - Controls that all other necessary documents are in the folder.
   - Scans all of the documents.
   - Archives the paper documents.

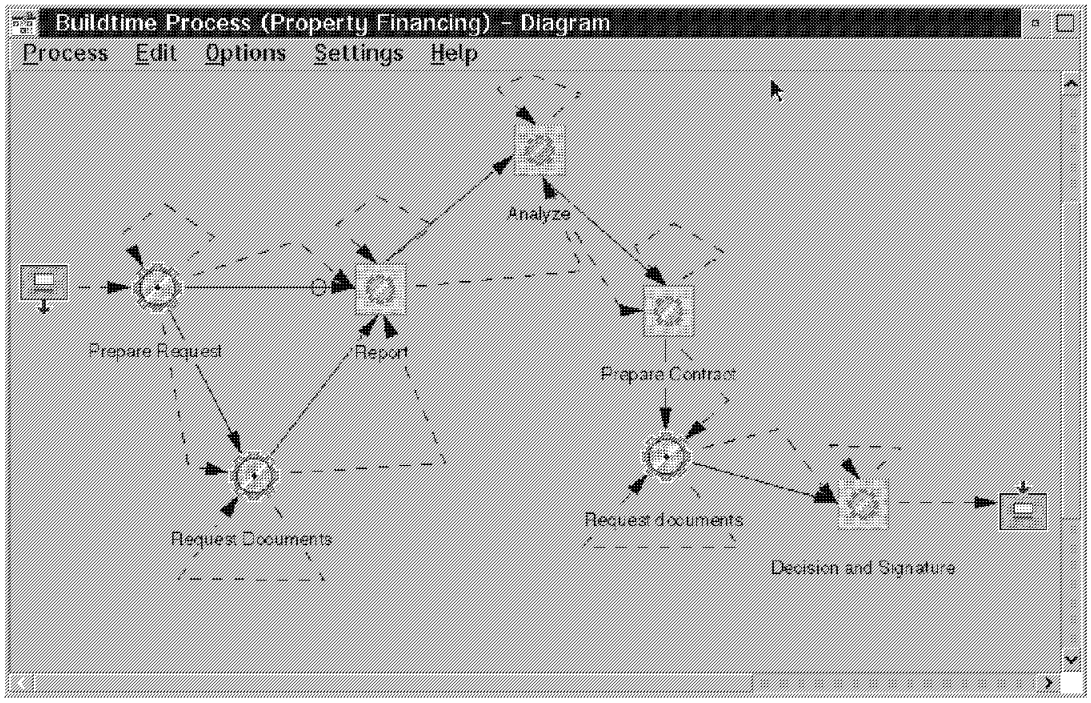


Figure 4. Credit Process

2.5 Solution Overview

The solution for the new process consisted of a workflow manager (FlowMark) for modeling and managing the workflow process, a document management system (VisualInfo) for scanning paper documents and archiving them as images in electronic folders, common desktop applications for text processing, spreadsheet applications, and a terminal emulation software for retrieving from or updating data on legacy systems. Additionally, you need interface programs to integrate the single components using their API functions, DDE, OLE, or some similar techniques.

In the new process, all the steps in the process are controlled by the workflow system, so that end-users are provided a list of their ″to-do″ items in their work lists. The work lists provide a single point of access to all the data and applications the users need to start and complete the work items in the work list. Double-clicking a work item from the work list opens an application dialog-box where all necessary data concerning the specific process are presented in a user-friendly notebook type form. Access to specific electronic documents and folders needed to complete the step are also provided from this dialog-box.

This section discusses the products that were used or integrated in the pilot system:

- Workflow system: FlowMark
- Document management system: VisualInfo, VisualInfo Client Application
- Desktop applications: Microsoft Office Standard and Lotus Smartsuite
- Terminal emulation: EXTRA! for Windows

The development environment for the prototype consisted of two OS/2 workstations and three Windows (3.11, 95, and NT) workstations. One OS/2 workstation was the FlowMark and VisualInfo server, and all of the other workstations were FlowMark and VisualInfo clients. The server workstation had the following software installed:

- FlowMark runtime and database server,
- FlowMark buildtime and runtime client,
- VisualInfo library and object server
- OS/2 Warp, DB2/2, CSet++

On the Windows workstations, the following software was installed:

- FlowMark Runtime Client
- VisualInfo VisualView Client
- VisualBasic 4.0
- MS Office Standard and Lotus Smartsuite
- 3270 emulation EXTRA!

The following diagram shows how these products were integrated on the Windows platform:

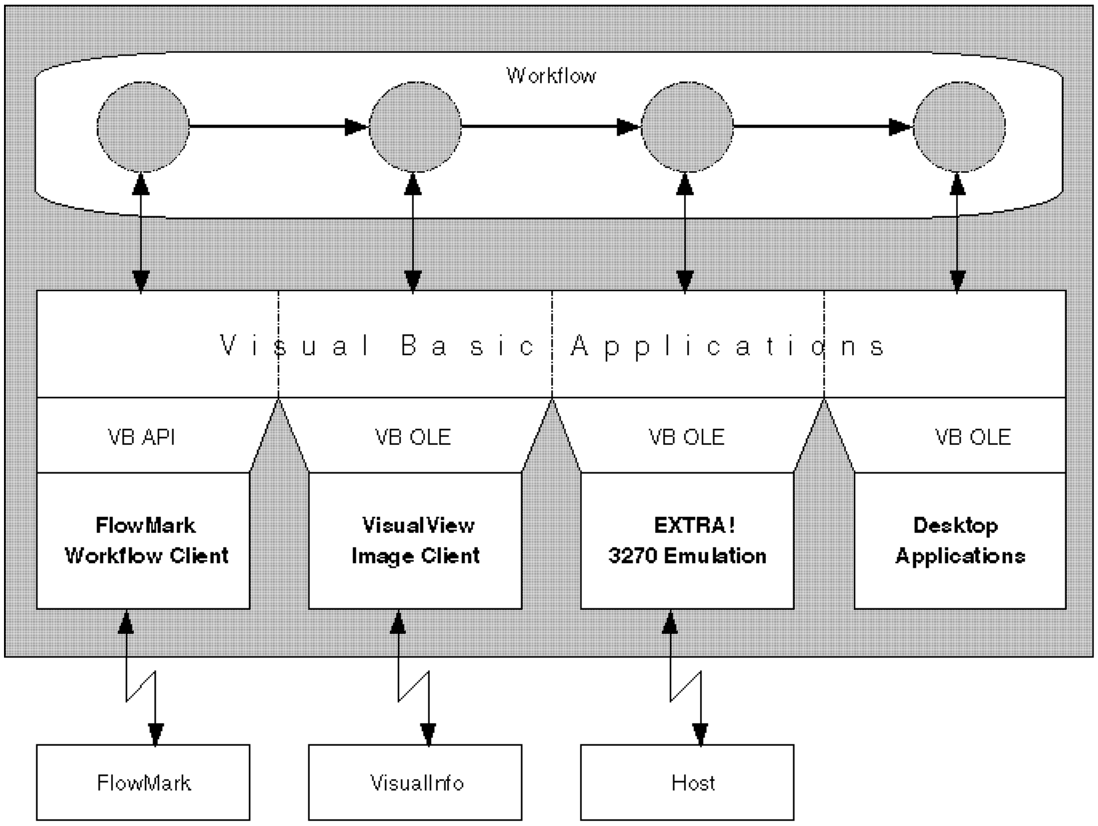


Figure 5. Solution Overview for Windows Prototype

The following diagram shows the solution that was developed in the first prototype on the OS/2 platform:

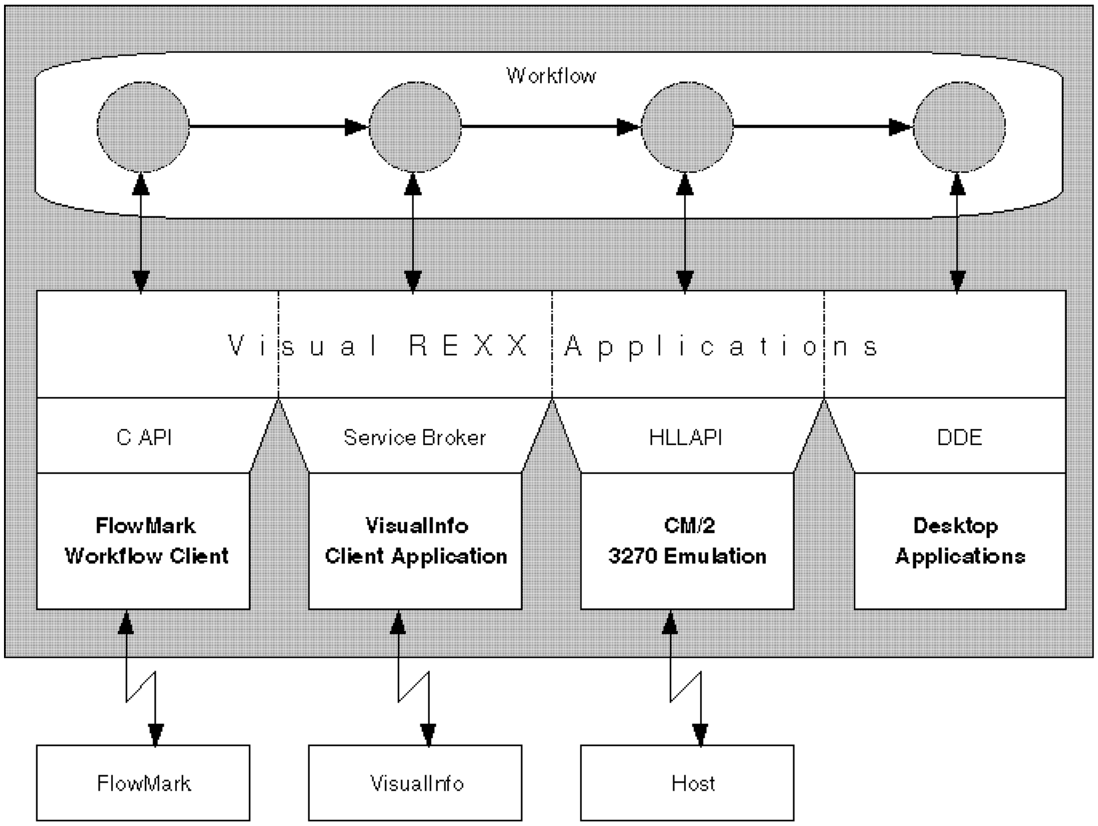


Figure 6. Solution Overview for OS/2 Prototype

2.5.1 Project Members

During the project, there were 13 people involved in the project, including the IBM and customer team members. The number on site at any one time varied according to the project phase.

From IBM, the following people were involved:

- An IBM executive (member of steering committee)
- A project manager on the IBM side (member of steering committee)

  This manager was the coordinator of the IBM project team, and was responsible for the project planning, documentation, reviews, and contact to the customer. The manager coordinated the weekly review of the project status with the customer at the customer site, where the prototype was being developed. Also, the weekly internal IBM project meetings were managed and documented by the project manager.
- A solutions marketing representative (member of steering committee)

  The representative initially offered the customer the IBM solution together with the IBM marketing representative who was responsible for this particular customer. The representative kept the communication with the customer open, and had to control the political side of the project.
- A solutions architect

The architect was responsible for the technical design of the solution. The architect was the one who contacted the right people to get the latest code of the different software that was being used (where some was at ″beta″ state). The architect also designed and modeled the workflow process in FlowMark.

- An industry advisor from the Banking vertical

  This person was a valuable member of the team, since the advisor understood the banking terms, and the way banks generally work in the credit area. The advisor analyzed the requirements and made them understood by the solutions architect and the other team members. The advisor worked together with the solutions architect in the modeling of the processes, and the design of the user interface of the applications.

- An image specialist

  This person analyzed the bank′s existing customer folders, and wrote a report on the requirements of the image capturing process that covered the bank′s needs to capture and file these documents into VisualInfo. This was a project that ran in parallel to the prototype workflow system that was developed.

- Three specialists for VisualInfo and FlowMark

  They were responsible for the installation and configuration of FlowMark and VisualInfo (or VisualView) on the development workstations (OS/2, Win 3.11, 95, and NT) and the OS/2 server workstation. They were also the programmers in the project.

The project team of the customer (the service provider) consisted of the following people:

- An executive (member of steering committee)
- A project manager on the customer side (member of steering committee)
- An application developer responsible for the design and implementation of the Visual Basic applications.
- A specialist who worked together with the IBM VisualInfo specialist to define the index classes in VisualInfo

The IBM and customer project members worked together in a team to successfully do the prototype system in the given time frame of six weeks.

2.5.2 Current Project Status

What were our plus points against the competition in the eyes of Bank A and the service provider?

- An external database is not a ″must″ for FlowMark. All other competitors had as a prerequisite an external database, which in the eyes of the customer was not acceptable for the first prototyping stage, as they did not want the overload of having to manage this database system as well.
- More importantly, the Bank believes we know their problems. Our ″honest″ attitude is well accepted by the customer.

What is the scope of the project? The scope is not yet defined exactly as this prototype is just to show the technical possibilities, but probably will not go into production as is. However, currently there are around 350 credit requests per month for five groups (front-office). Each group has around six people. This adds up to around 30 users for the front-office, 10 users for the back-office and 10 for all other type of users. The duration of these credit request processes vary from less than two months, to two-to-three months, to over three months.

How stable are the requirements? Requirements are not clearly defined. During the entire prototyping, requirements changed daily parallel to development efforts. For the prototype system, a formal discussion with the end-users took place, where the front-office, back-office, and the logistics (support) departments for credit applications sat together for the first time. This led to completely new requirements that differed from those for the first prototype.

What are the major problems in understanding the requirements and designing the system? The problem is in clearly identifying what is the process logic and what is the application logic. Currently, the banking specialist in the IBM project team who acts as a business consultant is helping the service provider to understand why the users have such requirements and how to present the solution to the users so that it corresponds to their needs. The specialist is helping the service provider to better understand the user requirements and to clarify their view.

One of the problems we faced was that the end users were not our direct customers. The information needed to understand and carry out the solution was filtered by the service provider and routed to our solutions architect. This resulted in inconsistent information when our team did have discussions directly with the end users.

Another reason for this situation was that the service provider had to find a unique solution for three banks and he was inexperienced in this application area.

Did the customer decide to implement a solution involving FlowMark, VisualInfo and IBM? Yes. They plan to build this solution using Windows NT for the clients. They will begin with AIX or NT for the servers, with a possible migration to MVS in the future. Reasons for this decision include:

- Strong IBM people involvement with high skills
- IBM openness (commitment to NT and UNIX)
- IBM commitment to robustness and globalness of our products
- Enterprise solution
- Strong support from development labs of both VisualInfo and FlowMark
- IBM′s available skills in reengineering and technology

Chapter 3. FlowMark - the Workflow Manager

Let′s take a closer look at the prototype by discussing the software components, FlowMark (this chapter) and VisualInfo (next chapter).

FlowMark is an IBM solution for workflow management that helps organizations define, document, test, control, and improve their business processes. FlowMark′s workflow management functions help organizations document their business processes, control execution of those processes, and progressively improve them.

FlowMark supports a heterogeneous client/server environment. A FlowMark server running either OS/2 or AIX can be connected to FlowMark buildtime clients on OS/2 and to FlowMark runtime clients on AIX, Windows, and OS/2 environments.

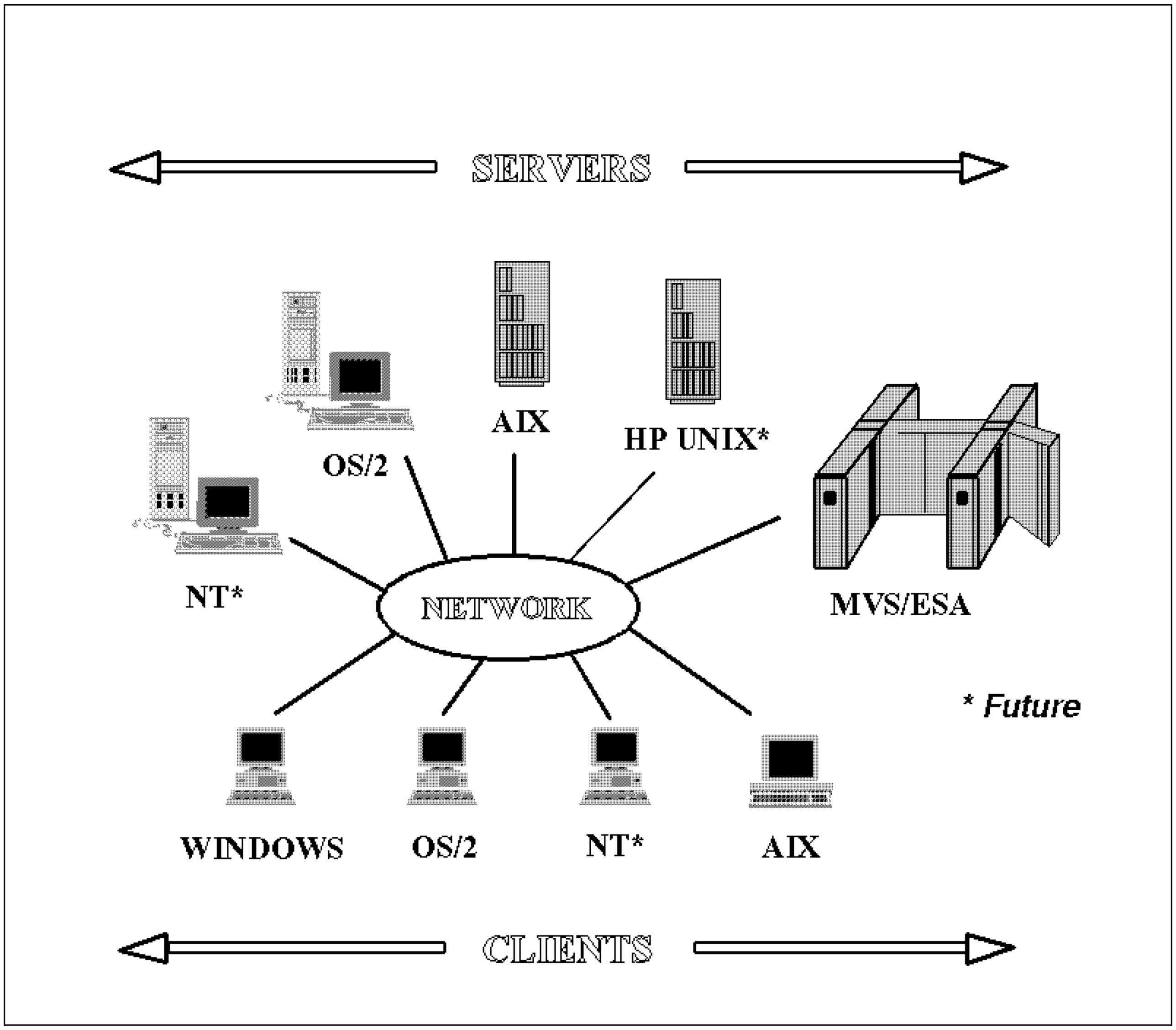


Figure 7. Platforms for FlowMark Servers and Clients

3.1 FlowMark Components

FlowMark consists of three components:

Server Functions

- Object-Oriented Database Server

  Stores the workflow information (process definitions and runtime data).

- Process Navigation Engine and Worklist Management

  Controls the flow of processes.

- Audit and Notification Services

  Creates the audit trail and notifies users of delays in the processes.

- Delivery Server

  The FlowMark delivery server is used in a process environment with distributed processes. For each database, you have one delivery server. It ensures the communication between the FlowMark runtime servers.

Buildtime Client Functions

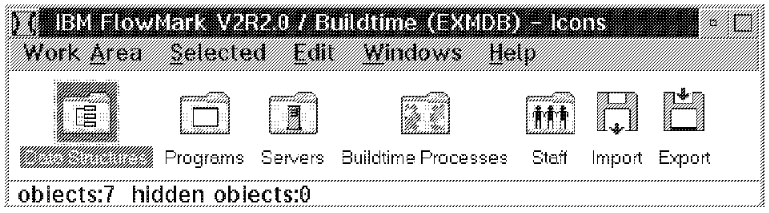


Figure 8. FlowMark Buildtime Folder

- Process Definition

  The specification of process models as activity networks.

- Staff Definition

  A definition of organizations, roles, people, and their relationships.

- Program Registration

  The registration of programs attached to activities.

- Data Structure Definition

  The definition of data structures used by the activities.

- Server Definition

  The definition of FlowMark servers for remote execution of subprocesses.

- Process Animation

  The verification of logical consistency and dynamic behavior of processes.

- Model Import/Export

  The backup and restore of buildtime definitions such as staff, programs, categories or processes is done via FDL files. These files are also used to migrate workflow models from old FlowMark releases to the current version, or to import data from business process modelling tools such as Aeneis**.

- Runtime Data Import/Export

  FlowMark Runtime Language (FRL) can be used as a report, to restore a specified point in your automated tests, or to dump the current runtime data for debugging purposes.

Runtime Client Functions

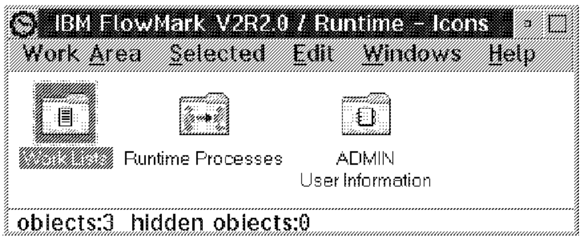


Figure 9. FlowMark Runtime Folder

- Process Execution

  Controls the execution of process instances by users.

- Worklist Manager

  The processes and activities are shown in the process list and in the worklist of the user. He is able to manipulate their state, monitor or transfer them.

- Service Broker Manager

  Makes it possible for the user to work with multiple applications invoked as FlowMark activities.

The execution of business process models does not only remind the users that there is work to do, it also simplifies their work by linking the users automatically with the required applications. Searching and starting applications is eliminated for the users. Therefore, FlowMark is able to handle jobs that mainly consist of starting programs or tools without user interaction.

FlowMark plays a central role in the design and implementation of distributed applications, consisting of component programs residing on workstations or on a host computer. It controls the execution and cooperation of these programs that may be new or may already exist, and that may run on workstations under OS/2, AIX, Microsoft Windows, or on mid-range processing on other platforms.

You can integrate applications with FlowMark using the following programming interfaces:

- C, C++, REXX, and COBOL for OS/2
- C, C++, and Visual Basic for Windows
- C and REXX for AIX

You can use the **service broker** concept for a higher level of integration. To use the service broker concept, you need a service broker DLL that is responsible for the connection, and a service DLL that provides the services you want to work with. You can use this concept to do your own service brokers and service

DLLs, or you can use the ones that are shipped with FlowMark, such as the Lotus Notes and FlowMark service brokers. Other brokers such as the VisualInfo service broker are shipped with the products. The Service Broker Architecture is described in more detail in section 5.1.4, “Service Broker Architecture” on page 48.

The **building block** concept is another possibility to integrate applications. You have to write programs that use the public interfaces of FlowMark and the application. Sample building blocks for MQSeries, HLLAPI, and DDE are already available. These are discussed in detail in chapter 5.1.3, “Building Blocks” on page 47.

3.1.1 FlowMark and ObjectStore Relationship

Between FlowMark and ObjectStore, there is a client/server relationship. The OS server belongs to the FlowMark database server. A FlowMark runtime server is the link between the FlowMark database server and the runtime clients. It acts as an OS client. Another OS client is the FlowMark buildtime client. On each workstation containing an OS client, an OS cache manger has to handle the OS client cache. Usually, the cache manager is started and stopped automatically.

3.2 FlowMark V2R2 versus V2R1

The new FlowMark Version 2 Release 2, generally available since February 1996, has the following improved features compared to the Version 2 Release 1:

- Capacity
  - Increased number of runtime instances supported.
  - Facility to pre-allocate database size.
  - More efficient database usage.
- Performance improvements
  - Faster runtime logon and refresh.
  - Faster APIs.
- Server-to-server communication
  - Subprocesses can be run from runtime servers that are attached to different databases as the originating server.
- Scalability
  - No more need for an acquaintance node.
- Profile
  - All environment settings for FlowMark are held in one file. The default profile name is exmpzcfg.prf.
- Worklist Manager APIs
  - Allow you to implement a custom-made runtime client.
  - Are available in C and C++ for OS/2, and in C from Microsoft Windows.
- New Process and Container APIs
  - Start process instance with a unique name.

    - Delete a finished or terminated process instance.
    - Query the user ID of activity starter.
    - Container APIs in COBOL.
    - New set of container APIs for faster container access.
- Service Broker Manager
    - Establishes and maintains connections to applications.
    - Available for OS/2 and Microsoft Windows.
    - Significantly improves performance for repetitively executed applications.
    - Common interface for frequently used API functions.
- Lotus Notes Integration
    - Based on the Service Broker Manager architecture.
    - Start, stop, suspend, and resume a FlowMark process from within a Lotus Notes document.
    - Check if a document is signed or encrypted.
    - Logon to a specific database.
    - Create, delete, read, update, and sign a Lotus Notes document from FlowMark.
- FlowMark for MVS
    - Enablement of FlowMark for MVS/ESA with the Application Integration Feature (AIF).
    - AIF participates as workflow initiator and as MVS Server for distributed workflow activities.
- Runtime import/export of process instances
    - Used as official migration utility.
- MQSeries Building Block
    - Can be used to establish communication to diverse IBM and non-IBM platforms.
- Realtime Notification
    - Generated at the moment the duration period is passed.
    - Tuning through an environment variable.
- Enhanced problem determination
- Installation verification

3.3 FlowMark Outlook

The following gives a brief overview of some of the new FlowMark features which are in the development plan:

- **FlowMark Lotus Notes Client** - based on worklist manager APIs
- **FlowMark NT Support** - server and runtime client
- **FlowMark HP Support** - server
- **Object Store 4.0 Support**

- **AIX 4.1.3 Support**
- **Windows 95 Support**
- **DB/2 Support** - for FlowMark runtime server
- **Simulation** -
- **Support for AS/400 FSIOP** -
  - Estimate resource requirements of a process.
  - Find the optimized process behavior.
  - Estimate the duration of a process.
  - Estimate impact of process changes.

Chapter 4. VisualInfo - the Document Manager

We have heard for a long time about the paperless society, but have not seen it. Now, using an IBM ImagePlus VisualInfo system, you can reduce the amount of paperwork handled by businesses and improve productivity by storing images of paper documents, and routing them electronically to where they are needed. The VisualInfo system can manage diverse types of information, such as images, fax, spreadsheets, graphics, word processing, audio, and video.

VisualInfo provides the following features:

- Distributed information management.
- Comprehensive APIs to enable customized applications.
- Powerful security for the enterprise through assignment of privileges and library items to users.
- Automatic migration of images (for instance, to magnetic or optical storage), which is performed by the system totally transparent to the users and based on rules you define.
- Easy to use on-line administration of data formats, fileroom, system managed storage, workbaskets, users and privileges, and utilities.
- Search and retrieval of documents and folders using index values, wildcard, Boolean logic, and full text search.
- Customizable Client Applications for OS/2 and Windows environments.
- Serial workflow through in and out workbaskets.
- Advanced workflow through a high level integration with IBM FlowMark.
- Sharing of large enterprise VisualInfo Archives with Lotus Notes Document Imaging (LN:DI) applications, which enables users of Lotus Notes business applications to archive, search, and retrieve documents to and from VisualInfo.

IBM ImagePlus VisualInfo is part of IBM′s ImagePlus document management family of products that are available on multiple platforms. Scalable servers on PC, AIX, MVS, and also flexible clients on OS/2 and Windows can be mixed in a distributed client/server environment allowing enterprise-wide document management solutions.

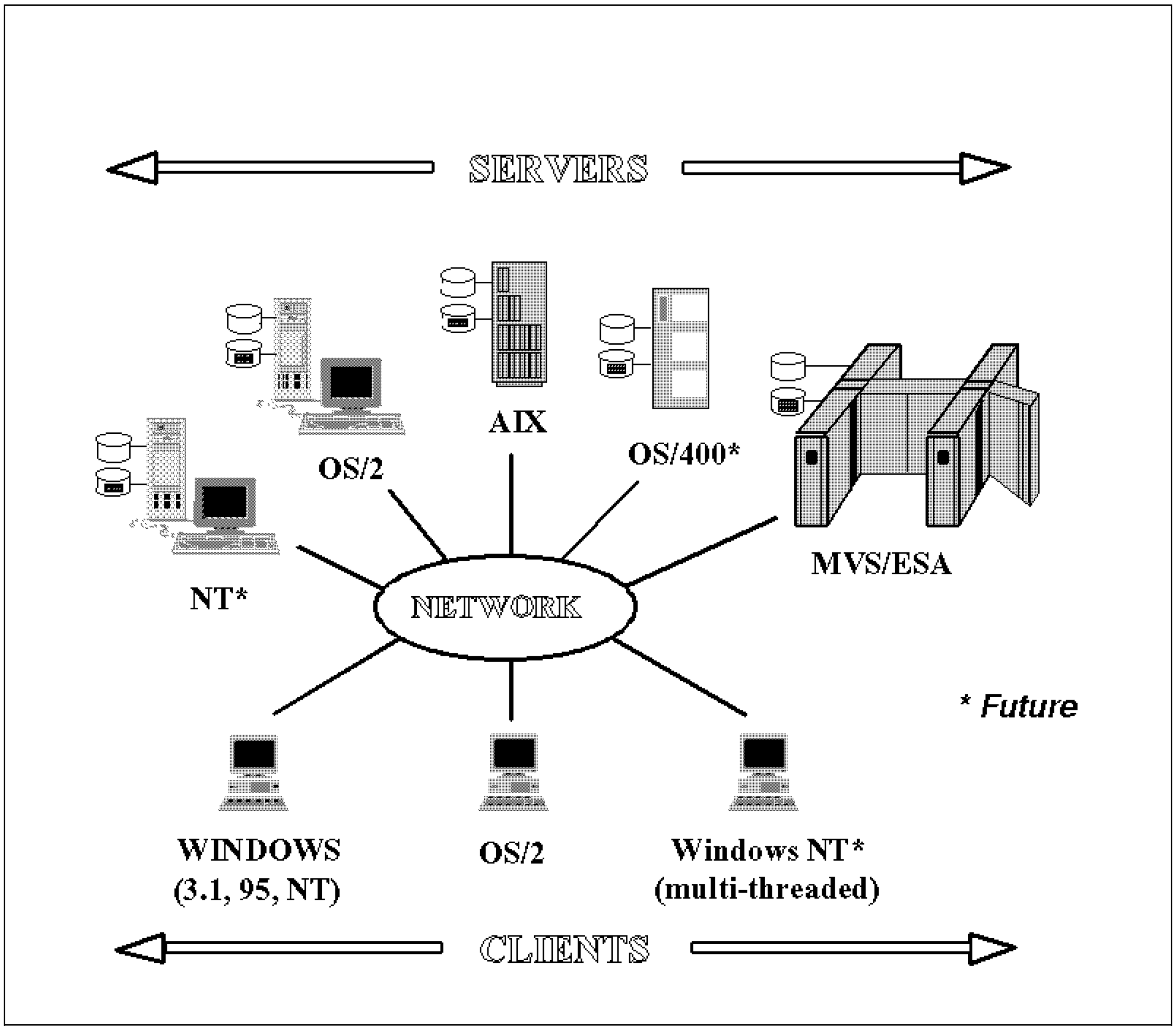


Figure 10. Platforms for VisualInfo Clients and Servers

VisualInfo works like a public library system where books and other printed material are stored and assigned unique locations. All stored information in the library is located by using a catalog that indexes the printed material by author, subject, publisher, and physical location.

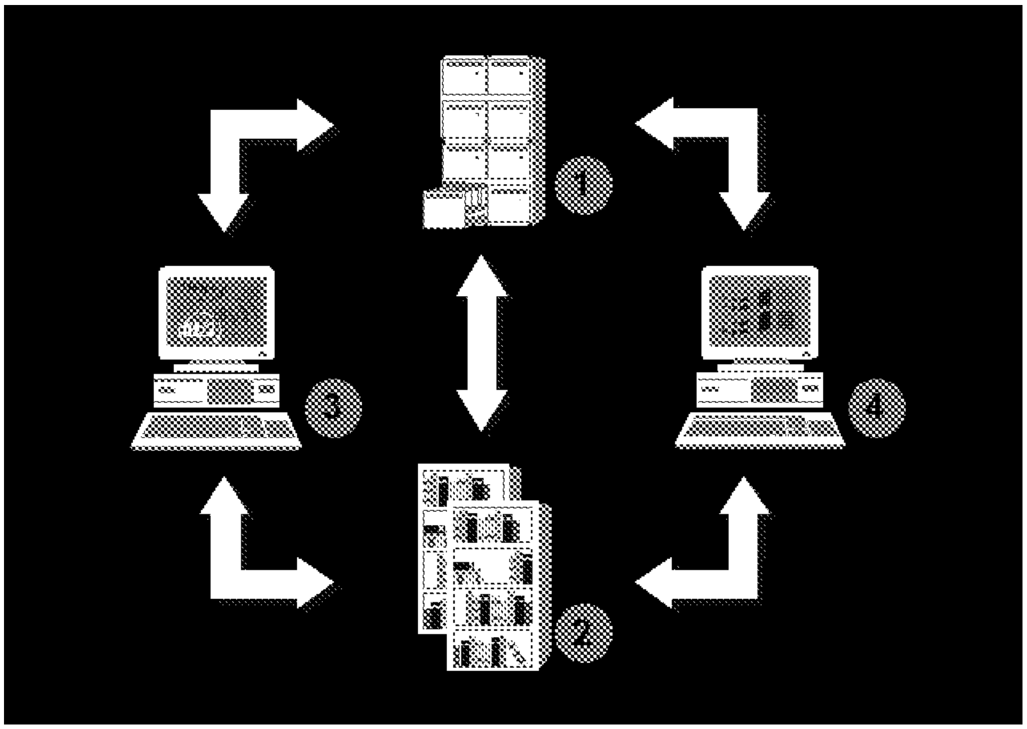


Figure 11. VisualInfo Client/Server Architecture

1. VisualInfo Library Server
2. VisualInfo Object Server (or servers)
3. VisualInfo Client for OS/2
4. VisualView Client for Windows

The VisualInfo catalog contains **Index Classes**, which are groups of **key fields** that are used to search and locate documents or images. Each key field represents a specific search criterion. To locate specific documents or images, users select an Index Class and enter data in the key field. VisualInfo uses the data entered to search for any document or image that matches the key field criteria entered. On completion of the search function, VisualInfo provides a list or Table of Contents (TOC) of the documents, images, or folders that meet the search criteria.

The following paragraphs describe the components in detail:

1. **VisualInfo Library Server**

   The library server contains the catalog. It acts as an interface between the **library client** (part of the client) and object servers by directing the library client's requests to the suitable object server. Based on this, the user is able to store, retrieve, and update documents that are stored in the **object server**. Then you can update and query the indexes and descriptive information stored in the library catalog.

   To manage library data, maintain index information, and control access to documents, the library server uses IBM DB2/2.

2. **VisualInfo Object Server**

An object server maintains the documents. It supports the attachment of both direct access storage devices (DASD) and optical storage devices. Further, it allows documents to move automatically from one storage medium to another as defined in the **management classes**.

Management classes are defined within system-managed storage. They are a collection of attributes describing backup, retention, and class transition characteristics for a group of objects in a storage hierarchy.

3. **VisualInfo Client for OS/2**

   The Client for OS/2 consists of several components:

   - **Client Application**

     The Client Application is the end-user interface for the VisualInfo system on the client workstation. It interacts with the other components of the client and includes the following functions:

     a. **Scan and Capture, Import and Export**

        A document is captured by scanning a paper document, by receiving a facsimile transmission, or by importing a document file. Scanning is the process of digitizing a document and converting it to a computer image. Existing image files can be included in the VisualInfo system by importing them. Documents and folders in the VisualInfo system can be exported to image files.

     b. **Search**

        To find specific documents and folders in the VisualInfo system, you may use the **basic search** or the **advanced search**. The basic search allows you to enter simple search criteria, while the advanced search lets you create and modify existing search profiles.

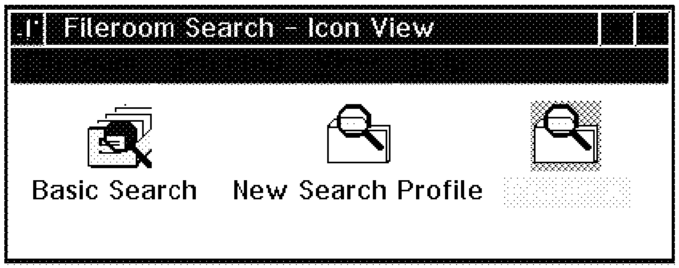


Figure 12. VisualInfo Search Folder

     c. **Display and Print**

        Display and print functions belong to the **Image Services** just the same as the scan and capture functions.

     d. **Document and Folder Management**

        In VisualInfo, you can group documents within folders, folders within folders, and documents and folders within folders, just as you do with paper documents and folders.

     e. **Security**

        The actions of users (scan, import, export, and print) on documents and folders are controlled by defining security levels. These are defined at the user, user group, or index data class level.

   f. **User Exits**

      The Client Application offers user exits where you can add your own functions to the system.

- **Image Services**

  Image Services are located on the client workstation. They enable the handling of data object types such as MO:DCA, TIFF, IOCA, Bitmaps, ASCII and PPDS, and AmiPro, Word, and so on.

- **System Administration Program**

  System administrators use this program to define and configure the system:

  a. System-Managed Storage
  b. Users and their access
  c. User Groups and their access
  d. Workbaskets and Workflows
  e. Index Classes and Data Formats

  Note: The System Administration program must be installed and run on an OS/2 workstation. It is not supported on a Windows workstation.

- **Folder Manager**

  The Folder Manager stores and organizes document images into electronic file folders for convenient access and retrieval. The user can index, process, and retrieve documents and folders, depending on the level of access.

- **Library Client**

  You can use the Library Client APIs to develop your own application that performs functions similar to the folder manager. This allows you to create an alternative data model to the folder manager. You can store, retrieve, delete, and update the VisualInfo items (folders, documents, workbaskets, and workflows).

4. **VisualInfo Client for Windows**

   The VisualInfo client for Windows (VisualView) that we used is described in the next chapter.

   Note: In the current release of VisualInfo, you can use the IBM VisualInfo Windows Client in place of VIP VisualView.

A large set of APIs lets you customize or integrate VisualInfo with your solution. The primary programming interfaces for VisualInfo are known as **CAPI**s. The CAPIs consist of a subset of the VisualInfo folder manager APIs and the Images Services APIs. These APIs are described in detail in section 5.2.1, “VisualInfo Standard APIs” on page 50.

To enhance or replace VisualInfo functions, you can use the VisualInfo **user exits**. You have to put your individual processing logic in a DLL file and register the names of the functions and DLL file in VisualInfo. For example, you may replace the search statements for a specified index class, or modify the store routine for files of a specified type (for example, ASCII file or AmiPro text document).

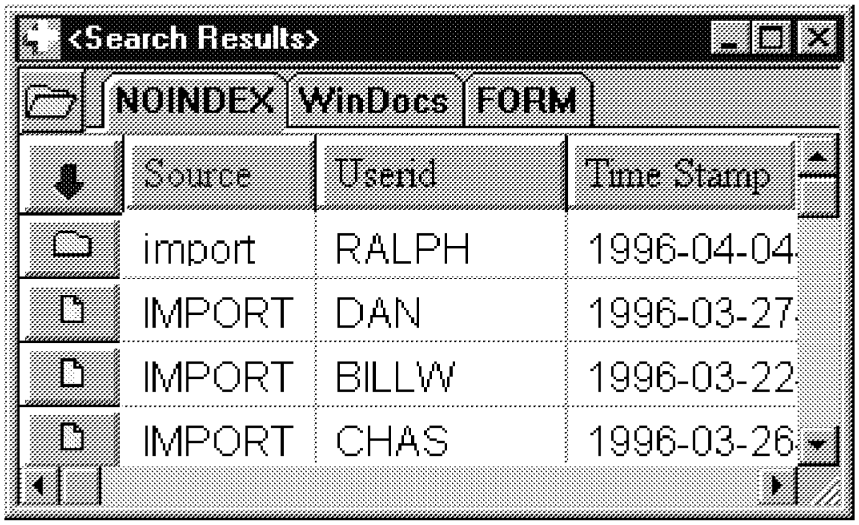
The VisualInfo High Level Programming Interface (VHLPI) allows you to easily integrate FlowMark with VisualInfo without any coding required. These APIs are described in detail in section 5.2.3, "VisualInfo High Level API for OS/2" on page 55.

Note: The Windows client does not support the System Administration program. You must have an OS/2 client workstation for this purpose.

4.1 The VisualInfo Client Application

The VisualInfo Client Application for the Windows environment provides a direct connection to the ImagePlus VisualInfo system. It offers such functions as retrieval of folders and documents, image display and manipulation, advanced annotation, and indexing. The retrieval functions allow users to search and retrieve folders and documents by full or partial key fields.

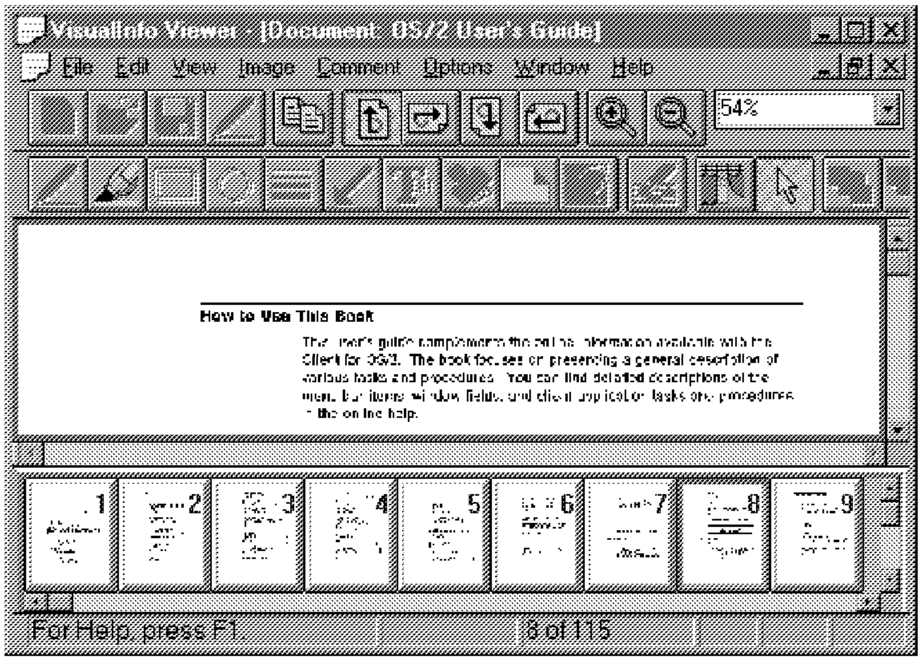
Retrieved and imported documents are found in the user′s workbasket, or can be searched for and displayed in a Search Results folder:



Source	Userid	Time Stamp
import	RALPH	1996-04-04
IMPORT	DAN	1996-03-27
IMPORT	BILLW	1996-03-22
IMPORT	CHAS	1996-03-26

Figure 13. A Search Results Folder

The VisualInfo Viewer is used to display the documents. With this window, the user is able to add notes, texts, lines, arrows and free hand drawings to the document, and also highlight areas and place stamps on it. The document can also be rotated, zoomed, or printed.


VisualInfo Viewer - [Document: OS/2 User's Guide]
File Edit View Image Comment Options Window Help
54%
For Help, press F1
8 of 115

How to Use This Book

This user's guide complements the online information available with the Client for OS/2. The book focuses on presenting a general description of various tasks and procedures. You can find detailed descriptions of the menu bar items, window fields, and client application tasks and procedures in the online help.

Figure 14. VisualInfo Viewer for Windows

VisualInfo Client Application uses the TCP/IP protocol to connect to ImagePlus VisualInfo servers on OS/2, AIX or MVS platforms.

The VisualInfo Client Application developer's toolkit is based on object oriented development techniques and **OLE automation**. This provides you with a development environment to create custom client applications. Using the OLE 2.0 automation, external Windows-based applications can to logon to VisualInfo, and perform functions such as searching for documents and folders.

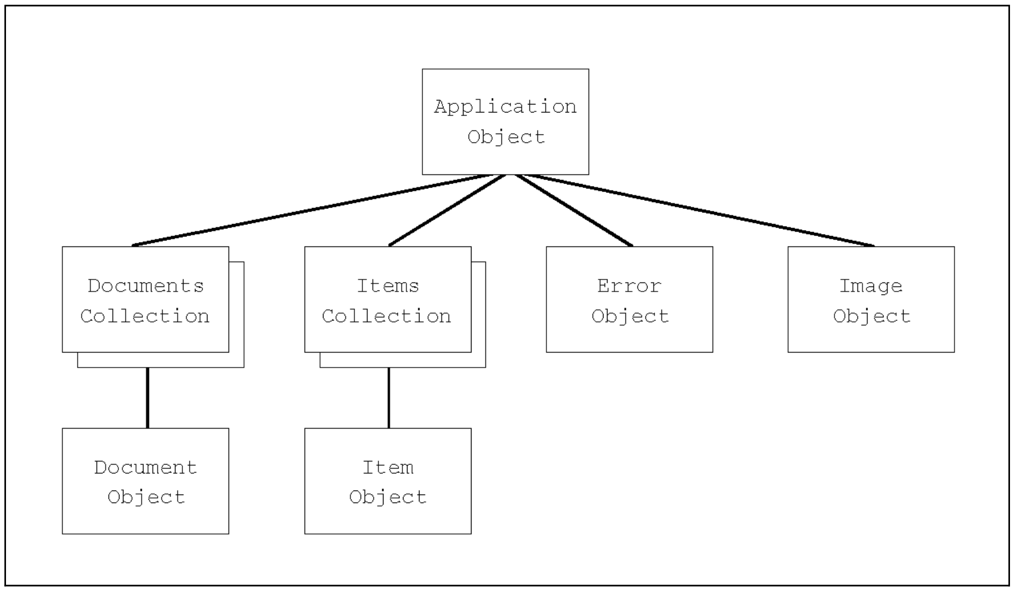


Figure 15. OLE VisualInfo Client Application Client Objects

The following OLE VisualInfo Client Application Client objects are provided (see Figure 15):

- Application Object

A program that is designed to control VisualInfo Client Application must first create an Application object. The methods and properties of an Application object apply to the application (here VisualInfo Client Application). With this object, it is possible to perform VisualInfo functions.

- Documents Collection and Document Object

  The Documents collection is a kind of queue that is able to hold Document objects. Document objects represent VisualInfo folders and workbaskets. They can be created from the Documents collections.

- Items Collection and Item Object

  An Item object represents VisualInfo items (folders, documents, and workbaskets). You can display an item, query its index class and key fields, and perform other actions on it. The Items collection is a list of Item objects.

- Image Object

  The currently visible document in VisualInfo is represented by the Image object.

- Error Object

  If an error occurs, an Error object is created that holds important information about the error, including the VisualInfo return codes.

VisualInfo Client Application also offers VisualInfo **user exits**, where you can specify your individual routines for processing. Therefore it is possible to enhance or replace default functions. Your routines need to be in a DLL file and you have to register them, so VisualInfo Client Application knows which routine to take. Some sample code showing the use of user exits is included in section 5.2.2, “User Exits” on page 51.

The predecessor to the VisualInfo Client Application was an application called VisualView by VIP. For further information on OLE and user exits, see the *Technical Reference Manual* of the VIP VisualView Client.

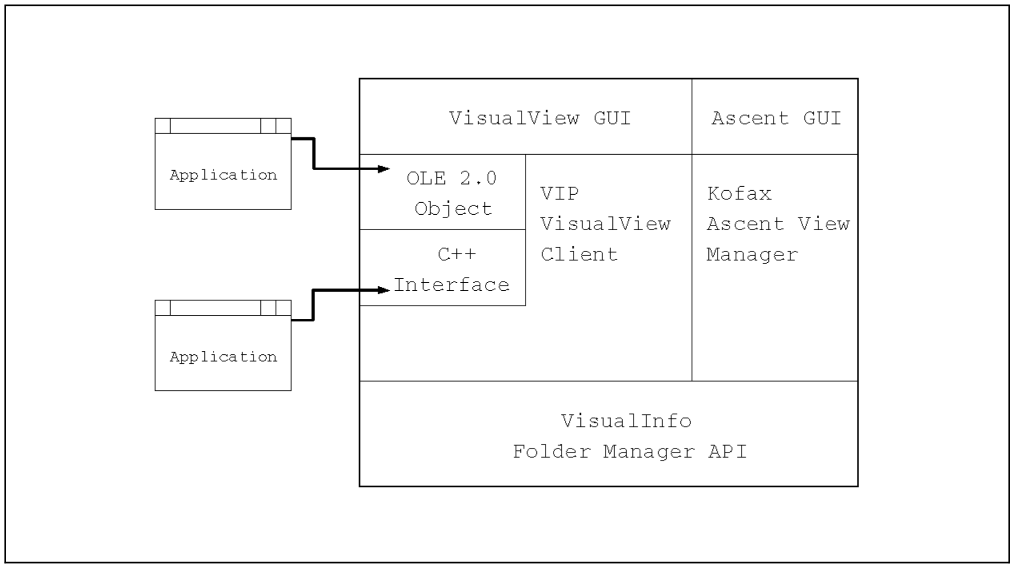


Figure 16. VisualInfo Client Application Interfaces

Chapter 5. Implementing an Integrated Solution

This chapter describes the integration techniques that we used to carry out the pilot system. Visual Basic was used to create a graphical user interface on the client workstations and to put some logic behind it. It was chosen because you can develop user interfaces very rapidly with it in a Windows-based environment. On an OS/2 platform, you may use Visual REXX or a similar product for this.

Not all of the functions can be covered by using Visual Basic or REXX programming. To do the remaining functions, it is necessary to use a more flexible language such as C or C++. Therefore, many base functions for the integration of different products in the Windows environment were written by using the Borland C++ 4.5 compiler (with OS/2 you may use the IBM VisualAge C++ compiler). It is possible to design these functions as an interface to Visual Basic or REXX, so you can easily access them when developing your user interface.

5.1 FlowMark Integration

This section describes the FlowMark programming interfaces:

- FlowMark Language APIs
- Workflow Client APIs
- Building Blocks
- Service Broker Architecture

5.1.1 FlowMark Language APIs

The FlowMark language APIs are designed to work with data, processes, and activities in a program called by a FlowMark activity. You can get data from the activity′s input container, work with it, and write it back to the activity′s output container. It is possible to change the status of an activity, as well as to start, stop, terminate, delete, suspend, resume, and restart processes.

You can work with the FlowMark language APIs using the following operating systems and programming interfaces:

Table 2. Platforms: FlowMark Language APIs

Language	**OS/2**	**Windows**	**AIX**
C	X	X	X
REXX	X		X
Visual Basic		X	
COBOL	X		

The next table lists the functions you can call from C, REXX, Visual Basic, and COBOL.

Table 3. Functions: FlowMark Language APIs

Function	**C**	**REXX**	**VB**	**COBOL**
Initialize container API	X		X	
Get a session id	X		X	
Get user id of activity starter	X	X	X	
Query size of a container data structure			X	
Query a container data structure	X		X	
Get a data item	X	X	X	X
Set a data item	X	X	X	X
Send container data back	X		X	
Begin a process control session	X	X	X	
Begin a process control session extended	X	X	X	
Begin a process control session within a process	X	X	X	
End a process control session	X	X	X	
Clone a process template	X	X	X	
Start a process instance	X	X	X	
Fast start a process instance	X		X	
Start a unique process instance	X	X	X	
Fast start a unique process instance	X		X	
Delete a process instance	X	X	X	
Suspend a process instance	X	X	X	
Resume a process instance	X	X	X	
Terminate a process instance	X	X	X	
Restart a process instance	X	X	X	
Change the state of an activity	X	X	X	

If you choose to install the APIs when installing FlowMark, then sample code for C, REXX, Visual Basic, and COBOL is added to the FlowMark API directory.

5.1.2 Workflow Client APIs

This section describes the workflow client APIs of IBM FlowMark workflow manager that have been available since FlowMark version 2.2. With the workflow client APIs, you are able to log on to or log off from the workflow server, manipulate worklists and work items, and work with process instances and container data. This allows you to build your own FlowMark Runtime client. FlowMark offers two types of workflow client APIs:

1. **FlowMark C Coalition API**

   The FlowMark C coalition API is defined by the Workflow Management Coalition (WFMC) and is independent from FlowMark. This API was implemented using the functions of the FlowMark C++ workflow client API.

2. **FlowMark C++ Workflow Client API**

   The FlowMark C++ workflow client API is specific to FlowMark. So far, there is a larger number of function calls available with the FlowMark C++ workflow client API than with the FlowMark C coalition API. See Figure 17 on page 45.

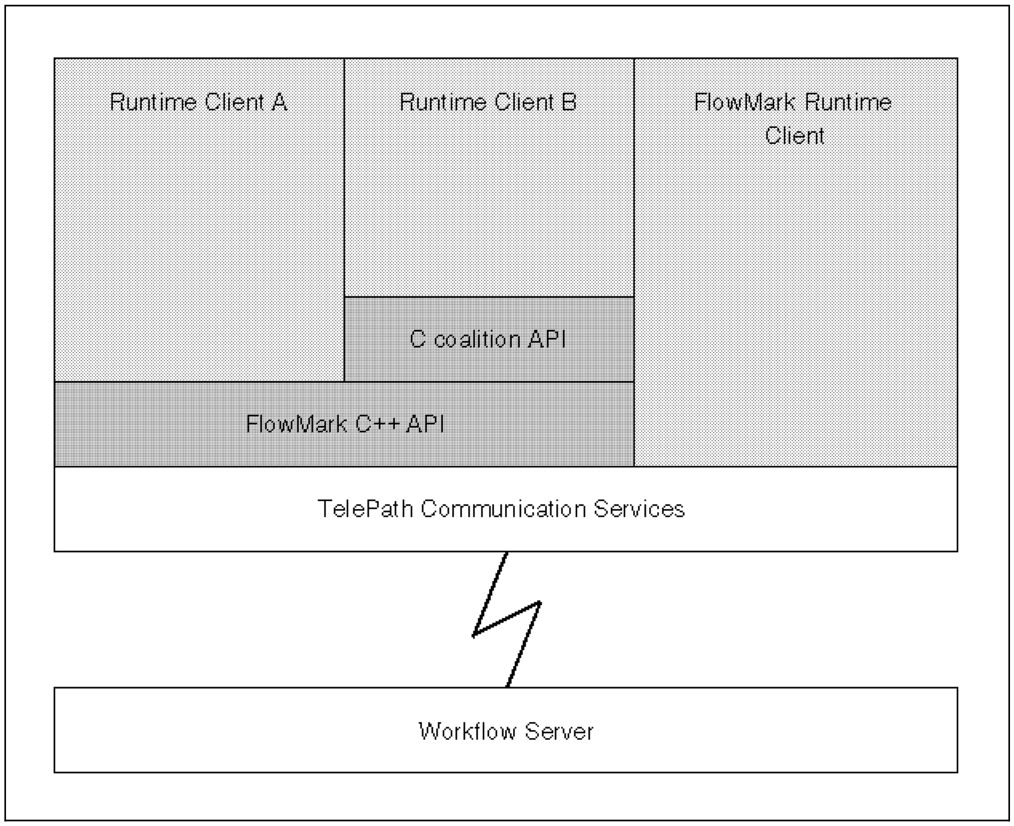


Figure 17. Runtime Clients in the FlowMark Workflow System

The FlowMark C coalition API is available for OS/2 only. It provides interchangeability of applications between different workflow engines. The function calls are grouped as follows:

- **Connection Functions**

  You need the connection functions to log on to or log off from a workflow server.

- **Process Control Functions**

  Using the process control functions enables you to change the operational state of one or more process instances in your workflow.

- **Activity Control Functions**

  The activity control functions work similar to the process control functions, but here you can change the operational state of activities.

- **Process Status Functions**

  Using the process status functions calls, you can work with process instance lists and process instances.

- **Activity Status Functions**

  The activity status function calls are used to work with activity lists and activities.

- **Worklist Functions**

  With the worklist function calls, you are able to work with work items; for instance, read work item related data or transfer a work item to another worklist.

- **Administration Functions**

  The administration functions are needed to control processes and activities.

- **Cleanup Functions**

  The cleanup functions are not defined in the Workflow Management Coalition Application Programming Interface (WAPI) specification, but they are needed to free memory you allocated in previous C coalition API function calls.

For more information, refer to *IBM FlowMark: Programming Guide* and *Workflow Management Coalition Application Programming Interface 2 (WAPI) Specification*.

The FlowMark C++ API is available for the OS/2 and Windows platforms. There are two things you have to remember:

1. For Windows, the FlowMark C + + API must not be called before the message loop in your application; this is, for example, in LibMain or WinMain. Always call these APIs after the initialization of your program.
2. The FlowMark C + + API uses the standard template library (STL) that provides a vector template. You have to define the default constructor for all classes that you want to construct a vector of. For all FlowMark C++ API classes, except for ExmServer, this default constructor is provided.

Three kinds of methods are provided by the FlowMark C++ API:

1. **Basic Methods**

   The basic methods are used in each FlowMark C++ API class. These methods are needed to construct, destruct, copy, compare, and assign objects.

2. **Accessor Methods**

   The accessor methods enable you to read data from transient objects. This data can be read as long as the transient object exists, regardless of the corresponding persistent object and the connection to the server. Methods such as ″Name()″ or ″IsEmpty()″ are accessor methods.

3. **Action Methods**

   The action methods can be used to establish a server connection and to create or update transient copies of persistent objects. Accessor methods operate on these transient copies. Methods such as ″Logon()″, ″QueryWorkitems()″, or ″Delete()″ are action methods.

The following list is an overview of the FlowMark C++ API classes:

- **ExmServer**

  You can establish a server connection, change the password for a user, and query instances, templates, and worklists for the logged on user with the methods provided by ExmServer.

- **ExmProcessTemplate**

  The action methods of ExmProcessTemplate enable you to create instances of process templates and delete or refresh templates. Using the accessor methods, you can get information such as the description, name, or category of a template.

- **ExmProcessInstance**

It is possible to query and set the state of process instances, and query the instance′s properties such as start time or starter.

- **ExmProcessInstanceNotification**

  The ExmProcessInstanceNotification class allows you to work with notifications related to process instances. For example, you can query if the maximum duration time of a process instance is already expired.

- **ExmWorklist**

  You can delete the worklist represented by your ExmWorklist object from the runtime′s database, or query work items (activities), or notifications related to instances and workitems.

- **ExmWorkitem**

  ExmWorkitem class provides methods to query the properties and containers of an item, and delete or change the state of an item.

- **ExmWorkitem Notification**

  The ExmProcessInstanceNotification class lets you work with notifications related to work items. For instance, you can get the level of escalation (first or second escalation).

- **ExmContainer**

  The ExmContainer class is the superclass of ExmReadOnlyContainer class and ExmReadWriteContainer class. Their methods enable you to work with the input and output containers, such as reading a value from the input container.

- **ExmContainerElement**

  You can work with container elements (count the members and query the type) and arrays of container elements using the ExmContainerElement class.

- **ExmFilter**

  The ExmFilter class is used to specify filter criteria, that you want to use when, for example, querying work items.

- **ExmDateTime**

  The ExmDateTime class provides methods to handle ExmDateTime objects, for example, query minutes or seconds.

A sample program that uses the FlowMark workflow client C++ APIs is included at the end of this book in Appendix A, “FlowMark C++ API Sample” on page 95.

For more information about the FlowMark C++ API, refer to the IBM FlowMark: V2.1 Programming Guide, SH19-8240.

5.1.3 Building Blocks

The building block concept provides the possibility to write separate programs for the integration of FlowMark with another product, using their published interfaces.

The building block for MQSeries support can be used with FlowMark to start, suspend, resume, and terminate a FlowMark process on another FlowMark system by using MQSeries. Usually, MQSeries makes it possible to communicate between applications that run on the same or on a different platform. The applications then use queues provided by MQSeries to exchange data.

A sample on how to use the MQSeries building block with FlowMark is also installed when you install FlowMark. This sample includes:

Table 4. MQSeries Building Block

File	Description
exmp2abb.fdl	FlowMark process model
exmp2abb.mqi	MQSeries definition
exmp2asd.exe	Start and execution control (restart, resume, terminate, suspend) of remote processes; return data from a child process to the remote parent process.
exmp2asp.exe	Suspend own or current process.
exmp2arm.exe	Suspend FlowMark processes in OS/2 and AIX environment.
exmp2arv.exe	Write data to output container.
exmp2asv.exe	Start and execution control (restart, resume, terminate, and suspend) of local processes.

For a detailed description of this sample, refer to *IBM FlowMark Application Integration V2R2, SH12-6267*. For more information about MQSeries, refer to *MQSeries Distributed Queuing Guide, SC33-1139* or *MQSeries Command Reference, SC33-1369*.

5.1.4 Service Broker Architecture

The Service Broker Architecture 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. The aim is to allow 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 **Service Broker Manager** controls the operations of **service broker** sessions and the interaction between the **service requester** and **services**. The Service Broker Manager is part of the FlowMark product. The service brokers for FlowMark and Lotus Notes are also included in FlowMark. The service broker for VisualInfo (VHLPI) is provided with the VisualInfo product and is described in detail in the next chapter.

For any other base product or application that you want to integrate in your workflow system, you may write your own service broker for that product or application using the Service Broker Architecture. For information on how to write your own service broker, refer to the *IBM FlowMark Application Integration V2R2, SH12-6267*.

Furthermore, you can implement a customized service requester to call service functions of the already existing service brokers if they expect a special input or output format.

The following diagram shows how you can integrate FlowMark with other applications using the service broker concept. Service brokers for Lotus Notes and VisualInfo on OS/2 are used as examples.

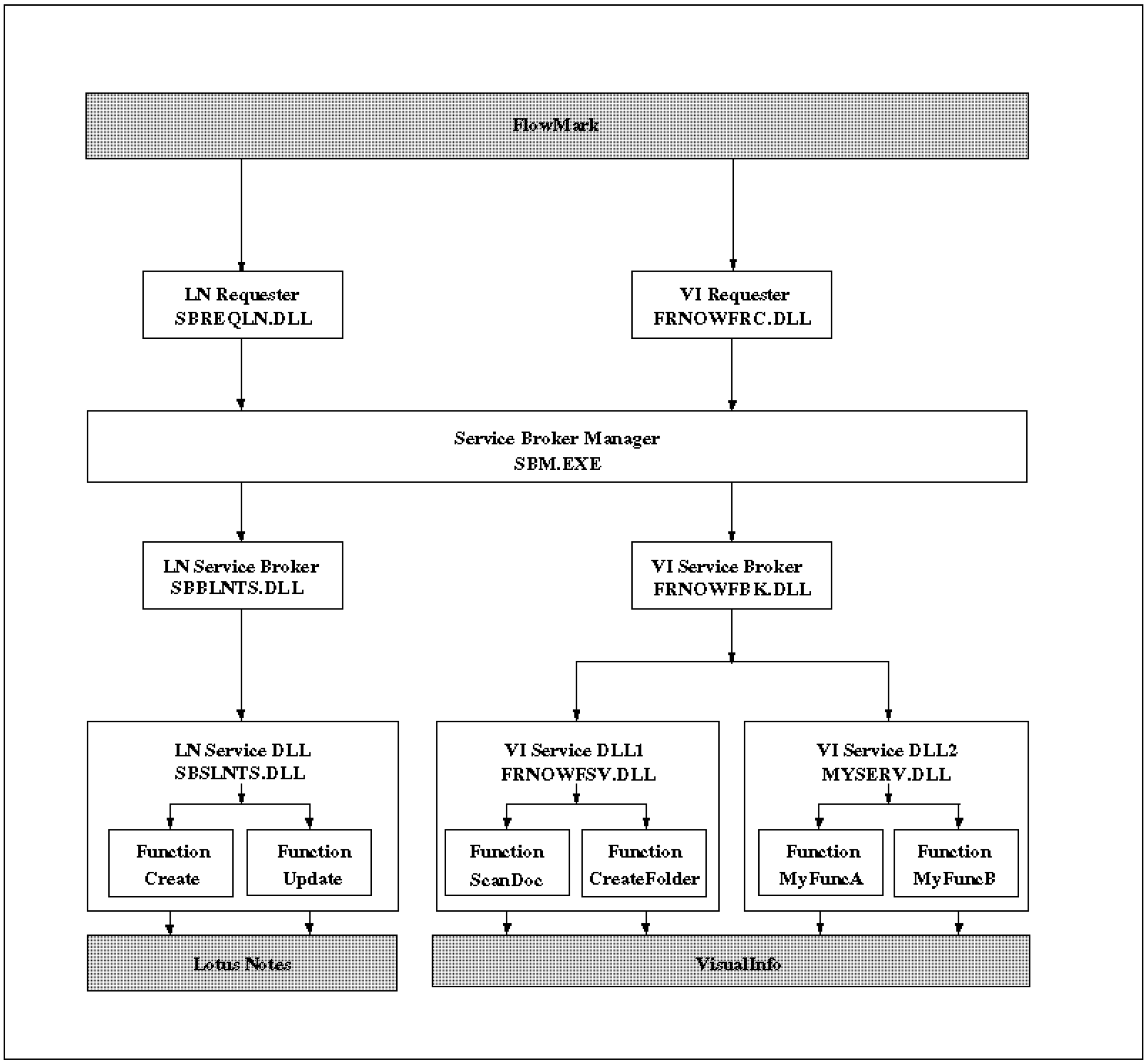


Figure 18. Using the Service Broker Concept to Integrate FlowMark with Other Applications

To minimize the number of simultaneous connections to the server, connections are established by a service broker. When this service broker DLL is started, it establishes the connection to the server of the base product, and keeps the connection open when the service broker is running.

The **service broker** can share this connection with several services through a shared structure that is accessible from all registered services of this particular service broker. Whenever a service functions is called, it can use the existing connection and does not need to log on again.

A **service function** is a subroutine that provides a specific service. For a Lotus Notes service function, this might be to create a document or to read a document. Several related service functions can be stored within one service DLL. Some standard functions are useful in a wide range of situations, while others are specific to a certain environment. You may prefer to write your own service DLL that carries out enhancements and extensions to the service functions of an existing service broker. This is illustrated in the diagram with the

VisualInfo service broker (MYSERV.DLL). The Service Broker Manager can simultaneously manage several services for a service broker.

A **service requester** is the interface to the user application. The user application calls the service requester to request the product to perform some work. The service requester formats the user data and issues a request to the Service Broker Manager which sends the request to the appropriate service function.

For information on using the Service Broker Manager and how to program your own service brokers, refer to the *IBM FlowMark Application Integration V2R2, SH12-6267*.

5.2 VisualInfo Integration

This section discusses the programming interfaces of VisualInfo:

- VisualInfo Standard APIs
- User Exits
- VisualInfo High Level API for OS/2 (and for MS Visual Basic)
- VisualInfo OLE Automation API for Windows

5.2.1 VisualInfo Standard APIs

VisualInfo provides a rich set of API functions you can use in your programs. The APIs are grouped as follows:

- Folder Manager, Application, and Library Interfaces

  These interfaces include the Common APIs, Folder Manager APIs, Client Application APIs, Interchange APIs, System Administration APIs, Library Server APIs, Object Server APIs, List Manager APIs, and Device Manager APIs. For example, you can use functions to index, store, or search documents, and also use functions to manage access control.

- Image Services

  1. Environment Services

     The environment services are used to initialize image services, describe the current environment, and memory management (allocating and deallocation).

  2. Working Set Services

     You use working sets as a container for documents and other data managed by your application. The working sets are the base building blocks of the image services. Once you have created a working set, you can add, view, manipulate, and store pages and documents using the image services APIs.

  3. Display Services

     The display services provide tools to present data to the end users. You can create and manage display windows that enable the end users to view and manipulate data objects in the working set.

  4. Scan Services

     Scan services provide methods to scan documents into a working set. You can use functions to set up and initiate the scan operation. Also the scan services offer an end user interface that allows you to have operational control of the scanner. Note that the scan services are device-independent.

5. Print Services

   With the print services, you can use a basic set of APIs to set up and initiate the printing operation. The end user interface of the print services provides the definition of a printing profile. Note, the print services are device independent.

6. Keystroke Routing Services

   The keystroke routing services enable you to create accelerator keys to route key operations between your application windows and any image services window.

For a detailed description of VisualInfo′s programming interfaces, refer to the *IBM ImagePlus VisualInfo Application Programming Guides for Windows, OS/2 and MVS* (SC31-9055, SC31-9059 and SC31-9060), and *IBM ImagePlus VisualInfo Application Programming Reference* manuals (SC31-9061 through SC31-9063).

5.2.2 User Exits

To customize an existing application, you can often use user exits. A user exit is a specific point in an application where your routines are called for processing. Here we explain how you can work with user exits using a VisualInfo example.

The following user exits refer to the VisualInfo library server. You can use them to specify your routines for access control, password encryption, definition of static queries, and task priorities.

- Library Access Control

  There are two user exits for access control: LibACUserExitOne and LibACUserExitTwo. These user exits are used by the internal reference monitor during the processing of access control. The first one is called when the internal reference monitor makes a decision before processing the access control lists, and the second one is called when the decision is made after processing the access control lists.

- Password Encryption

  Passwords are encrypted by the library client before they are transmitted to the library server. If you want to replace the encryption with your own algorithm, you have to use LibEncryptPassword. You can pass your own encrypted string to LibEncryptPassword, and the library client sends your string to the server, or you can use the default encryption by passing a null pointer to LibEncryptPassword. If you do not want to have any encryption of the password, you have to pass the non-encrypted password to LibEncryptPassword.

- Static Query

  Using LibUserQueryExit, you can start your application specific static queries. You have to define tables that contain the host variables used in your query and a column description of the columns returned by your select statement. You also need to provide functions for handling the cursor (declare, open and close) and fetching data.

- Task Priority

LibSelectLibraryTransPriority allows you to set the CICS transaction priority for host library server requests.

The following list of user exits refers to the VisualInfo Client Application APIs:

- Alternate Search

  Using the AlternateSearchUserExits, you are able to replace the Client Application′s default search routine. The type of search will determine when your user exit is loaded. If you define the search against a particular view, the Client Application loads your function when running a basic search against that view. Your function is loaded for the base view of the NOINDEX class if you run an advanced search, or a basic search if you have defined the search against all views.

- Determine Next Workbasket

  This exit is associated with an index class and it is called:

  1. Whenever the user chooses the option ″Route to″ from the ″Process″ menu for an item that has the exit defined in its index class.
  2. Before displaying the ″Route to″ dialog.
  3. When the user chooses the options ″Start workflow″ or ″Change workflow″ from the ″Process″ menu.
  4. Before the actual routing of an item.

- Change System-Managed Storage

  The user exit ChangeSMSUserExit can change the system-managed storage for each part of an item. It is called if the index class is changed for an item before the library object window is closed.

- Determine Workflow

  If an index class is defined to automatically start items in a workflow when a document or folder of that index class is saved, then the DetWorkflowUserExit is called. If the item has been in any workflow before, this user exit is not called.

- Overload Trigger

  If the overload condition is triggered for a workbasket, the OverloadTriggerUserExit is called. In the system administration program, you can specify the maximum number of items allowed in a workbasket before the exit is called. This number is the overload trigger.

- Query Sort

  Using QuerySortUserExit, you are able to define a sort order other than the default ascending or descending order. You may use this user exit also to filter out documents and folders that you do not want to display.

- Save Record

  The SaveRecordUserExit is called when you try to save changes to user-defined attributes of a document or folder. You can validate the entered data or change the user-defined attribute fields.

- Index a Document from Working Set

  To index a document and optionally move it to a new workbasket, use the Ip2ExitIndexDocument user exit. This exit is only called when you work with advanced scan.

- Perform Work on Document after Scan and Index

  When a document is scanned and indexed into the NOINDEX class, the Ip2ExitUseDocumentData user exit is invoked. You can re-index the document or place it in a new workbasket. It is also possible to perform some processing based on the item ID, the scanner data field data, or the MGDS data.

Use the following object server user exit to influence the flow of the storing process. We recommended that you write these user exits as reentrant and call them at the process level.

- LAN-Based Object Server

  Use Ip2LBOSExit to decide if a specific volume is used to store the data. It is possible to reject a volume for only the current storage or for future ones.

You can use the interchange user exits to input documents and folders from other platforms to your VisualInfo system.

- Match Import Attributes

  Using the Ip2ImportAttrExit, you can specify which VisualInfo attribute name is to be used instead of the attribute name found in the CIF header.

- Match Import Classes

  Using the Ip2ImportClassExit, you can specify which VisualInfo index class name is to be used instead of the index class name found in the CIF header.

- Set Up Import Attributes

  Ip2ImportSetupAttrExit lets you map system attributes in the CIU file to VisualInfo attributes, and permits you to convert attribute data specified in the CIU file to a different type.

The following list of user exits refers to the System Administration Program:

- Privilege Set

  The purpose of Ip2ExitChangeApplPriv is to manipulate the VisualInfo privilege bytes.

- Object Sort

  The Ip2ExitCompareStrings compares two null terminated strings. This function is used when you select the ″Sort by name″ option from the ″View″ menu of the system administrator program′s secondary windows.

If you want to use your own code page conversion, you need to use the language support user exit.

- Code Page Convert

  Use IsoCpConvertString to translate a string from one code page to another.

If you plan to do your own error message processing, use the message processing user exit.

- Alert User

  The Ip2UtAlertExits lets you control the error logging and generation of generic alerts by VisualInfo.

The following sample code demonstrates how you can work with user exits.

VisualInfo User Exit Sample

```
//----------------------------------------------------------------------
// Sample of "AlternateSearchUserExit()"
//----------------------------------------------------------------------
int SIMENTRY AlternateSearchUserExit ( HSESSION hSession,
                                       HWND hWnd,
                                       PSZ pszUserID,
                                       USHORT usTypeFilter,
                                       BITS fWipFilter,
                                       USHORT usSuspendFilter,
                                       USHORT usIndexClass,
                                       USHORT usNumCriteria,
                                       PLIBSEARCHCRITERIASTRUCT pCriteria,
                                       PITEMID pItemIdResultFolder)
{
  RCSTRUCT rcs;                         // Return data structure

  // Set result empty
  (*pItemIdResultFolder)[0] = 0;

  // Perform search
  SimLibSearch( hSession,               // VisualInfo session handle
                NULL,                   // Item filter (not supported)
                NULL,                   // Link criteria (n. supp.)
                SIM_SEARCH_DYNAMIC,     // Use dynamic SQL query
                usTypeFilter,           // Type of items to search for
                fWipFilter,             // WIP status of items  "
                usSuspendFilter,        // Suspension status  "
                usIndexClass,           // Index class identifier
                usNumCriteria,          // Elements in pCriteria
                pCriteria,              // Search criteria for each view
                SIM_SEARCH_MAKE_FOLDER, // Return results in a search folder
                NULL,                   // Synchronous processing
                &rcs );                 // Return data structure

  // Unsuccessful operation ?
  if ( SIM_RC_OK != rcs.ulRC )
    return 1;

  // Successful, but no matches
  if ( NULL == rcs.usParam )
    return 0;

  // No pointer to PITEMID field (item id of search results folder)
  if ( NULL == rcs.ulParam1 )
    return 1;

  // Copy result and free memory
  memcpy( pItemIdResultFolder, (PVOID)rcs.ulParam1, sizeof(ITEMID) );
  SimLibFree( hSession, (PVOID)rcs.ulParam1, &rcs );

  // Completed successfully
  return 0;
}
```

5.2.3 VisualInfo High Level API for OS/2

IBM ImagePlus VisualInfo supports the **Service Broker Architecture** by providing the VisualInfo High Level Programming Interface (VHLPI). The VHLPI is used to integrate VisualInfo with FlowMark or your own applications. It contains Broker, Service, and Requester DLLs that provide functions using the VisualInfo Folder Manager, Client Application, and Image Services capabilities, allowing you to manipulate VisualInfo objects from your C or REXX programs.

The VisualInfo High Level Programming Interface allows multiple applications within a client workstation to access VisualInfo, and eliminates the need for these applications to repeatedly log on to VisualInfo. Furthermore, it simplifies the access to VisualInfo, as the most commonly needed VisualInfo functions are provided in simple APIs, thus reducing the time to create customer VisualInfo applications. The VHLPI is part of the IBM ImagePlus VisualInfo for OS/2 product, and is composed of the following components:

Table 5. VHLPI Components

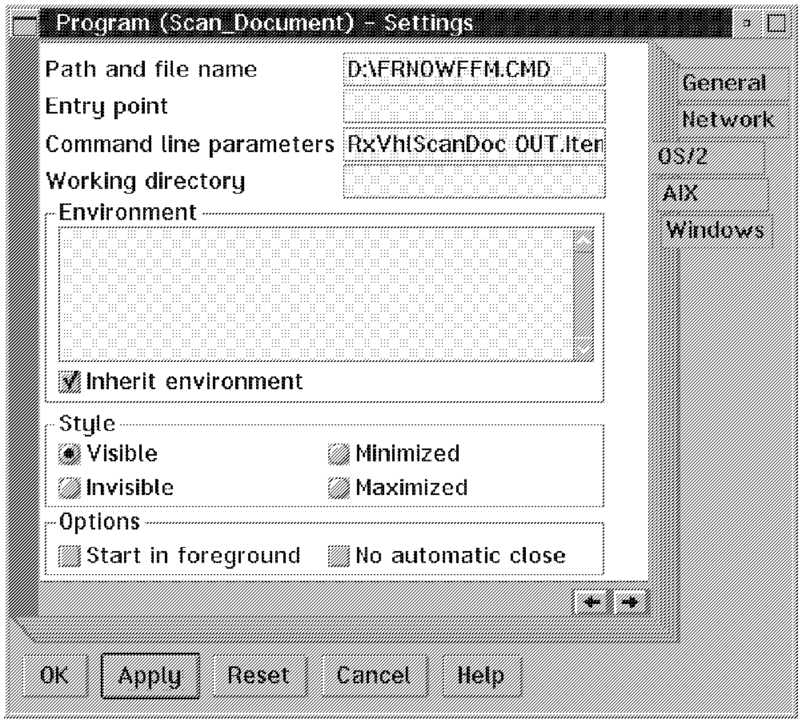
Name of file	Description
FRNOWFFM.CMD	Command file to call VHLPI functions from FlowMark
FRNOWFRX.DLL	Requester module for REXX calls
FRNOWFRC.DLL	Requester module for C calls
FRNOWFBK.DLL	Broker module
FRNOWFSV.DLL	Service module
FRNOWFUL.DLL	Utility module for logging and INI files
FRNOWFSB.EXE	Service Broker Manager replacement

The **FRNOWFFM.CMD**, which is a REXX command file, allows VHLPI REXX functions to be called directly from FlowMark activities without having to create a C or REXX program. This enhanced integration facility allows you to create workflow models with FlowMark that access the VisualInfo functionality, without coding customized programs.

All you need to do to quickly integrate FlowMark and VisualInfo on OS/2 is to install and set up the IBM FlowMark and IBM ImagePlus VisualInfo products and do the following:

1. Design and model the workflow process with FlowMark.
2. For FlowMark activities where you want to invoke VisualInfo functions such as scanning a document and displaying the image, register the FRNOWFFM.CMD as a program object and define the necessary parameters.

   To register a program object in FlowMark Buildtime Client, you create a new program object from the Programs folder, and specify the program attributes in the Settings notebook as shown in Figure 19 on page 56.



Path and file name: D:\FRNOWFFM.CMD
Entry point:
Command line parameters: RxVhlScanDoc OUT.Iten
Working directory:

Environment
[✓] Inherit environment

Style
● Visible
Invisible
Minimized
Maximized

Options
Start in foreground
No automatic close

Figure 19. FlowMark Program Registration

The following FDL extract illustrates what these settings on the OS/2 page of the Settings notebook may look like when you export them:

FDL Extract of Program Registration

```
STRUCTURE 'Default Data Structure'
END 'Default Data Structure'

STRUCTURE 'Doc_Info'
    'ItemId':   STRING;
END 'Doc_Info'


PROGRAM 'Scan_Document' ( 'Default Data Structure', 'Doc_Info' )
  DESCRIPTION 'Scan a document into ''CreditRequest'' index class with
               the attribute ''Origin'' set to ''Scan''.
               Return the item id to output container.'

  OS2 PATH_AND_FILENAME 'FRNOWFFM.CMD'
  OS2 PARAMETER 'RxVhlScanDoc OUT.ItemId CreditRequest Origin Scan'
END 'Scan_Document'

PROGRAM 'Display_Document' ( 'Doc_Info', 'Default Data Structure' )
  DESCRIPTION 'Display a document image specified by item id
               at position 0,0 and size 500,600.'

  OS2 PATH_AND_FILENAME 'FRNOWFFM.CMD'
  OS2 PARAMETER 'RxVhlDisplayDocView %ItemId% N 0 0 500 600'
END 'Display_Document'
```

3. Translate and run the process.

The list of VisualInfo functions you can invoke from the **FRNOWFFM.CMD** are listed in Table 6 on page 58.

The steps previously described may be enough for quick results, however you probably want to create a customized application with a full user interface. This may be done with tools such as VisualREXX or VisualAge. The following gives a brief outline on how to create a fully-customized workflow and document management application on OS/2, using FlowMark for OS/2, VisualInfo for OS/2, and the VisualInfo High Level Programming Interface:

1. Define the workflow process that you want to carry out, where VisualInfo objects are manipulated from activities in the FlowMark process.
2. Model your workflow process with FlowMark Buildtime Client

   A workflow model is a complete representation of a process, containing a process diagram and the settings that define the logic behind the components of the diagram.

   a. Draw a diagram of your process showing each activity and block and the control and data connectors. Specify the settings of the process such as name, duration, category the process belongs to, and the process description.

      This provides you with an overview of the complete process to work with.

   b. Define the data structures to be used by the activities in your process.

   c. Define the people, roles, and organization as well as levels.

   d. Register the programs that are to be used for the activities in the process. For each program, define the input and output data structures, the path and filename, and the command-line parameters. The following program types are supported for OS/2: EXE, CMD, DLL, BAT, COM, or PIF.

   e. Define, in detail, the logic behind your process diagram.

      - For each activity, specify the start and exit conditions, the people, data structures, and programs required to perform the activity.
      - For each control connector, optionally specify a transition condition that must evaluate to True for control to flow that way.
      - For each data connector, specify how the data in the output container of one activity is mapped to the input container of another.
      - Check the definitions in your process diagram using the Check facility. This checks the logical expressions of all the conditions you have defined (start, end, exit, transition), the settings for the activities, the data mappings, and the consistency between the program registrations and the program activities.

   f. Test and verify the workflow model using the FlowMark animation facility.

   g. Translate the workflow model into a runtime process template.

3. Write your application programs that you have registered in FlowMark (in step 2.d.). The VHLPI requester functions are available in C and REXX, thus for all programs where you need to manipulate VisualInfo objects, you need to first decide which interface to use. Depending on this, you need to write C programs or REXX command files.

Your application programs may consist of the following parts:

- Definition of user interface, if user interaction is required.
- Calls to the standard FlowMark API functions to read and write the input and output container variables of the current program activity.
- Calls to one or more VHLPI functions, where the FlowMark container variables are used for input or output. Table 6 lists the VHLPI requester functions that are provided.

4. If you have many programs to write, it may be a good idea to have dummy programs that have the same file name as those you want to write. This allows you to test the FlowMark process in Runtime. These dummy programs are just empty or very simple EXEs or CMDs. As you complete your programs one after the other, you just need to replace these dummy programs with the real programs and test your runtime process.

5. Once you have finished implementing and testing your programs, you are ready to test run the entire application. Here you need to verify that the process runs as you expect and that all of the necessary data is passed on correctly between the activities.

Table 6 (Page 1 of 2). VHLPI Requester Functions

C Function	**Purpose**
VhlAddFolderItem	Adds an Item to a folder.
VhlAdminItemNoteLog	Reads, appends, replaces or deletes Note Logs.
VhlChangeItemIndex	Changes the index class of an Item to another index class.
VhlCheckInItem	Checks-in the Item.
VhlCheckOutItem	Checks-out the Item for exclusive update access.
VhlCloseDocViews	Closes all document image view windows.
VhlCopyDoc	Creates a document and copies the contents of an existing document into it.
VhlCreateFolder	Creates a new folder.
VhlCreateFolderAddItem	Creates a folder and adds the specified Item into it.
VhlDeleteItem	Deletes the Item from IBM ImagePlus VisualInfo.
VhlDisplayDocView	Displays a document image using Image Services.
VhlDisplayVIItem	Displays a document, folder, or workbasket in a Library Object Window
VhlExportDocObj	Creates an external file from a document base object.
VhlGetVIUserID	Returns the UserID logged onto IBM ImagePlus VisualInfo.
VhlImportDocObj	Creates a document base object from an external file image.
VhlListContClasses	Lists all Content Classes.
VhlListFolderItems	Lists all folder Items (of specified index classes).
VhlListFolderItemsAttr	Lists all folder Items and their attribute values.
VhlListFolderXrefItem	Lists all folders which contain the specified Item.
VhlListIndexClassAttr	Lists all properties and attributes of a specified Index Class.

Table 6 (Page 2 of 2). VHLPI Requester Functions

C Function	Purpose
VhlListIndexClasses	Lists all Index Class names.
VhlListItemCC	Lists the Content Class of an Item′s base object.
VhlListItemInfo	Lists an Item′s type index class name, index attributes and values.
VhlListWBItems	Lists all Items in a specified workbasket.
VhlListWorkBaskets	Lists all workbasket names and Item IDs.
VhlRemoveFolderItem	Deletes the Item only from the specified folder.
VhlScanDoc	Invokes the IBM ImagePlus VisualInfo Scan facility.
VhlSearchAdv	Returns all Items which match the advanced Search Criteria.
VhlSearchItem	Returns all Items which match the index class and index attribute specification.

For a complete description of the VHLPI functions and syntax, refer to the *VisualInfo High Level Programming Guide and Reference*. For information on defining your workflow with FlowMark, refer to the *FlowMark V2.1 Modeling Workflow, SH19-8241-00*.

5.2.4 VisualInfo OLE Automation API for Windows

With the VisualInfo OLE Automation API for Windows, you can manage a single logon to the VisualInfo system for a Client Workstation. These APIs are available with the VisualInfo Client Application and provide equivalent function to the VisualInfo OS/2 High Level API.

The VisualInfo Client Application OLE Automation objects are manipulated by 4GL tools such as Visual Basic, Visual C++, and PowerBuilder.

The following Visual Basic code extracts show how you can use OLE automation to control VisualInfo Client Application from your application.

First you have to connect to the VisualInfo Client Application OLE server, and this is done in the Form_Load sub. VisualViewApp is the VisualInfo Client Application OLE application object, and it is global.

VisualViewApp Declaration

```
Global VisualViewApp As Object
```

Form_Load

```
Private Sub Form_Load()

    ′ Get connection to VisualInfo Client Application OLE server
    Set VisualViewApp = CreateObject(″Vic.Application″)

End Sub
```

The following sub is designed to perform a VisualInfo Client Application logon. The Visual Basic form contains buttons called ″Logon″ and ″Cancel″, and entry fields called ″UserID″, ″Password″ and ″Server″. VILogon is a global variable (flag) that shows whether the logon was successful or not.

Logon_Click

```
Private Sub Logon_Click()

    ′ Data declaration
    Dim Rc As Integer

    ′ Set logon information from the entryfields
    VisualViewApp.User = UserID.Text
    VisualViewApp.Password = Password.Text
    VisualViewApp.Server = Server.Text

    ′ Perform logon to VisualInfo
    Rc = VisualViewApp.Logon
    If Rc = 1 Then
        MsgBox ″An error occurred while logging on to VisualInfo.″
    Else
        VILogon = 1
        Unload Logon
    End If

End Sub
```

On another form, you have a button called ″GetNext″. If you press it, the next item (document or folder) from the ″To be indexed″ workbasket is displayed.

GetNext_Click

```
Private Sub GetNext_Click()

    ′ Data declarations
    Dim Workbasket As Object
    Dim Item As Object

    ′ Get the item from the workbasket
    Set Workbasket = VisualViewApp.GetWorkbasket(″To be indexed″)

    ′ Get next item from workbasket
    Set Item = Workbasket.NextWorkbasketItem

    ′ Find out if the item is a folder or a document
    If (Item.Type = 1) Then
        ′ Document! Display it.
        VisualViewApp.Image.OpenDocument Item
    Else
        ′ Must be a folder. Display it.
        VisualViewApp.Documents.OpenTOC Item
    End If

End Sub
```

You can create a new customer folder by pressing the button called ″CreateCustFold″. First, a new folder with the NOINDEX class is created, and then it is re-indexed with the data from the form′s entryfields. Finally, it is displayed.

CreateCustFold_Click

```
Private Sub CreateCustFold_Click()

    ' Data declaration
    Dim Docs As Object

    'Create a new folder with the NOINDEX class
    Set Item = VisualViewApp.CreateFolder("CREDIT")

    'reindex the folder with the data from the entryfields
    Item.Class = "Form C"
    Item.KeyFields("Name")   = Name.Text
    Item.KeyFields("Number") = Number.Text
    Item.KeyFields("Date")   = Date.Text
    Item.UpdateIndex

    'Display it
    Set Docs = VisualViewApp.Documents
    Docs.OpenTOC Item

End Sub
```

Chapter 6. Integration Techniques

The way you can integrate applications into your system depends on the interfaces they offer. Desktop applications generally offer DDE, user exits or OLE. Host applications can be accessed through remote procedure calls or HLLAPI. Some products have a product specific interface for integration. To use EXTRA! functions, you may use the EXTRA! Basic. These specific interfaces cause problems when you decide to substitute the product with a similar one. Therefore, you should try to use standard interfaces whenever possible.

This chapter discusses these integration concepts in detail.

6.1 DDE

Dynamic Data Exchange allows applications to communicate with each other. A DDE client application queries data from a DDE server application according to a defined protocol. First, the client has to specify which DDE server application (AmiPro, Excel) it wants to communicate with. Second, the topic of the communication is defined. The operating system routes the client′s request for communication to all possible DDE servers. It is up to them to respond to the client. They have to decide if they are the right server (matching of server name), and if they are able to talk about the defined topic.

Note: The server name is not case-sensitive.

Usually, DDE server applications support a topic called ″System″. This topic offers information about other topics the DDE server is able to talk about, and often some application version specific data. Topics can also be file names, for instance, TABLE.XLW, if you use Excel as your DDE server.

The link between a DDE client and DDE server is established if the client decides to talk to a server that answered the request for communication. Now the client can query information about items related to the defined topic. An item can be the position of a specified cell in Excel or the name of an entry field in your DDE server application. There are three kinds of links, distinguished by how the server updates the client when data changes.

- **Automatic Link**

  The DDE server is forced to supply data to the DDE client whenever data defined by the item changes. The DDE client should establish an automatic link, if the transferred data is small and each update of that data is needed.

- **Manual Link**

  The DDE client has to request each update of data defined by an item. A manual link can be established to query data only one time or in a client-defined interval.

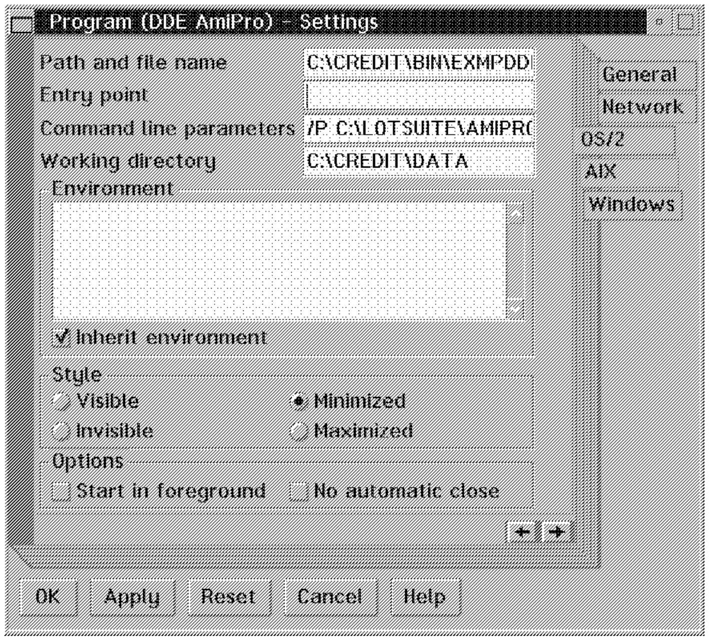
- **Notify Link**

  If data related to an item changes, the DDE server has to notify the DDE client of that change. It is up to the DDE client to request the changed data. The DDE client should establish a ″notify″ link if it needs to know when data changes (but is too large to be transferred each time).

The flow of information is usually from the DDE server to the DDE client. However, sometimes the client needs to transfer data to the server. To solve this problem, DDE allows you to ″poke″ data to your server.

The benefit of DDE is that you can use it on various platforms such as OS/2, Windows 3.x, 95, or NT. It is possible to implement a DDE server or client application with different programming languages such as C/C++ or Visual Basic. There are also some tools that help you to use DDE communications without programming, such as the FlowMark DDE Building Block (EXMPDDEI.EXE). You may use this building block in activities of your workflow to load documents in a word processor (such as AmiPro) and execute a macro.

The following screen shot shows how the DDE building block can be registered as a FlowMark program:


Program (DDE AmiPro) - Settings

Path and file name: C:\CREDIT\BIN\EXMPDD
Entry point:
Command line parameters: /P C:\LOTSUITE\AMIPR(
Working directory: C:\CREDIT\DATA

Environment
[✓] Inherit environment

Style
( ) Visible [● Minimized]
( ) Invisible ( ) Maximized

Options
[ ] Start in foreground [ ] No automatic close

Figure 20. FlowMark Program Registration

The following FDL extract illustrates the exported program registration:

FDL Extract of Program Registration

```
PROGRAM 'Write_Accept_Doc'      ( 'Credit_Info', 'Default Data Structure' )

  DESCRIPTION 'DDE AmiPro'

  OS2 PATH_AND_FILENAME 'C:\CREDIT\BIN\EXMPDDEI.EXE'
  OS2 PARAMETER '/P C:\LOTSUITE\AMIPRO\AMIPRO.EXE C:\TEMPLATE\PROTOCOL.SAM'
  OS2 WORKING_DIRECTORY 'C:\CREDIT\DATA'
  OS2 STYLE MINIMIZED

END 'Write_Accept_Doc'
```

For more information about FDL, refer to *FlowMark V2.1 Modeling Workflow, SH19-8241-00*.

If you want to develop a DDE client with Visual Basic, you will find the following sample useful.

Establish a manual link between a textbox and Excel, when the user first clicks ″ButtonStart″. All of the following clicks update the data. It is assumed that ddesrv is a string containing ″EXCEL″, ddepath ″C:\MSOFFICE\EXCEL″, and ddefile ″CREDIT.XLW″. The row and column of the worksheet are specified through row and col.

Visual Basic Code Extract

```
Private Sub ButtonStart_Click ()

  If MyText.LinkMode = vbLinkNone Then

    MyText.LinkTopic = ddesrv & ″|″ & ddepath & ″\[″ & ddefile & ″]Sheet1″
    MyText.LinkItem  = ″R″ & row & ″C″ & col
    MyText.LinkMode  = vbLinkManual

  else
    MyText.LinkRequest

End Sub
```

For more information about how you can use DDE communication with Visual Basic, refer to *Microsoft′s Visual Basic Programmer′s Guide ″Dynamic Data Exchange (DDE)″*.

6.2 OLE Automation

OLE Automation servers are a mechanism for providing services in a Windows environment. Once a service has been implemented as an OLE server, it can be used by your application. You can carry out these services using Visual Basic or a Windows C++ compiler such as Microsoft Visual C++ or Borland C++.

To speak in client/server terms, your application is the client that requests services from the OLE server. Both client and server may be contained on a single computer, or they may run on different computers that are connected through a network. In the first case the OLE server is called a local server, and in the second case, it is called a remote server. Local OLE servers may be implemented as an executable file (EXE) or as a dynamic link library (DLL). An executable file runs in a separate address space, therefore OLE servers such as Excel are called *out-of-process servers*. The *in-process servers* are dynamic link libraries that run in the same process as the client. This client and the called dynamic link library share one address space. Therefore calls to routines of the OLE server can use the client′s stack and the server routines can access the client′s data directly by using the same address. No pointer conversion is necessary.

An external process is not able to use pointers to an OLE client′s address space. This problem is solved by copying the data in the out-of-process server′s address space and replacing the original pointers with new pointers to the copied data. The new pointers are needed by the server to modify the data. When the server routine ends, the data of the call by reference parameter is copied back into the client′s address space. This procedure is called marshaling and it is completely transparent to the client. Note that marshaling is limited in several ways. First it is slower than passing parameters in a single address space, and second you cannot pass pointers to another address space, you have to use a copy of this data. The benefit of an out-of-process server is that its services can be used by several clients, while each client needs its own copy of an in-process server when calling its routines.

The following figure shows an OLE client and an out-of-process OLE server:

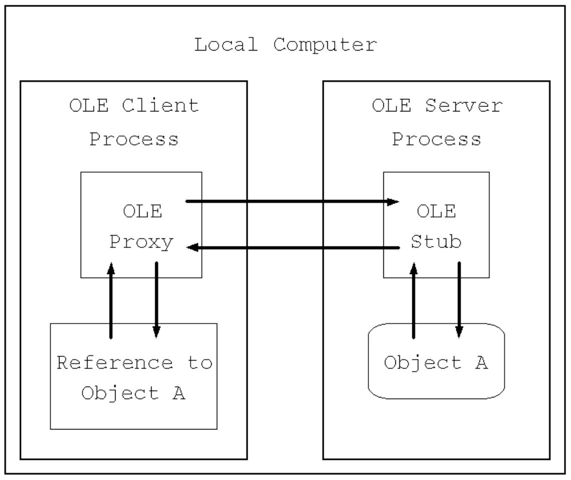


Figure 21. Out-of-Process OLE Server

When using remote OLE automation, a program is needed to establish and control the communication between the client and the server process. This program is called *Automation Manager*. The Automation Manager also takes part in the process of marshaling and unmarshaling. To communicate across the network, remote procedure calls (RPC) are needed. Therefore, the proxy and stub used by local OLE automation are replaced with modules that are able to use RPC.

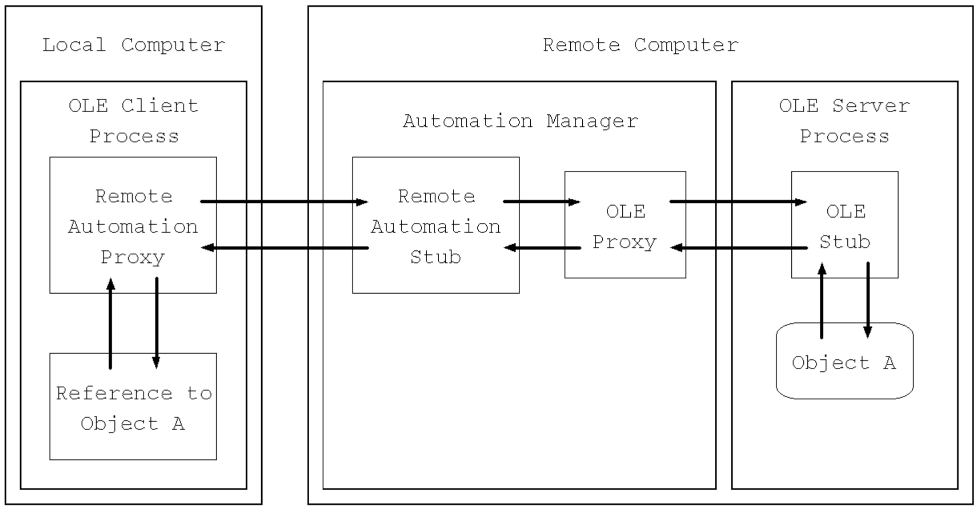


Figure 22. Remote OLE Server

Remote OLE automation requires that the Automation Manager be running on the server before any client calls are started. Also, you have to use Windows 95 or NT. Any earlier version of Windows does not support remote OLE automation. Remote servers should not display errors by using message boxes or a similar output on the remote computer; they should always return errors to the client. Also remote servers must not be put in a state in which they are waiting for user input. Waiting for user input may hang this remote server until an operator is able to enter data on the remote computer.

You can find more information about OLE and implementing your own OLE server in *Inside OLE 2 and OLE 2 Programmer′s Reference* published by Microsoft Press.

To integrate existing applications in your workflow, the programming of OLE clients is more important than the programming of the servers. The following sample demonstrates how to access and use OLE objects within your application.

Sample code demonstrating the use of the VIP VisualView (or VisualInfo Client Application) OLE server to integrate a Visual Basic application with the document management features of VisualInfo on the Windows platform has been included in Chapter 4, "VisualInfo - the Document Manager" on page 35.

The following section describes the OLE automation supported by EXTRA! MAINFRAME for Windows application. The EXTRA! Personal Client supports OLE automation for a smooth integration of host applications in your solution. As is usual in OLE, you can control the OLE objects by properties and methods. In the hierarchical OLE object model of EXTRA!, the **session** object is the top level object. It provides access to all EXTRA! OLE objects.

The **sessions collection** manages the open session objects. They enable you to access host data, and work with EXTRA! functions. The quick pad, toolbar, and display are part of the session. They provide access to the session′s quick pad, toolbar, and the host display's presentation space. Quick pads and toolbars are managed by quick pad collection and toolbar collection.

The area object provides access to a defined area of the display.

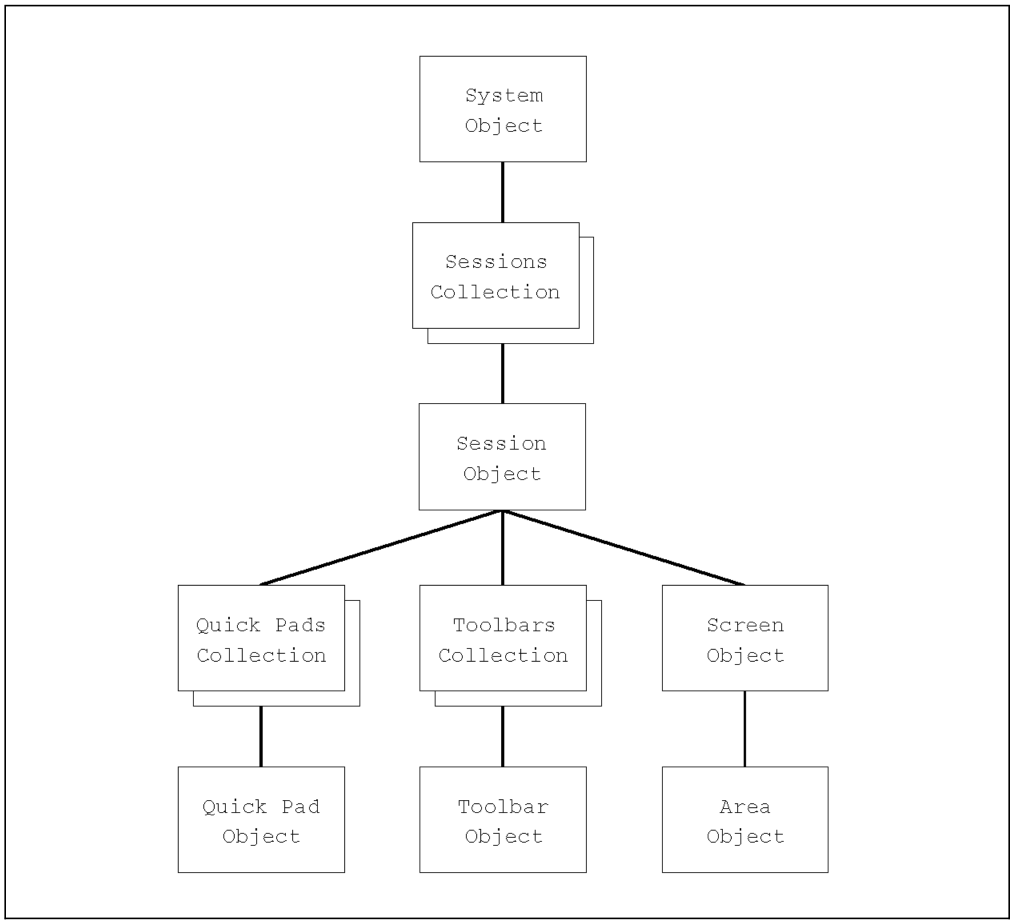


Figure 23. Hierarchy of the EXTRA! OLE Objects

To work with the EXTRA! OLE objects, you must use a macro or programming language that supports OLE automation such as EXTRA! Basic or Visual Basic.

The following Visual Basic extract illustrates a sample EXTRA! OLE integration, where data is exchanged between the application and the host display.

EXTRA! OLE Code Extract

```
Global gvSess As Object   ′ Extra Session object
Global gvScreen As Object ′ Extra Screen Object
Global gvArea As Object   ′ Extra Area Object

Public Function mySearchExtraString(pStr As String) As Boolean

  ′ Search string
  Set gvArea = gvScreen.Search(pStr)

  ′ Read position of string; ″-1″ -> not found
  If gvArea.Left = -1 Then
    ′ String not found
    mySearchExtraString = False
  Else
    ′ String found
    mySearchExtraString = True
  End If

End Function

Public Function IsListEmpty() As Boolean

  ′ Init return value
  IsListEmpty = False

  ′ Correct screen displayed?
  If Not mySearchExtraString(″CREDIT A02″) Then

    ′ Wrong screen
    MsgBox(″Wrong screen″, vbOk)
    Exit Function

  End If

  ′ Choose option 4 and press enter
  gvScreen.SendKeys (″4<Enter>″)
  gvScreen.WaitForString(″entries″)

  ′ Any entries found?
  If mySearchExtraString(″no entries″) Then IsListEmpty = True

End Function
```

6.3 HLLAPI

The High-Level Language Application Programming Interface (HLLAPI) is an application programming interface that allows PC programs to interact with a host using 3270 or 5250 emulation. IBM′s implementation of this programming interface in the Communications Manager application is called a Emulator High-Level Language Application Programming Interface (EHLLAPI).

A program using HLLAPI acts similar to a programmed operator, which means that the HLLAPI program carries out and monitors activities that are usually done by a human operator. As you can see, with HLLAPI programs, you can automate data exchange, and integrate existing host applications.

HLLAPI programs can be coded in several programming languages listed here along with an IBM example:

- BASIC (IBM BASIC Compiler/2)
- COBOL (IBM COBOL/2)
- C (IBM Visual Age C/C++)
- REXX (DOS Restructured Extended Executor)
- Macro Assembler (IBM Macro Assembler/2)

The simplified figure illustrates how a HLLAPI program interacts with a host application:

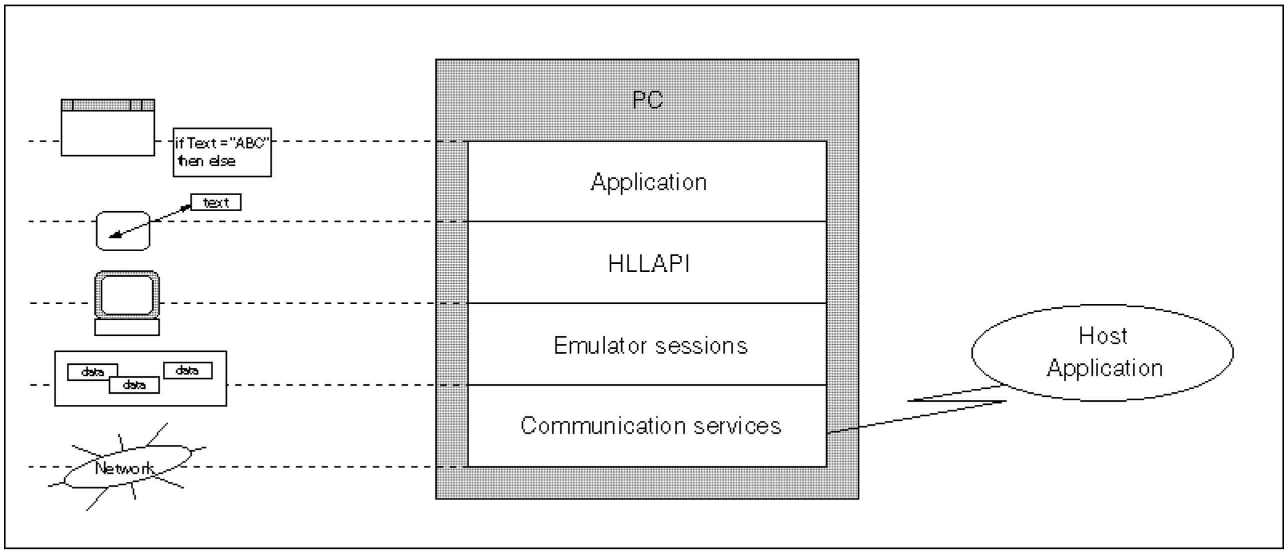


Figure 24. Simplified HLLAPI PC Host Communication

The communication services receive data from the network, and send data to the network. This data is passed to the emulator software, which is able to put it in a readable format on its presentation space. The HLLAPI works with these presentation spaces, therefore a HLLAPI application can exchange strings with a host application.

HLLAPI programs generally work in the following way:

1. **Connect**

   In your HLLAPI program, you first have to connect to the presentation space of the emulator session, where your host program is shown. A presentation space is a region in computer memory that can be displayed on the display of 3270 or 5250 emulation.

   Note: The communications services and the emulator sessions have to be started before you can run your HLLAPI program. For example, on an OS/2 platform with Communications Manager/2, you must perform a *CMSTART*, and start at least one of your emulator sessions before you can run your EHLLAPI program.

2. **Read and Write Data**

   There are several ways to exchange data with the emulator′s presentation space. You can simply read and write characters or strings, or you can search a defined string in the presentation space, and then read or write characters or strings. It is also possible to wait for the occurrence of a defined string, and then read or write anything. If you are working with a 3270 emulation, you can access structured fields. Therefore, you can design your program in a more comfortable way.

3. **Disconnect**

   At the end of your HLLAPI program, you have to disconnect from the emulator′s presentation space. Once you are disconnected, you can stop the emulator session and the communication services as you normally do.

HLLAPI can also be used to send files to the host, or receive files from the host. It is possible to specify what kind of file you want to transfer: binary or text. For file transfer, you have to use ″send.exe″ and ″receive.exe″ which are located in your communication services directory

The HLLAPI is very easy to use, but it is not a *fast* interface. When high performance is not your priority, you can use it to navigate through host applications and exchange data with them.

The following REXX HLLAPI sample illustrates how you can use HLLAPI calls to query keystrokes from your emulator session, and how you can write characters and strings to your emulator session.

REXX HLAPPI -- Keystrokes in an Emulator Session

```
/*------------------------------------------------------------------*/
/*  Keystroke Recorder  -  Record and play back keystrokes          */
/*                                                                  */
/*  Input: Emulator session (A, B, C, etc.)                   */
/*------------------------------------------------------------------*/

/*------------------------------------------------------------------*/
/* Read only first argument ″session″                               */
/*------------------------------------------------------------------*/
parse arg session .

/*------------------------------------------------------------------*/
/* If no session name entered, use session ″A″                      */
/*------------------------------------------------------------------*/
if session = ″″ then session = ″A″

/*------------------------------------------------------------------*/
/* Define control keys for ″toggle″, ″play back″ and ″quit″         */
/*------------------------------------------------------------------*/
toggle_key = ″@rt″       /* CTRL-T  toggle recording mode           */
play_key   = ″@rp″       /* CTRL-P  Play back keystrokes            */
quit_key   = ″@rq″       /* CTRL-Q  Quit the program                */

/*------------------------------------------------------------------*/
/* Display control keys                                             */
/*------------------------------------------------------------------*/
say ″Toggle recoding mode ......: Ctrl-T″
say ″Play back .................: Ctrl-P″
say ″Quit ......................: Ctrl-Q″

/*------------------------------------------------------------------*/
/* Load HLLAPI functions, if not loaded before                      */
```

```
/*------------------------------------------------------------------*/
if rxfuncquery("hllapi") then
   call rxfuncadd "hllapi","saahlapi","hllapisrv"

/*------------------------------------------------------------------*/
/* Show session on OS/2's presentation space and connect to it      */
/*------------------------------------------------------------------*/
rc = hllapi("Set_session_parms","CONPHYS")
if rc<>0 then do
   signal quit
end

rc = hllapi("Connect",session)
if rc<>0 then do
   signal quit
end

/*------------------------------------------------------------------*/
/* Filter all keystrokes from the specified session                 */
/*------------------------------------------------------------------*/
rc = hllapi("Start_keystroke_intercept",session,"L")
if rc<>0 then do
   signal quit
end

/*------------------------------------------------------------------*/
/* Initialize flags for "quit" and "record mode"                    */
/*------------------------------------------------------------------*/
quitprog = 0
record = 0

do while quitprog<>0
   /*---------------------------------------------------------------*/
   /* Wait for a keystroke                                          */
   /*---------------------------------------------------------------*/
   key = hllapi("Get_key",session)
   select

      /* Quit program                                               */
      when (key = quit_key) then do
          quitprog = 1
      end

      /* Stop recording                                             */
      when (key = toggle_key) & record then do
         say "STOP"
         record = 0
      end

      /* Clear "keystroke memory" and start recording               */
      when (key = toggle_key) & ¬record then do
         say "START"
         record = 1; string = ""
      end

      /* Play back string                                           */
      when (key = play_key) then
         say "PLAY BACK"
         rc = hllapi("Sendkey",string)
      /* Save keystroke in string and send it to emulator′s ps      */
      when record then do
          string = string || key
          rc = hllapi(″Sendkey″,key)
      end

      /* Send keystroke to emulator′s ps without saving it          */
      when record<>0 then
          rc = hllapi(″Sendkey″,key)

      /* Do nothing                                                 */
      otherwise nop
   end   /* select statement */
end   /* do statement */

/*-----------------------------------------------------------------*/
/* Stop filtering all keystrokes from the specified session        */
/*-----------------------------------------------------------------*/
rc = hllapi(″Stop_keystroke_intercept″,session)

/*-----------------------------------------------------------------*/
/* Disconnect and exit                                             */
/*-----------------------------------------------------------------*/
quit:
  call hllapi (″Disconnect″)

exit
```

End of REXX HLAPPI -- Keystrokes in an Emulator Session

The sample above allows you to save all of the keystrokes that you type in your emulator session. You can start and stop the recording of keystrokes with Ctrl-T. By pressing Ctrl-P, you can play back the saved keystrokes. To quit the program, press Ctrl-Q.

For more REXX samples (cmmacro.cmd: Keyboard macro facility, similar to the preceding sample; qtime.cmd: Set PC time to host time) and further information about EHLLAPI programming, refer to *IBM Communications Manager/2 V1.11 EHLLAPI Programming Reference*, SC31-6163.

6.4 Accessing C and C++ APIs from Visual Basic

If you are integrating applications with Visual Basic that do not provide Visual Basic APIs, you need to find a way to access these APIs from Visual Basic.

To use C APIs from VisualBasic, you have to know in which dynamic link library (DLL) they are located. For C++ calls, it may be necessary to write a C++ program that encapsulates the C++ interface and exports the functionality you need in a DLL as normal C functions.

These DLLs have entry points (the exported C functions of the DLL) that you can register in Visual Basic. Once an entry point is registered, you can call it just as you call a Visual Basic function.

Note: Visual Basic is not able to verify if you pass the right arguments in the correct order to the functions in your DLL.

If you pass anything wrong, your application may crash.

To access your C or C++ calls from Visual Basic, follow these steps:

1. **C or C + + Coding**

   Code all functions you want to access from Visual Basic in your C or C++ files. Generate a LIB and a DLL file. The LIB file contains the entry point definitions. You need them to translate programs that use your DLL. The DLL file contains your translated code.

   All of the functions in your DLL that you want to access from other DLLs or EXEs, have to be declared as exportable functions. With Borland C++ 4.5, you have to enter:

   Exportable Function

   ```
   #define APIENTRY  far pascal _export

   char APIENTRY func1( char chMyChar );
   void APIENTRY func2( long lMyLong );
   ```

   Note: It is possible to create OLE 2.0 objects or OCX controls (or even ActiveX controls) in the Windows world and OpenDoc objects for OS/2, Windows and AIX platforms which can be compatible with OLE on Windows platforms using IBM VisualAge C++.

   Instead of defining APIENTRY yourself, you can include EXMWJAPC.H, located in your EXMWIN\API directory. You can use a makefile such as MYDLL.MAK to compile and link your programs:

MYDLL.MAK

```
#------------------------------------------------------------
# MAKEFILE .....: "MYDLL.LIB" and "MYDLL.DLL"
# COMPILER .....: Borland C++ 4.5
#------------------------------------------------------------

#------------------------------------------------------------
# Tools
#------------------------------------------------------------
IMPLIB  = Implib
BCC     = Bcc
TLINK   = TLink

#------------------------------------------------------------
# Debug flags
#------------------------------------------------------------
CDEBUG   = -v
LDEBUG   = /v

#------------------------------------------------------------
# Compiler and linker flags
#------------------------------------------------------------
CFLAGS  = -ml -c -3 -tWDE $(CDEBUG)
LFLAGS  = /d /Twd $(LDEBUG)

#------------------------------------------------------------
# Path information
#------------------------------------------------------------
PATHBC  = C:\APPL\WIN\BC45

#------------------------------------------------------------
# File information
#------------------------------------------------------------
FILE    = MyDll

#------------------------------------------------------------
# Target LIB
#------------------------------------------------------------
$(FILE).lib : $(FILE).dll
  $(IMPLIB) $@ $(FILE).dll

#------------------------------------------------------------
# Target DLL
#------------------------------------------------------------
$(FILE).dll : $(FILE).obj
  $(TLINK) -L$(PATHBC)\LIB $(LFLAGS) \
 c0dl.obj+$(FILE).obj, $(FILE).dll,, @$(FILE).cfg, $(FILE).def

#------------------------------------------------------------
# Target OBJ
#------------------------------------------------------------
$(FILE).obj : $(FILE).cpp $(FILE).mak
  $(BCC) $(CFLAGS) -I$(PATHBC)\INCLUDE \
 -D_RTLDLL;_BIDSDLL; $(FILE).cpp
```

2. **Entry point registration**

   You can register the entry points as a Visual Basic function in the declaration section of a form, standard, or class module. Use the **Declare** statement such as:

   Declare Sample

   ```
   Declare Function func1 Lib ″MyDll″ (ByVal chMyChar As Byte) As Byte
   Declare Sub func2 Lib ″MyDll″ (ByVal lMyLong As Long)
   ```

   As you can see in the sample, you have to declare functions that have a return value as **Function**. Functions without a return value (void) have to be declared as **Sub**. If you declare your DLL functions in the standard module, they can be called by code anywhere in your Visual Basic program because they are public. To declare a DLL function in the declaration section of a form or class module, you have to include the **Private** keyword in the declaration.

   The library name specified after the keyword **Lib** is not case sensitive for 16-bit versions of Visual Basic, but is case sensitive for 32-bit ones. The search order is:

   a. Directory of executable file

   b. Current directory

   c. Windows 32-bit and 16-bit system directory

   d. Windows directory

   e. Directories stored in the PATH environment variable

   If you want to use different function names in Visual Basic than you used in C or C++, you have to use the **Alias** keyword. To work with ″f2″ instead of ″func2″, you should enter:

   Alias Sample

   ```
   Declare Sub f2 Lib ″MyDll″ Alias ″func2″ (ByVal lMyLong As Long)
   ```

Refer to appendix Appendix A, “FlowMark C++ API Sample” on page 95 to find a sample including the full source code, a definition file and make file. That sample shows how it is possible to use the FlowMark C++ APIs from Visual Basic.

For more information on how to access DLL functions in Visual Basic, refer to Microsoft′s *Visual Basic Programmer′s Guide*. You also find information about how to pass your specific data (user defined types, flexible types) to your functions. Also, you get information about calling DLL functions in 16 and 32-bit environment.

6.5 Accessing C and C++ APIs from REXX

Using REXX, you can develop your programs very rapidly. However, some applications you want to integrate may not offer REXX APIs, but C or C++ APIs instead. This section describes how you make these APIs available for your REXX programs.

First let′s have a look at typical REXX and C function calls:

Typical REXX Calls

```
call Func1 'some Text', 2
myVal = Func1( myText )
```

Typical C Calls

```
myFunc1( "some Text", 2 );
rc = myFunc1( pszText, sShort );
```

As you can easily see, REXX and C calls differ in several ways. Therefore, you cannot just register any C function in REXX and call it.

It is possible to extend the REXX language with new functions, or to extend an application with REXX. This is done by creating handlers for subcommands, external functions, and system exits. A subcommand is a command that is issued from a REXX program. It runs as an application macro. External functions extend the native set of REXX functions. With system exits, you can customize the REXX interpreter's behavior by replacing the default routines for REXX system requests. Also, applications can manipulate the variables in REXX programs by using the variable pool interface and execute REXX routines directly from memory by using the macrospace interface.

Subcommands, external functions and system exits are coded, compiled, and packed similarly. The following sections focus on the external functions. External functions can be categorized in two parts:

- Routines written in REXX:

  External functions written in REXX need not be registered with REXX; they are found by a disk search for a REXX procedure file that matches the function name.

- Routines written in other languages:

  External functions written in languages other than REXX must be registered with the REXX interpreter.

To register external functions, you can use the RexxRegisterFunctionExe or RexxRegisterFunctionDll calls:

- RexxRegisterFunctionExe:

  If the function resides within application code (EXE file), use RexxRegisterFunctionExe to register the function.

- RexxRegisterFunctionDll:

  If the function resides within dynamic link library (DLL file), use RexxRegisterFunctionDll to register the function.

The following code shows how you can register your functions with the REXX interpreter. MyLoadFuncs must be called from the REXX program. It registers all of your functions.

Register Functions

```
ULONG MyLoadFuncs( CHAR *pszName,     /* Name of the function      */
                   ULONG ulNumArgs,   /* Number of args passed      */
                   RXSTRING args[],   /* Array of arguments         */
                   CHAR *queuename,   /* Current queue name         */
                   RXSTRING *retstr ) /* Return value               */
{
  INT    iEntries;                     /* Num of entries             */
  INT    iCount;                       /* Counter                    */

  retstr->strlength = 0;               /* Set return value           */

  if (ulNumArgs > 0)                   /* Check arguments            */
    return INVALID_ROUTINE;
                                       /* Calculate number of funcs  */
  iEntries = sizeof( aRxFncTable ) / sizeof( PSZ );

  for (iCount = 0; iCount < iEntries; iCount++)
  {
     RexxRegisterFunctionDll( aRxFncTable[iCount],  /* Function name */
                              pszDllName,           /* DLL name      */
                              aRxFncTable[iCount]); /* DLL func. name */
  }

  return VALID_ROUTINE;                /* Successful                 */
}
```

The variables pszDllName (name of the DLL file containing the external functions) and aRxFncTable (array of the names of the external functions) may to be set to the following values:

Global Variables

```
  static PSZ pszDllName = "MYDLL";                 /* DLL name      */
  static PSZ aRxFncTable[] = { "MyFunc",           /* Array of funcs */
                               "MyLoadFuncs",
                               "MyDropFuncs" };
```

You should use the array aRxFncTable for deregistering the functions at the end of the program also. You can name the function for deregistering MyDropFuncs. It works the same as MyLoadFuncs, but uses RexxDeregisterFunction instead of RexxRegisterFunctionDll.

RexxDeregisterFunction Syntax

```
  RexxDeregisterFunction( pszFunctionName );
```

All functions you want to export from the DLL or EXE file, have to be defined in the same way, such as MyLoadFuncs:

Prototype and Implementation

```
RexxFunctionHandler MyFunc;

ULONG MyFunc( CHAR *pszName,     /* Name of the function       */
              ULONG ulNumArgs,   /* Number of args passed      */
              RXSTRING args[],   /* Array of arguments         */
              CHAR *queuename,   /* Current queue name         */
              RXSTRING *retstr ) /* Return value               */
{
}
```

RexxFunctionHandler and RXSTRING are defined in ″rexxsaa.h″. To translate your C program, you can use a definition file (DEF) and make file (MAK) the same as the following:

MYDLL.DEF

```
LIBRARY MYDLL INITINSTANCE LONGNAMES
PROTMODE
DESCRIPTION ′MYDLL can export REXX functions...′
DATA MULTIPLE NONSHARED
STACKSIZE 32768
EXPORTS
   MYFUNC            = MyFunc            @1
   SYSLOADFUNCS      = MyLoadFuncs      @2
   SYSDROPFUNCS      = MyDropFuncs      @3
```

MYDLL.MAK

```
#--------------------------------------------------------------
# IBM Visual Age C++ compiler and linker
#--------------------------------------------------------------
COMP    = ICC
LINK    = ILINK
#--------------------------------------------------------------
# Compiler and linker flags
#--------------------------------------------------------------
CFLAGS  = -c -Ge-
LFLAGS  = /NOFREE
#--------------------------------------------------------------
# File information
#--------------------------------------------------------------
FILE    = MYDLL
#--------------------------------------------------------------
# Target DLL
#--------------------------------------------------------------
$(FILE).dll: $(FILE).obj  $(FILE).def
  $(LINK) $(LFLAGS) $(FILE).obj,$(FILE).dll,,REXX,$(FILE).def;
#--------------------------------------------------------------
# Target OBJ
#--------------------------------------------------------------
$(FILE).obj: $(FILE).c
  $(COMP) $(CFLAGS) $(FILE).c
```

MYDLL.CFG defines some additional libraries which are linked to MYDLL.DLL:

MYDLL.CFG

```
exmpjapi.lib+
exmcjapc.lib+
bidsi.lib+
import.lib+
crtldll.lib
```

″CallC.CMD″ is a small REXX program that illustrates how you load, use, and unload the external functions from your REXX code:

CALLC.CMD

```
/*------------------------------------------------------------*/
/*                                                            */
/* CALLC.CMD loads the external functions from MyDll and      */
/* calls one of them.                                         */
/*                                                            */
/*------------------------------------------------------------*/

/*------------------------------------------------------------*/
/* Load ″MyLoadFuncs″ and execute it.                        */
/*------------------------------------------------------------*/
call RxFuncAdd ′MyLoadFuncs′, ′MYDLL′, ′MyLoadFuncs′
call MyLoadFuncs

/*------------------------------------------------------------*/
/* Call a function, loaded by MyLoadFuncs.                   */
/*------------------------------------------------------------*/
call MyFunc
rc = MyFunc( ′any Text′ )

/*------------------------------------------------------------*/
/* Unload the functions.                                      */
/*------------------------------------------------------------*/
call MyDropFuncs
```

For more information about using and programming REXX code, refer to the on-line help of OS/2 (*OS/2 Procedures Language 2/REXX*) and Visual Age C++ (*REXX Program Reference*).

6.6 User Interface

An important factor for the success of your project is the front end of your application. This is the first thing in your application that the customer and the user will see. Therefore, you have to think about how to present your information on the display, and which controls (buttons, entryfields, and listboxes) you use in the dialogs.

Especially in a workflow and document management project, you always have to keep one thing in mind: The user is accustomed to the manual work environment, which means, for example, the user has an idea how a workbasket or folder is organized. The users will accept your solution more easily if fewer changes are made in the way the data is presented to them. Therefore, ask the users about their daily work and their habits. Never try just to represent your data in the same way you keep it in your data structures.

There are some rules to create a user interface that you should remember:

1. Keep the dialogs simple and use a clear layout.
2. Always use customer′s terms. Only use computer specific terms when necessary.
3. Group your input and output controls in a way that makes sense, and not in the structure you use them in your code.
4. If you use icons, try to use unambiguous ones.
5. Never use many different fonts in your dialogs.
6. Try to use standard dialogs, where possible.
7. Use standard menus, icons, and controls when you design your dialogs.
8. Unify your dialogs and controls.
9. Provide default values in your entryfields, combo boxes, and so on.
10. Never force the user to type the same data several times.
11. Do not pop up too many overlaying dialogs.
12. Provide simple and logical navigation paths through your application.
13. Make sure that you support all of the input devices (keyboard, mouse, and so on) that the customer wants to use. Each control must be accessible with all of the devices.
14. Decide whether your user interface is designed for:
    - Graphics or character display
    - Fixed or various resolutions
    - Monochrome or color displays
    - Small (14″) or large (21″) displays
    - Multiple languages
15. Provide help for your dialogs and controls.
16. Discuss your display layout and all of your dialogs as often as necessary with your customer.

Also, you can refer to guidelines that describe how you have to design a user interface according to standards such as CUA.

When designing user interfaces in a workflow project with products such as FlowMark, there are additional points to think about. FlowMark and all the standard applications you want to integrate have their own specific user interface. If each application just pops up anywhere on the display, your desktop looks very untidy. Even worse, some applications that the user needs may be hidden by others that were started. For example, parts of the document viewer are hidden by a word processor that was started to create a letter according to the contents of the displayed document. The user first has to arrange the application windows manually before the user is able to start writing the letter. This lowers productivity, and the user may not like your solution.

There are several ways to solve these kinds of problems. The easiest one is to arrange the components of FlowMark (worklist, process list), then arrange the other applications′ windows and save their position in each application. This only works if the applications restore their old window positions when they are started again. For each new client workstation you install your solution on, you have to repeat the procedure of arranging windows. From the programmer′s view, this way is less difficult because no code is needed for it. On the other hand, you have nearly no control of the applications running, which may lead to other problems.

Therefore, you may try another way. With techniques such as OLE, you are able to integrate applications on a very high level. You need more code, but you have control of the applications. This permits you to do data exchange in a more simple fashion. If you also use the FlowMark C++ workflow client APIs, you are able to implement your own FlowMark runtime client with all of your needs. In summary, there are several ways that lead to a solution with good usability. You may choose a way with almost no coding and a low level of integration, a way with a high level of integration, or anything between.

The following scenario illustrates an ideal solution. On the client workstation, you have installed your workflow application, your document management system, and other necessary applications. The display is split vertically: on the right side is the document viewer′s window, and on the left you have the integrated runtime client. This runtime client′s window contains a worklist, a simple launch pad for processes, and some dialogs that pop up when needed. This is all that users see when they log on to the system.

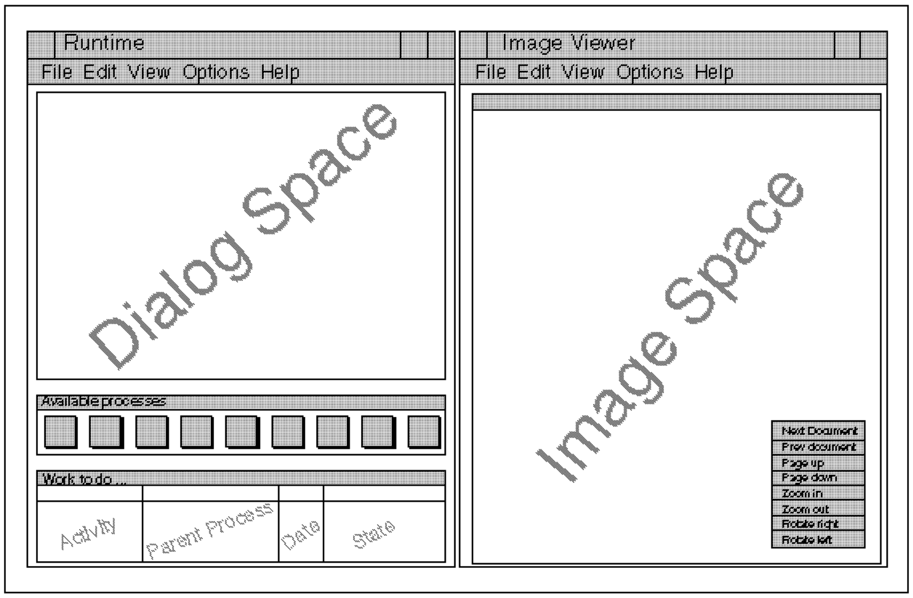


Figure 25. Display Layout

The logon process is designed as a **single logon**, which means that the user enters an ID and password just one time, which will give access to all necessary applications. To design a single logon, you have to know how the needed applications can be started without prompting for logon information. Then you have to know how you can change passwords for a given user ID in the applications. It is also necessary to know the different rules for passwords that the applications use. You have to decide whether you want to use unified passwords for all applications or use different passwords.

Note: The different passwords have to be stored somewhere, or must be derivable from the single logon password.

It is necessary for the system to control the logon to the applications. It must be able to report errors, in case of a logon failure or an abnormal application termination. The system has to ensure that all needed applications run correctly. In case of a fatal error, the system has to log off and close all other applications normally.

The user can perform all actions in the window of the integrated runtime client, except some actions related to the displayed document, such as zoom or print. The integrated runtime client is composed of three parts: the runtime client itself, the user interface to all needed applications, and the message router. The following sections describe the functions of each part.

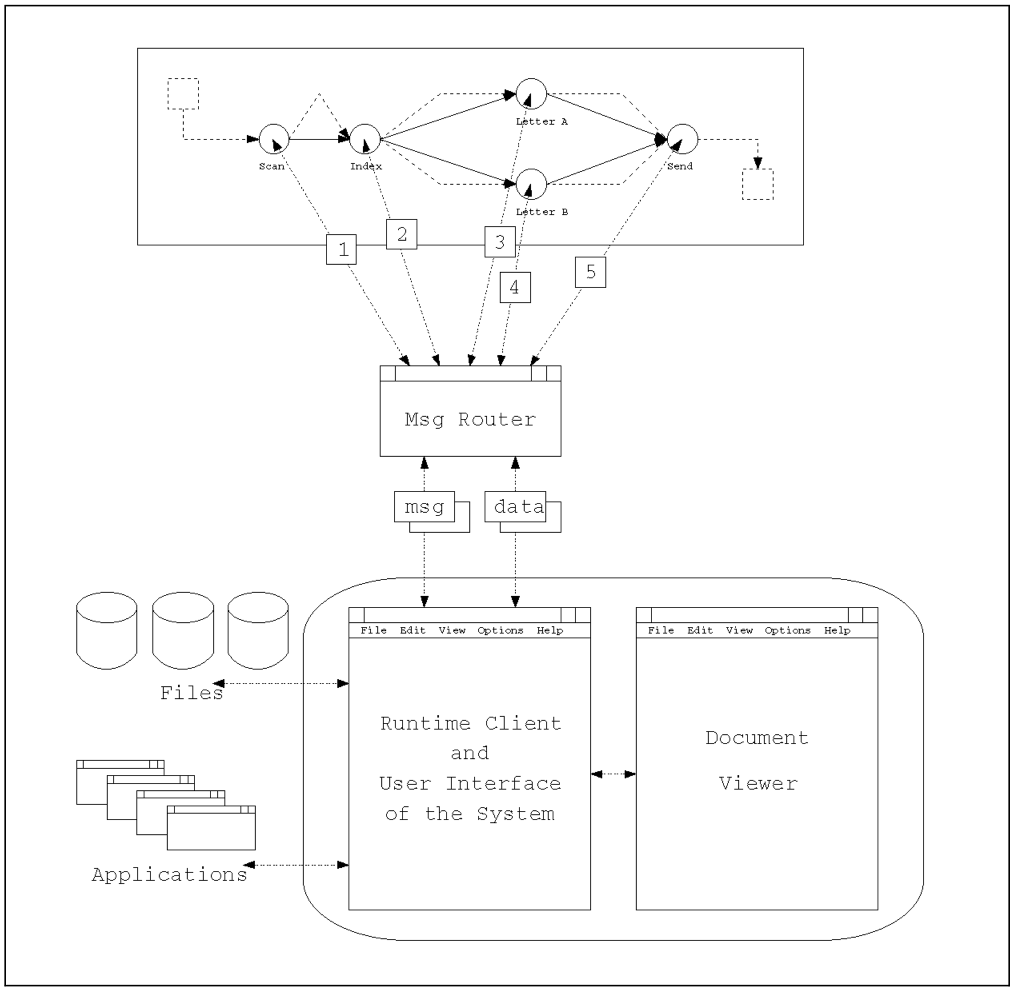

Scan
Index
Letter A
Letter B
Send
1
2
3
4
5
Msg Router
msg
data
Files
Applications
File Edit View Options Help
Runtime Client and User Interface of the System
File Edit View Options Help
Document Viewer

Figure 26. Interface: Dataflow and Communications

1. **Runtime Client**

   The runtime client provides all workflow runtime functions. It has to update and display the work list and the process launch pad. The worklist may be similar to the one used in FlowMark. The process launch pad is a toolbar with icons on it, and a fly-over text with the process name or description.

Both the work list and the process launch pad can be carried out with FlowMark C and C++ APIs and some GUI programming.

It is good to design a highly customizable client, which means that you should use parameters instead of hard-coded values where possible. Store these parameters (color, font, size, position, default user id, timeout, and refresh time in a separate INI file. Do not abuse the system INI file for your data; just store a pointer to your INI file in the system′s INI.

2. **User Interface**

   The application integration part of your system is located here. This part presents data to the user, and is also the link to other applications you use. Once the user is logged on, it has to make sure that all of the required applications are running and available. The central error handling has also to be coded in the integration part. It is a good idea to map the error codes of other applications to a central error code list. Good error handling (message display, error logging) not only helps your users, but also helps in locating errors while testing.

   The interface part has to ensure that just one interactive activity is running on a workstation. An interactive activity is an activity that presents data to the users, or queries data from the user. For batch processing activities such as print or fax jobs, you should define separate user IDs (PRINT1 or FAX1). These non-human users should work on separate workstations, and not on any FlowMark or VisualInfo server, or any workstation for human users.

   Do not code your strings and dialogs in the program; use resource files instead. This simplifies corrections to your layout and texts. To store messages and errors, use predefined file formats.

3. **Message Router**

   The message router is a very small communications program. It is the link between the FlowMark activities and the user interface of the integrated runtime client. In the FlowMark model, you specify the message router as the executable file of an activity.

   The parameter for the message router is just a list of commands, that should be executed by the user interface part. The commands can be plain text such as ″scan″ or ″print″, or coded information such as ″1″, ″a2″, or ″c34″. The message router reads these parameters, and sends it together with some FlowMark activity information to the user interface part. The FlowMark information is the name of the process, the name of the activity, and the FlowMark session ID (needed to read data from the input container or write data to the output container).

   It is up to the interface part to do the right things, when it receives the information: query data items from FlowMark, communicate with other applications, display data, read user input, and write data back to FlowMark. The last thing in an activity is that the user interface part has to inform the message router that all actions have been performed for this activity. Therefore, the message router receives a ″finished″ message and a return code from the user interface part. Once this message is received, the message router sets the return value in the FlowMark container and terminates; the FlowMark activity is finished.

   The message router can communicate in several ways. It can use shared memory, queues, DDE, or messages. A protocol must be defined for the communication. It is used in the message router and in the interface part. Therefore, it is a good idea to do the functions for communication in one DLL file. This DLL can be used by both the message router and the interface part.

If you would like to develop a more document-centric application, you may implement the code of the runtime client and the user interface in the document viewer. In such a solution, the user has only one (VisualInfo) to work with, which may simplify the execution of tasks. To code this integration, use VisualInfo functions such as: SimDspCreateWin, SimDspSetWin, SimDspCreateMenu, SimDspSetMenu. For detailed information, refer to the chapter on ″Display Services Programming Interfaces″ in the IBM VisualInfo Application Programming Reference.

Chapter 7. Hints and Tips

7.1 FlowMark Configuration

FlowMark V2.2 allows you to tailor your FlowMark system by editing the FlowMark profile. All environment settings for FlowMark are held in this one flat file. Thus there is no need to change the config.sys or to re-boot the system if you need to change your FlowMark configuration.

The default profile is a file called **EXMPZCFG.PRF**, and it is located in \EXM\BIN on OS/2, in \EXMWIN\BIN on Windows, and in /usr/lpp/exm/bin on AIX.

FlowMark searches its profile in the current directory first and, if it cannot be found there, the PATH environment variable is searched. Other profiles can be used by coding the ″/f″ option. This option is recognized by all FlowMark components and works similar to this:

```
START component /f=C:\mydir\myprof.prf
```

First, all FlowMark variables are searched in the profile and, if they cannot be found there, they are taken from the operating system′s environment.

The following gives a sample profile.

```
EXM_LANG=EU
EXM_FILES=D:\EXM
EXM_PROTOCOL=TCI,LOC
EXM_TIMEOUT=30000
EXM_HOST=VSL293
EXM_DB_PATH=D:\EXM
EXM_DB_NAME=CREDITDB
EXM_SERVER_NAME=EXMSRV
EXM_RUNTIME_SERVER=CREDITDB,EXMSRV,9.244.70.223,TCI,\
                   TESTDB,TESTSRV,9.244.71.221,TCI
```

You can verify your FlowMark installation using the FlowMark Installation Verification Utility. We recommend that you run this verification utility after installation and after any change to the FlowMark configuration. This helps you find and correct any installation errors and inconsistencies and verifies that:

- Environment variables are set correctly.
- Network drivers are installed properly.
- Network configuration files have been updated.
- The FlowMark profile contains consistent settings.
- Connections between client and server machines can be established.

On OS/2, the executable for the Installation Verification Utility is EXMPZIVT.EXE. On AIX, it is EXMAZIVT, and on Windows 3.x, it is EXMWZIVT.EXE.

7.1.1 Bundle Server Setting

For each database, you can specify one bundle server. Use the EXM_BUNDLE_SERVER setting to define the name of the FlowMark database, the node, and the protocol you want to use. Syntax:

```
EXM_BUNDLE_SERVER = databasename, node, protocol
```

The type of ″node″ is dependent on the protocol you choose. If you have installed a FlowMark stand-alone system (LOC), the node is ′LOCAL′. Using APPC protocol (SNA), you have to enter a fully-qualified CP (Control Point) name. Finally, when working with TCP/IP (TCI), ″node″ is a valid IP (Internet Protocol) address. According to this, ″protocol″ can be one of ″LOC″, ″SNA″, or ″TCI″. The following is an example for one local bundle server:

```
EXM_BUNDLE_SERVER = EXMDB, LOCAL, LOC
```

7.1.2 Code Page Setting

For a Windows environment, use EXM_CCSID to set the needed code page conversion. Syntax:

```
EXM_CCSID = number
```

7.1.3 Database Name Setting

The default FlowMark database name is defined by EXM_DB_NAME. It is used as a default for the Buildtime or Runtime logon dialog, or if no database name is specified when you start from the command line. Syntax:

```
EXM_DB_NAME = databasename
```

If, for example, you use TESTDB as your default database, you have to enter:

```
EXM_DB_NAME = TESTDB
```

7.1.4 Database Path Setting

The path for your database is set in EXM_DB_PATH. The directory you specify has to contain the database schema. If your database is located on an OS/2 server, the extension of the schema file is ″.ADB″; on AIX, it is ″.adb″. The syntax is:

```
EXM_DB_PATH = path
```

Working with ″testdb″ in an AIX environment, the statement looks similar to this:

```
EXM_DB_PATH = /var/exm/db/testdb.adb
```

7.1.5 Working Directory Setting

The FlowMark working directory is specified in the same way, except use EXM_FILES instead of EXM_DB_PATH. FlowMark places files such as message and debug monitor profiles and display services profile into the working directory, It must be located on the local machine.

7.1.6 Database Server Setting

The variable EXM_HOST defines the name of the machine where your database is located.

Note: For a stand-alone installation and for a database machine, EXM_HOST must be blank.

Syntax:

```
EXM_HOST = machinename
```

The value of ″machine name″ is case-sensitive. If your database is located on DBHOST, you have to enter:

```
EXM_HOST = DBHOST
```

The variable EXM_HOST is only needed for ObjectStore clients, but not for other FlowMark components, such as a TelePath server or Runtime client. Buildtime client, Runtime server, and Bundle server are examples of ObjectStore clients. The value of EXM_HOST can be a host name or an IP address if you use the TCP/IP protocol.

7.1.7 Delivery Server Database Recheck Setting

The recheck interval used by the delivery server is set in EXM_RECHECK_INT. The recheck interval is the period when the delivery server starts to scan the database for undelivered messages and retries their delivery. Syntax:

```
EXM_RECHECK_INT = seconds
```

To set the recheck interval to 10 minutes, the profile has to contain:

```
EXM_RECHECK_INT = 600
```

7.1.8 Delivery Server Message Resend Setting

If the delivery server has sent a message to a target but has not received an acknowledgement, the server waits at least the time specified in EXM_RESEND_INT until the message is resent. Syntax:

```
EXM_RESEND_INT = seconds
```

To set the resend interval to 2 minutes, use:

```
EXM_RESEND_INT=120
```

7.1.9 Language Setting

The variable EXM_LANG sets the language version for FlowMark. The default is U.S. English (EU). Syntax:

```
EXM_LANG = languagecode
```

For all available languages codes, refer to *IBM FlowMark Installation V2.2, SH12-6260*. The following line sets the language version of FlowMark to Korean:

```
EXM_LANG = KO
```

7.1.10 Logon Details Setting

The amount of detail downloaded to the FlowMark runtime client during the logon process is set in EXM_LOGON_DETAILS. Syntax:

```
EXM_LOGON_DETAILS = number
```

If EXM_LOGON_DETAILS is not set, everything is downloaded. You can use the following values:

Table 7 (Page 1 of 2). EXM_LOGON_DETAILS Values

Value	Description
0	Nothing is downloaded.

Table 7 (Page 2 of 2). EXM_LOGON_DETAILS Values

Value	Description
1	Download processes only.
2	Download worklists only.
3	Download worklists and processes.
4	Download work items plus worklists.
5	Download processes and work items plus worklists.

The total amount of time needed to download the information to the client workstation cannot be tuned by the EXM_LOGON_DETAILS setting. This means, logging on with EXM_LOGON_DETAILS = 0 and an explicit refresh to get all processes and workitems displayed afterwards takes the same amount of time as downloading everything at logon. To download only worklists at logon time, set:

```
EXM_LOGON_DETAILS = 2
```

7.1.11 Interval Setting for Overdue Notification and Process Cleaning

The intervals between the server checks for overdue activities and processes and finished process instances that can be deleted from worklists are set in EXM_NOTIF_INT and EXM_CLEAN_INT. Syntax:

```
EXM_NOTIF_INT = numberofdays EXM_CLEAN_INT = numberofdays
```

Assuming the interval for overdue checks is two days, and the one for delete is one day, your profile has to contain the following statements:

```
EXM_NOTIF_INT = 2 EXM_CLEAN_INT = 1
```

7.1.12 TelePath Protocol Setting

All supported protocols that the TelePath services should be able to run on, are listed in EXM_PROTOCOL. Syntax:

```
EXM_PROTOCOL = listofprotocols
```

- Local mode: LOC
- APPC communication: SNA
- TCP/IP communication: TCI

If local mode and APPC communications are supported in your environment, specify the following:

```
EXM_PROTOCOL = LOC, SNA
```

7.1.13 Runtime Server Setting

The runtime servers that should be accessible from a client machine are listed in EXM_RUNTIME_SERVER. You have to specify four values for each entry in the list.

```
EXM_RUNTIME_SERVER = databasename, servername, node, protocol
```

To work with the database TESTDB on the FlowMark server EXMSRV (IP address 153.45.236.23), or with CREDIT on BANKSRV (IP address 153.45.236.20) the profile must contain this statement:

```
EXM_RUNTIME_SERVER = TESTDB, EXMSRV, 153.45.236.23, TCI,\
                     CREDIT, BANKSRV, 153.45.236.20, TCI
```

The name of the Server must be supplied in capital letters (only Version 2.2 without any CSDs). The name of the database must be unique if you use more than one database. Using the TCP/IP protocol, you have to use the IP address instead of the hostname. Currently, host names are not supported for this setting.

7.1.14 Database Segment Size Setting

The size of a FlowMark segment in the FlowMark database is defined in EXM_SEGMENT_SIZE. The default value is 200KB. Syntax:

```
EXM_SEGMENT_SIZE = numberofbytes
```

The value depends on the average size of a process and the number of process instances.

7.1.15 Runtime Server Name Setting

The default name for the FlowMark runtime server is defined in EXM_SERVER_NAME. Syntax:

```
EXM_SERVER_NAME = servername
```

If BANKSRV should be your default name for the FlowMark server, the statement must be the same as this:

```
EXM_SERVER_NAME = BANKSRV
```

7.1.16 Logon Timeout Setting

EXM_TIMEOUT defines the timeout value for a FlowMark logon. The default value is three minutes (180000 milliseconds), but when working with large databases, this value should be increased. If the timeout value is too low, you cannot log on to FlowMark. Syntax:

```
EXM_TIMEOUT = milliseconds
```

For example, a timeout value of four minutes:

```
EXM_TIMEOUT = 240000
```

7.1.17 TelePath Keep Setting

Whether TelePath communication services is kept or shut down after the last local application that uses it at this node is stopped is defined in EXM_TP_KEEP. Syntax:

```
EXM_TIMEOUT = yesorno
```

To keep TelePath communication services (YES) is the AIX default value, and services are shut down (NO) is default in an OS/2 or Windows environment.

7.1.18 TelePath Connections Setting

The number of concurrently opened connections by TelePath server is set in EXM_TP_MAX_CONN. The default is 25. Syntax:

```
EXM_TP_MAX_CONN = count
```

The term connection refers to one OS/2 thread (plus resources needed by the communications subsystem for one connection). When you reach the amount of connections specified in EXM_TP_MAX_CONN, the TelePath server closes the connection that was open the longest time and used the least. For example, to increase the number of connections to 30, use:

```
EXM_TP_MAX_CONN = 30
```

7.2 Compacting the ObjectStore Database

The segments in an ObjectStore database consist of extents, which are allocated in the space provided for the database. When there are no free extents left and growth of an ObjectStore segment is required, the ObjectStore server extends the database file to provide the additional space. When you compact a database, you actually free the space that consists of holes in segments for use by other segments in the database.

Note that when you compact a database, it is possible that each segment of the database is modified. Consider this when you back up a compacted database.

To compact the ObjectStore database, invoke the ObjectStore compact utility:

File System	Command
AIX, OS/2 HPFS	**oscompact -dbs_to_compact** *database_name*
OS/2 FAT	**oscompac -dbs_to_compact** *database_name*

Replace *database_name* with the name of the database you want to compact.

7.3 FlowMark Design

Always remember that FlowMark is designed to control the *flow of work* between different users, and not simply to automate the launching of programs for a single user.

When modeling in FlowMark, you should try to minimize the number of activities per process. All tasks that are performed without an interruption in time by the same user on a workstation should be implemented in only one FlowMark activity. This not only simplifies your model, but also increases performance.

Keep your application data in a database and not in the FlowMark data containers. Instead, place your workflow control data and references to your application data there. Often it is not easy to differentiate between pure application data and workflow control data. Therefore, you should ask for each data item if it directly influences the workflow.

If you do this, then multiple users can access the data generated in the process, and not only the single person who is currently working on an activity in the process.

If a process consists of several parts where only a few are used each time, you should divide the process in subprocesses. This avoids having a large process running even though only a part of it is used.

When choosing the right hardware for your solution, remember that you not only need to satisfy the FlowMark requirements, but also the requirements of all tools and applications you use in your workflow.

You have to consider all these requirements, and check the limitations of the operating system you are using.

7.4 VisualInfo Hints

Plan how to use index classes to represent your filing system. How do you want to fill in the documents in your system? Do you need folders; if yes, which ones? Should each folder be a different index class or not?

View Folder content:

- Index class (represented by tabs)
- Attribute names __

Problem: When searching, you can only search one index class (one tab) or all classes. It is not possible to search several index classes at once.

There is no nested search (search for a document which is in multiple subfolders).

Appendix A. FlowMark C++ API Sample

The following sample code shows how the FlowMark Workflow Client C++ APIs can be used to add more functionality to your workflow application. The FlowMark C++ APIs are available for OS/2 and Windows. The sample code listed below has been written for the Windows platform.

Note: Be sure to research the latest fixes for FlowMark (e.g. C++ API under Windows).

Some of the functionality offered by C++ APIs have been used and exported as native C functions in the CLIENT.DLL. This allows you to access these functions easily from Visual Basic or C programs.

Note: The following sample code is not intended to be demonstrate how to write C++ programs. It shows simple usage of the FlowMark C++ API functions.

A.1.1.1 CLIENT.CPP: Sample code **CLIENT.CPP** uses FlowMark Workflow Client C++ API.

Some sample functions that have been implemented are to:

- Logon to a FlowMark server
- Count all process instances in the current worklist
- Create a list of all process instances and their status in the current work list
- Set or query the state of a process instance
- Query properties of a process instance
- Count all work items in the current work list
- Create a list of all work items and their state in the current work list
- Set or query the state of a work item
- Get the input data container of a work item
- Transfer a work item to another user
- Query properties of a work item
- Logoff from a FlowMark server

These functions can be called from an external application to implement a customized FlowMark work list other than the standard work list provided, or to query a list of process instances according to specific criteria, and display these processes and their status in a self-defined format. By fully using these and other functionalities provided by the C++ APIs, you can design and build your own workflow client to FlowMark

```
//-----------------------------------------------------------------
//
//  Sample code for using
//  FlowMark 2.2 C++ Client API
//  with C or VisualBasic
//
//-----------------------------------------------------------------
//
//  C++ functions  -->  LIB & DLL file  -->  C or VB functions
//
//-----------------------------------------------------------------
//
//  File .........: CLIENT.CPP
//
//  Date .........: Feb. 1996
//  Authors ......: Unae Choi, Guido Auberger
//
//-------------------------------------------------------------------


//*******************************************************************
// Includes
//*******************************************************************

#include ”Client.h”          // public defs
#include ”ClientSl.h”        // server list defs
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>



//*******************************************************************
// Globals
//*******************************************************************

ServerAndHandle     *vServers=NULL;
long                ServerVectorSize=0;



//*******************************************************************
// Server functions
//*******************************************************************


//-------------------------------------------------------------------
//  Logon to FM server (default)
//-------------------------------------------------------------------
APIRET APIENTRY ServerLogon( ExmApiBegin  * pexmUserStruct,
                             ServerHandle * phServer )
{
  short           sCountLogin = 10;
  long            lTime = 2000;

  return ServerLogon2( pexmUserStruct, phServer, sCountLogin, lTime );
}
//-------------------------------------------------------------------
//  Logon to FM server (spec. sec & tries)
//-------------------------------------------------------------------
APIRET APIENTRY ServerLogon2( ExmApiBegin  * pexmUserStruct,
                              ServerHandle * phServer,
                              short          sCountLogin,
                              long           lSec )
{
  APIRET          rc;
  ServerAndHandle *pServerAndHandle;

  // Create a server object
  ExmServer *pServer;
  pServer = new ExmServer( pexmUserStruct->Server,
                           pexmUserStruct->Database );
  pServer->SetTimeout( lSec );
  // logon to flowmark
  do
  {
    rc = pServer->Logon( pexmUserStruct->UserID,
                         pexmUserStruct->Password );

    if( EXM_API_OK != rc )
    {
      lSec += 3000;
      pServer->SetTimeout( lSec );
    }
  }
  while( ( EXM_API_OK != rc ) && ( sCountLogin-- ) );

  // logon successful?
  if( EXM_API_OK == rc )
  {
    ServerAdd( pServer, phServer );
  }
  else
  {
    *phServer = 0;
    pServer->Logoff();
    delete pServer;
  }

  return rc;
}
//------------------------------------------------------------------
//  Logoff from FM server
//------------------------------------------------------------------
APIRET APIENTRY ServerLogoff( ServerHandle hServer )
{
  APIRET    rc;
  ExmServer *pServer;

  // Delete ExmServer object from list
  pServer = ServerFind( hServer );
  if( NULL == pServer )
    return EXM_API_ERROR;

  // Logoff
  rc = pServer->Logoff();
  delete pServer;

  // delete server from list
  ServerDelete( hServer );

  return rc;
}



//******************************************************************
// Instance functions
//******************************************************************
```

```
//-------------------------------------------------------------------
//  Count instances in default worklist
//-------------------------------------------------------------------
APIRET APIENTRY InstancesCount( ServerHandle hServer,
                                FmStates     exmState,
                                long *       plCount )
{
  APIRET                        rc;
  ExmServer                     *pServer;
  vector<ExmProcessInstance>    vInstances;
  ExmFilter                     exmFilter(ExmFilter::ProcessInstance);

  // Init
  *plCount = 0;

  // Query ExmServer object from list
  if( NULL == (pServer = ServerFind( hServer ) ) )
    return EXM_API_ERROR;

  // Query process instances
  if( STATE_PRO_QUERY != exmState )
    exmFilter.SpecifyCriterion( ExmFilter::State, exmState );
  rc = pServer->QueryProcessInstances( exmFilter, vInstances );

  if( EXM_API_OK != rc)
    return rc;

  // Prepare output values
  *plCount = vInstances.size();

  return EXM_API_OK;
}
//-------------------------------------------------------------------
//  Create a list of all instances and their status in the
//  default worklist
//-------------------------------------------------------------------
APIRET APIENTRY InstancesList( ServerHandle     hServer,
                               InstanceStruct * pInstances,
                               long *           plCount )
{
  APIRET                        rc;
  ExmServer                     *pServer;
  vector<ExmProcessInstance>    vInstances;
  ExmFilter                     exmFilter(ExmFilter::ProcessInstance);
  long                          lCount;

  // Query ExmServer object from list
  if( NULL == (pServer = ServerFind( hServer ) ) )
    return EXM_API_ERROR;

  // Query process instances
  rc = pServer->QueryProcessInstances( exmFilter, vInstances );
  if( EXM_API_OK != rc)
    return rc;

  // Check size
  if( *plCount < vInstances.size() )
  {
    *plCount = vInstances.size();
    return EXMPJ_RETURN_BUFFER_TOO_SMALL;
  }

  for( lCount = 0; lCount < vInstances.size(); lCount++ )
  {
    pInstances· lCount “.pszInstanceName = strdup( vInstances· lCount “.Name().c_str() );
    pInstances· lCount “.exmInstanceState = vInstances· lCount “.State();
  }

  return EXM_API_OK;
}

//-------------------------------------------------------------------
//  Set and query state of an instance
//-------------------------------------------------------------------
APIRET APIENTRY InstanceState( ServerHandle     hServer,
                               InstanceStruct * pInstance )
{
  APIRET                         rc;
  ExmServer                      *pServer;
  vector<ExmProcessInstance>     vInstances;
  ExmFilter                      exmFilter(ExmFilter::ProcessInstance);
  long                           lCount;

  // Query ExmServer object from list
  if( NULL == (pServer = ServerFind( hServer ) ) )
    return EXM_API_ERROR;

  // Query process instances
  rc = pServer->QueryProcessInstances( exmFilter, vInstances );
  if( EXM_API_OK != rc)
    return rc;

  for( lCount = 0; lCount < vInstances.size(); lCount++ )
  {
    if( vInstances· lCount “.Name() == pInstance->pszInstanceName )
    {
      switch( pInstance->exmInstanceState )
      {
        case STATE_PRO_QUERY:
          pInstance->exmInstanceState = vInstances· lCount “.State();
          return EXM_API_OK;

        case STATE_PRO_READY:
          return vInstances· lCount “.Restart();

        case STATE_PRO_RUNNING:
          return vInstances· lCount “.Start();

        case STATE_PRO_FINISHED:
          return vInstances· lCount “.Terminate();

        case STATE_PRO_TERMINATED:
          return vInstances· lCount “.Terminate();

      }

    }
  }

  return EXM_API_ERROR;
}


//--------------------------------------------------------------------
//  Query properties of an instance
//--------------------------------------------------------------------
APIRET APIENTRY InstanceProperties( ServerHandle     hServer,
                                    char           * pszInstance,
                                    PropInstStruct * pProperties )
{
  APIRET                          rc;
  ExmServer                       *pServer;
  vector<ExmProcessInstance>      vInstances;
  ExmFilter                       exmFilter(ExmFilter::ProcessInstance);
  long                            lCount;

  // Query ExmServer object from list
  if( NULL == (pServer = ServerFind( hServer ) ) )
    return EXM_API_ERROR;

  // Query process instances
  rc = pServer->QueryProcessInstances( exmFilter, vInstances );
  if( EXM_API_OK != rc)
    return rc;

  for( lCount = 0; lCount < vInstances.size(); lCount++ )
  {
    if( vInstances· lCount “.Name() == pszInstance )
    {
      pProperties->StartTime = strdup( ((string) vInstances· lCount “.StartTime()).c_str() );
      pProperties->EndTime = strdup( ((string) vInstances· lCount “.EndTime()).c_str() );
      pProperties->NotificationTime = strdup( ((string) vInstances· lCount “.NotificationTime()).c_str() );
      pProperties->pszStarter = strdup( vInstances· lCount “.Starter().c_str() );
      pProperties->pszCategory = strdup( vInstances· lCount “.Category().c_str() );
      pProperties->pszDescription = strdup( vInstances· lCount “.Description().c_str() );

      return EXM_API_OK;
    }

  }

  return EXM_API_ERROR;
}
//********************************************************************
// WorkItem functions
//********************************************************************


//--------------------------------------------------------------------
//  Count all workitems in the default worklist
//--------------------------------------------------------------------
APIRET APIENTRY WorkItemsCount( ServerHandle hServer,
                                FmStates     exmState,
                                long *       plCount )
{
  APIRET                          rc;
  long                          lCount;
  ExmServer                     *pServer;
  vector<ExmWorklist>           vWorklists;
  ExmWorklist                   WorkList;
  vector<ExmWorkitem>           vWorkitems;
  ExmFilter                     exmFilter(ExmFilter::Workitem);

  // Init
  *plCount = 0;

  // Query ExmServer object from list
  if( NULL == (pServer = ServerFind( hServer ) ) )
    return EXM_API_ERROR;

  // Query Worklists
  rc = pServer->QueryWorklists( vWorklists );
  if (EXM_API_OK != rc )
  {
    return rc;
  }

  // Find the default Work List
  for ( lCount = 0; lCount < vWorklists.size(); lCount++ )
  {
    if ( 0 == vWorklists·lCount“.IsDefault() )
    {
      WorkList = vWorklists·lCount“;
      break;
    }
  }

  // Retrieve Work items
  if( STATE_ACT_QUERY != exmState )
    exmFilter.SpecifyCriterion( ExmFilter::State, exmState );
  rc = WorkList.QueryWorkitems( exmFilter, vWorkitems );

  if (EXM_API_OK != rc )
    return rc;

  // Prepare output
  *plCount = vWorkitems.size();

  return EXM_API_OK;
}
//------------------------------------------------------------------
//  Create a list of all workitems and their state in the
//  default worklist
//------------------------------------------------------------------
APIRET APIENTRY WorkItemsList( ServerHandle     hServer,
                               WorkItemStruct * pWorkItems,
                               long *           plCount )
{
  APIRET                        rc;
  long                          lCount;
  ExmServer                     *pServer;
  vector<ExmWorklist>           vWorklists;
  ExmWorklist                   WorkList;
  vector<ExmWorkitem>           vWorkitems;
  ExmProcessInstance            Instance;
  ExmFilter                     exmFilter(ExmFilter::Workitem);

  // Query ExmServer object from list
  if( NULL == (pServer = ServerFind( hServer ) ) )
    return EXM_API_ERROR;

  // Query Worklists
  rc = pServer->QueryWorklists( vWorklists );
  if (EXM_API_OK != rc )
  {
    return rc;
  }

  // Find the default Work List
  for ( lCount = 0; lCount < vWorklists.size(); lCount++ )
  {
    if ( 0 == vWorklists·lCount“.IsDefault() )
    {
      WorkList = vWorklists·lCount“;
      break;
    }
  }

  // Retrieve all Work items
  rc = WorkList.QueryWorkitems( exmFilter, vWorkitems );

  if (EXM_API_OK != rc )
    return rc;

  // Check size
  if( *plCount < vWorkitems.size() )
  {
    *plCount = vWorkitems.size();
    return EXMPJ_RETURN_BUFFER_TOO_SMALL;
  }

  // Prepare output
  for( lCount = 0; lCount < vWorkitems.size(); lCount++ )
  {
    pWorkItems· lCount “.pszWorkItemName = strdup( vWorkitems· lCount “.Name().c_str() );
    pWorkItems· lCount “.exmWorkItemState = vWorkitems· lCount “.State();
    vWorkitems· lCount “.GetProcessInstance( Instance );
    pWorkItems· lCount “.pszInstanceName = strdup( Instance.Name().c_str() );
  }

  return EXM_API_OK;
}
//-------------------------------------------------------------------
//  Set and Query status of a workitem
//-------------------------------------------------------------------
APIRET APIENTRY WorkItemState( ServerHandle     hServer,
                               WorkItemStruct * pWorkItem )
{
  APIRET                        rc;
  long                          lCount;
  ExmServer                     *pServer;
  vector<ExmWorklist>           vWorklists;
  ExmWorklist                   WorkList;
  vector<ExmWorkitem>           vWorkitems;
```

```
ExmProcessInstance            Instance;
ExmFilter                     exmFilter(ExmFilter::Workitem);

// Query ExmServer object from list
if( NULL == (pServer = ServerFind( hServer ) ) )
  return EXM_API_ERROR;

// Query Worklists
rc = pServer->QueryWorklists( vWorklists );
if (EXM_API_OK != rc )
{
  return rc;
}

// Find the default Work List
for ( lCount = 0; lCount < vWorklists.size(); lCount++ )
{
  if ( 0 == vWorklists·lCount“.IsDefault() )
  {
    WorkList = vWorklists·lCount“;
    break;
  }
}

// Retrieve all Work items
rc = WorkList.QueryWorkitems( exmFilter, vWorkitems );

if (EXM_API_OK != rc )
  return rc;

// Prepare output
for ( lCount = 0; lCount < vWorkitems.size(); lCount++ )
{
  if ( vWorkitems·lCount“.Name() == pWorkItem->pszWorkItemName )
  {

    switch( pWorkItem->exmWorkItemState )
    {
      case STATE_ACT_QUERY:
        pWorkItem->exmWorkItemState = vWorkitems· lCount “.State();
        return EXM_API_OK;
      case STATE_ACT_READY:
        return vWorkitems· lCount “.Restart();

      case STATE_ACT_RUNNING:
        return vWorkitems· lCount “.Start();

      case STATE_ACT_FINISHED:
        return vWorkitems· lCount “.Terminate();

      case STATE_ACT_SUSPENDED:
        // return vWorkitems· lCount “.Suspend();
        return EXM_API_ERROR;

      case STATE_ACT_TERMINATED:
        return vWorkitems· lCount “.Terminate();

    }
  }
  }

  return EXM_API_ERROR;
}


//-----------------------------------------------------------------
//  Get Data of a workitem
//-----------------------------------------------------------------
APIRET APIENTRY WorkItemData( ServerHandle           hServer,
                              WorkItemStruct *       pWorkItem,
                              ExmApiStructureData2 * pData )
{
  APIRET                        rc;
  long                          lCount;
  long                          lVar;
  long                          lSize = (long) pData->Number - 1;
  ExmServer                     *pServer;
  vector<ExmWorklist>           vWorklists;
  ExmWorklist                   WorkList;
  vector<ExmWorkitem>           vWorkitems;
  ExmProcessInstance            Instance;
  ExmReadOnlyContainer          InContainer;
  vector<ExmContainerElement>   vElements;
  ExmFilter                     exmFilter(ExmFilter::Workitem);

  // Query ExmServer object from list
  if( NULL == (pServer = ServerFind( hServer ) ) )
    return EXM_API_ERROR;

  // Query Worklists
  rc = pServer->QueryWorklists( vWorklists );
  if (EXM_API_OK != rc )
  {
    return rc;
  }

  // Find the default Work List
  for ( lCount = 0; lCount < vWorklists.size(); lCount++ )
  {
    if ( 0 == vWorklists·lCount“.IsDefault() )
    {
      WorkList = vWorklists·lCount“;
      break;
    }
  }

  // Retrieve all Workitems
  rc = WorkList.QueryWorkitems( exmFilter, vWorkitems );

  if (EXM_API_OK != rc )
    return rc;

  // Prepare output
  for ( lCount = 0; lCount < vWorkitems.size(); lCount++ )
  {
    if ( vWorkitems·lCount“.Name() == pWorkItem->pszWorkItemName )
    {
      // Workitem found
      ExmReadOnlyContainer          InContainer;
      vector <ExmContainerElement>  vElements;
      string                        strValue;
      long                          lValue;
      float                         fValue;
      char                         *pchHelp;

      // Get input container of workitem
      vWorkitems·lCount“.GetInContainer( InContainer );
      // Get data from input conatiner
      InContainer.Leaves( vElements );

      do
      {
        for( lVar = 0; lVar < vElements.size(); lVar++ )
        {
          if( vElements· lVar “.Name() == pData->MemberData· lSize “.Name )
          {
            // String values
            if( vElements· lVar “.Type() == ″STRING″ )
            {
              vElements· lVar “.GetValue( strValue );
              pData->MemberData· lSize “.DataArea = strdup( strValue.c_str() );
              continue;
            }
            else
            // Long Values
            if( vElements· lVar “.Type() == ″LONG″ )
            {
              vElements· lVar “.GetValue( lValue );
              pchHelp = (char *) malloc( sizeof( long ) );
              *pchHelp = lValue;
              pData->MemberData· lSize “.DataArea = pchHelp;
              continue;
            }
            else
            // Double Values
            if( vElements· lVar “.Type() == ″FLOAT″ )
            {
              vElements· lVar “.GetValue( fValue );
              pchHelp = (char *) malloc( sizeof( float ) );
              *pchHelp = fValue;
              pData->MemberData· lSize “.DataArea = pchHelp;
              continue;
            }

          } // end if names equal

        } // end for elements.count

      }
      while( lSize-- );

      return EXM_API_OK;
    }
  }

  return EXM_API_ERROR;
}
```

```
//------------------------------------------------------------------
//  Transfer a workitem to another user
//------------------------------------------------------------------
APIRET APIENTRY WorkItemTransfer( ServerHandle     hServer,
                                  WorkItemStruct * pWorkItem,
                                  char *           pszTargetUser )
{
  APIRET                        rc;
  long                          lCount;
  ExmServer                     *pServer;
  vector<ExmWorklist>           vWorklists;
  ExmWorklist                   WorkList;
  vector<ExmWorkitem>           vWorkitems;
  ExmProcessInstance            Instance;
  ExmFilter                     exmFilter(ExmFilter::Workitem);

  // Query ExmServer object from list
  if( NULL == (pServer = ServerFind( hServer ) ) )
    return EXM_API_ERROR;

  // Query Worklists
  rc = pServer->QueryWorklists( vWorklists );
  if (EXM_API_OK != rc )
  {
    return rc;
  }

  // Find the default Work List
  for ( lCount = 0; lCount < vWorklists.size(); lCount++ )
  {
    if ( 0 == vWorklists·lCount“.IsDefault() )
    {
      WorkList = vWorklists·lCount“;
      break;
    }
  }

  // Retrieve all Workitems
  rc = WorkList.QueryWorkitems( exmFilter, vWorkitems );

  if (EXM_API_OK != rc )
    return rc;

  // Transfer Workitem
  for ( lCount = 0; lCount < vWorkitems.size(); lCount++ )
  {
    if ( vWorkitems·lCount“.Name() == pWorkItem->pszWorkItemName )
    {
      return vWorkitems·lCount“.Transfer( pszTargetUser );
    }
  }

  return EXM_API_ERROR;
}


//------------------------------------------------------------------
//  Query properties of a workitem
```

```
//----------------------------------------------------------------
APIRET APIENTRY WorkItemProperties( ServerHandle         hServer,
                                    WorkItemStruct *     pWorkItem,
                                    PropWorkItemStruct * pProperties )
{
  APIRET                        rc;
  long                          lCount;
  ExmServer                     *pServer;
  vector<ExmWorklist>           vWorklists;
  ExmWorklist                   WorkList;
  vector<ExmWorkitem>           vWorkitems;
  ExmProcessInstance            Instance;
  ExmFilter                     exmFilter(ExmFilter::Workitem);

  // Query ExmServer object from list
  if( NULL == (pServer = ServerFind( hServer ) ) )
    return EXM_API_ERROR;

  // Query Worklists
  rc = pServer->QueryWorklists( vWorklists );
  if (EXM_API_OK != rc )
  {
    return rc;
  }

  // Find the default Work List
  for ( lCount = 0; lCount < vWorklists.size(); lCount++ )
  {
    if ( 0 == vWorklists·lCount“.IsDefault() )
    {
      WorkList = vWorklists·lCount“;
      break;
    }
  }

  // Retrieve all Workitems
  rc = WorkList.QueryWorkitems( exmFilter, vWorkitems );

  if (EXM_API_OK != rc )
    return rc;

  // Prepare output
  for ( lCount = 0; lCount < vWorkitems.size(); lCount++ )
  {
    if ( vWorkitems·lCount“.Name() == pWorkItem->pszWorkItemName )
    {
      pProperties->StartTime = strdup( ((string) vWorkitems· lCount “.StartTime()).c_str() );
      pProperties->EndTime = strdup( ((string) vWorkitems· lCount “.EndTime()).c_str() );
      pProperties->NotificationTime = strdup( ((string) vWorkitems· lCount “.NotificationTime()).c_str() );
      pProperties->pszOwner = strdup( ((string) vWorkitems· lCount “.Owner()).c_str() );
      pProperties->pszCategory = strdup( ((string) vWorkitems· lCount “.Category()).c_str() );
      pProperties->pszDescription = strdup( ((string) vWorkitems· lCount “.Description()).c_str() );
      pProperties->pszDocumentation = strdup( ((string) vWorkitems· lCount “.Documentation()).c_str() );
      pProperties->ulPriority = vWorkitems· lCount “.Priority();

      return EXM_API_OK;
    }
  }
  return EXM_API_ERROR;
}



//****************************************************************
// Serverlist functions
//****************************************************************


//----------------------------------------------------------------
//  Find a server in the serverlist
//----------------------------------------------------------------
ExmServer * ServerFind( ServerHandle hServer )
{
  unsigned long ulCount;

  // walk through list of servers
  for( ulCount = 0; ulCount < ServerVectorSize; ulCount++ )
  {
    if( vServers· ulCount “.hServer == hServer )
      return( vServers· ulCount “.pServer );
  }

  return( NULL );
}


//----------------------------------------------------------------
//  Delete a server from the serverlist
//----------------------------------------------------------------
APIRET ServerDelete( ServerHandle hServer )
{
  unsigned ulCount;

  // walk through list of servers
  for( ulCount = 0; ulCount < ServerVectorSize; ulCount++ )
  {
    if( vServers· ulCount “.hServer == hServer )
    {
      // delete reference to server
      vServers· ulCount “.pServer = NULL;
      vServers· ulCount “.hServer = 0;
      return EXM_API_OK;
    }
  }

  return EXM_API_ERROR;
}


//----------------------------------------------------------------
//  Add a server to the serverlist
//----------------------------------------------------------------
APIRET ServerAdd( ExmServer *pServer, ServerHandle *phServer )
{
  static long     lServerCount = 0;
  unsigned long   ulCount;
  char            chFound = 0;
  // walk through list of servers
  for( ulCount = 0; ulCount < ServerVectorSize; ulCount++ )
  {
    if( NULL == vServers· ulCount “.pServer )
    {
       // free entry found, use it
       chFound = 1;
       break;
    }
  }

  if( !chFound )
  {
    // expand list of servers
    vServers = (ServerAndHandle *) realloc( vServers, sizeof(ServerAndHandle) * ( ulCount + 1 ) );
    ServerVectorSize++;
  }

  // storage valid
  if( vServers )
  {
    // store server reference
    vServers· ulCount “.hServer = lServerCount++;
    *phServer = vServers· ulCount “.hServer;
    vServers· ulCount “.pServer = pServer;

    return EXM_API_OK;
  }

  return EXM_API_ERROR;
}
```

A.1.1.2 CLIENT.H: **CLIENT.H** has the definitions and includes for CLIENT.CPP for the workflow client APIs:

```
//-----------------------------------------------------------------
//
//  Sample code for using
//  FlowMark 2.2 C++ Client API
//  with C or VisualBasic
//
//-----------------------------------------------------------------
//
//  File .........: CLIENT.H
//
//  Date .........: Feb. 1996
//  Authors ......: Unae Choi, Guido Auberger
//
//-----------------------------------------------------------------


//-----------------------------------------------------------------
// For FlowMark C/C++ API's
//-----------------------------------------------------------------
#define EXM_WIN16               // use 16bit Windows code
#define __MINMAX_DEFINED        // do not use FM min,max defines
```

```
#include <exmwjapc.h>          // FM includes
#include <exmpjapi.h>
#include <exmpjstr.h>
#include <exmpjapi.hxx>

//-------------------------------------------------------------------
// Structure defines
//-------------------------------------------------------------------
typedef short FmStates;
#define STATE_ACT_QUERY           0
#define STATE_ACT_READY           ExmWorkitem::ready
#define STATE_ACT_RUNNING         ExmWorkitem::running
#define STATE_ACT_FINISHED        ExmWorkitem::finished
#define STATE_ACT_SUSPENDED       ExmWorkitem::suspended
#define STATE_ACT_TERMINATED      ExmWorkitem::terminated
#define STATE_ACT_DISABLED        ExmWorkitem::disabled
#define STATE_PRO_QUERY           0
#define STATE_PRO_READY           ExmProcessInstance::ready
#define STATE_PRO_RUNNING         ExmProcessInstance::running
#define STATE_PRO_FINISHED        ExmProcessInstance::finished
#define STATE_PRO_SUSPENDED       ExmProcessInstance::suspended
#define STATE_PRO_TERMINATED      ExmProcessInstance::terminated


//-------------------------------------------------------------------
// Handle to manage servers
//-------------------------------------------------------------------
typedef long  ServerHandle;

//-------------------------------------------------------------------
// State of workitem
//-------------------------------------------------------------------
typedef struct _WorkItemStruct
{
  char      *pszWorkItemName;
  FmStates   exmWorkItemState;
  char      *pszInstanceName;
} WorkItemStruct;

//-------------------------------------------------------------------
// State of instance
//-------------------------------------------------------------------
typedef struct _InstanceStruct
{
  char      *pszInstanceName;
  FmStates   exmInstanceState;
} InstanceStruct;

//-------------------------------------------------------------------
// Properties of instance
//-------------------------------------------------------------------
typedef struct _PropInstStruct
{
  char * StartTime;
  char * EndTime;
  char * NotificationTime;
  char * pszStarter;
  char * pszCategory;
  char * pszDescription;
} PropInstStruct;

//------------------------------------------------------------------
// Properties of workitem
//------------------------------------------------------------------
typedef struct _PropWorkItemStruct
{
  char *        StartTime;
  char *        EndTime;
  char *        NotificationTime;
  char *        pszOwner;
  char *        pszCategory;
  char *        pszDescription;
  char *        pszDocumentation;
  unsigned long ulPriority;
} PropWorkItemStruct;


//------------------------------------------------------------------
// Prototypes of the functions
//------------------------------------------------------------------

//------------------------------------------------------------------
// Server functions
//------------------------------------------------------------------
APIRET APIENTRY ServerLogon( ExmApiBegin * pexmUserStruct,
                             ServerHandle * phServer );

APIRET APIENTRY ServerLogon2( ExmApiBegin  * pexmUserStruct,
                              ServerHandle * phServer,
                              short          sCountLogin,
                              long           lSec );

APIRET APIENTRY ServerLogoff( ServerHandle hServer );

//------------------------------------------------------------------
// Instance functions
//------------------------------------------------------------------
APIRET APIENTRY InstancesCount( ServerHandle hServer,
                                FmStates     exmState,
                                long *       plCount );

APIRET APIENTRY InstancesList( ServerHandle     hServer,
                               InstanceStruct * pInstances,
                               long *           plCount );

APIRET APIENTRY InstanceState( ServerHandle     hServer,
                               InstanceStruct * pInstance );

APIRET APIENTRY InstanceProperties( ServerHandle     hServer,
                                    char           * pszInstance,
                                    PropInstStruct * pProperties );

//------------------------------------------------------------------
// WorkItem functions
//------------------------------------------------------------------
APIRET APIENTRY WorkItemsCount( ServerHandle hServer,
                                FmStates     exmState,
                                long *       plCount );


APIRET APIENTRY WorkItemsList( ServerHandle     hServer,
                               WorkItemStruct * pWorkItems,
                               long *           plCount );

APIRET APIENTRY WorkItemState( ServerHandle     hServer,
                               WorkItemStruct * pWorkItem );

APIRET APIENTRY WorkItemData( ServerHandle           hServer,
                              WorkItemStruct *       pWorkItem,
                              ExmApiStructureData2 * pData );

APIRET APIENTRY WorkItemTransfer( ServerHandle     hServer,
                                  WorkItemStruct * pWorkItem,
                                  char *           pszTargetUser );

APIRET APIENTRY WorkItemProperties( ServerHandle         hServer,
                                    WorkItemStruct *     pWorkItem,
                                    PropWorkItemStruct * pProperties );
```

A.1.1.3 CLIENTSL.H:

This code has the remaining definitions and includes for CLIENT.CPP:

```
//-----------------------------------------------------------------
//
//  Sample code for using
//  FlowMark 2.2 C++ Client API
//  with C or VisualBasic
//
//-----------------------------------------------------------------
//
//  File .........: CLIENTSL.H
//
//  Date .........: Feb. 1996
//  Authors ......: Unae Choi, Guido Auberger
//
//-----------------------------------------------------------------


//-----------------------------------------------------------------
// Type defs
//-----------------------------------------------------------------
typedef struct _ServerAndHandle
{
  ExmServer     *pServer;
  ServerHandle  hServer;
} ServerAndHandle;


//-----------------------------------------------------------------
// Prototypes
//-----------------------------------------------------------------
ExmServer * ServerFind( ServerHandle hServer );
APIRET ServerDelete( ServerHandle hServer );
APIRET ServerAdd( ExmServer *pServer, ServerHandle *hServer );
```

A.1.1.4 CLIENT.DEF:

CLIENT.DEF is the definition file for CLIENT.CPP used in CLIENT.MAK:

```
LIBRARY     CLIENT

DESCRIPTION 'FlowMark Workflow Client API sample code'

HEAPSIZE    16384
```

A.1.1.5 CLIENT.MAK: **CLIENT.MAK** is the makefile for CLIENT.DLL:

```
#-----------------------------------------------------------------
#
#  Sample code for using
#  FlowMark 2.2 C++ Client API
#  with C or VisualBasic
#
#-----------------------------------------------------------------
#
#  File .........: CLIENT.MAK
#
#  Date .........: Feb. 1996
#  Authors ......: Unae Choi, Guido Auberger
#
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# Borland C++ 4.5 compiler and linker
#-----------------------------------------------------------------
IMPLIB  = Implib
BCC     = Bcc
TLINK   = TLink

#-----------------------------------------------------------------
# Debug flags
#-----------------------------------------------------------------
CDEBUG   = -v
LDEBUG   = /v

#-----------------------------------------------------------------
# Compiler flags
#-----------------------------------------------------------------
CFLAGS16 = -ml
CFLAGS  =  $(CFLAGS16) -c $(CDEBUG) -3 -tWDE

#-----------------------------------------------------------------
# Linker flags
#-----------------------------------------------------------------
LFLAGS  = /d /Twd

#-----------------------------------------------------------------
# Path information
#-----------------------------------------------------------------
PATHBC  = C:\APPL\WIN\BC45
PATHFM  = C:\APPL\WIN\EXMWIN\API

#-----------------------------------------------------------------
# File information
#-----------------------------------------------------------------
FILE    = Client


#-----------------------------------------------------------------
# Target LIB
#-----------------------------------------------------------------
$(FILE).lib : $(FILE).dll
  $(IMPLIB) $@ $(FILE).dll
```

```
#------------------------------------------------------------------
# Target DLL
#------------------------------------------------------------------
$(FILE).dll : $(FILE).obj
  $(TLINK) $(LDEBUG) -L$(PATHBC)\LIB;$(PATHFM) $(LFLAGS) \
 c0dl.obj+$(FILE).obj, $(FILE).dll,, @$(FILE).cfg, $(FILE).def

#------------------------------------------------------------------
# Target OBJ
#------------------------------------------------------------------
$(FILE).obj : $(FILE).cpp $(FILE).mak
  $(BCC) $(CFLAGS) -I$(PATHBC)\INCLUDE;$(PATHFM) -D_RTLDLL;_BIDSDLL; $(FILE).cpp
```

A.1.1.6 CLIENT.CFG:

CLIENT.CFG defines some additional libraries which are linked to CLIENT.DLL:

```
exmpjapi.lib+
exmcjapc.lib+
bidsi.lib+
import.lib+
crtldll.lib
```

Appendix B. Special Notices

This publication is intended to help technical people from IBM, customers and business partners with the planning and implementation of a workflow and document management system based on FlowMark and VisualInfo by providing the steps that were taken in an actual IBM services project at a customer site. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM FlowMark and IBM VisualInfo. See the PUBLICATIONS section of the IBM Programming Announcement for FlowMark and VisualInfo 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.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

IBM

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Appendix C. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How To Get ITSO Redbooks" on page 121.

- *FlowMark V2.2 Design Guidelines*, SG24-4613
- *FlowMark Installation and Administration*, SG24-4614
- *A Simple Approach to VisualInfo*, GG24-4444
- *VisualInfo Sample Code for Client*, GG24-4369
- *VisualInfo: CID Installation*, GG24-4415

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

C.2 Other Publications

C.2.1 FlowMark

These publications are also relevant as further information sources.

- *IBM FlowMark*: Modeling Workflow, SH19-8241, describes the concepts of workflow modeling and how you can use FlowMark to define your model.
- *IBM FlowMark*: Managing Your Workflow, SH19-8243, introduces the FlowMark Runtime Client and shows how you can use it.
- *IBM FlowMark*: Programming Guide, SH19-8240, explains the FlowMark APIs for C, C++, REXX, Visual Basic and Cobol
- *IBM FlowMark*: Installation V2 R2, SH12-6260, shows how you install, administrate and maintain your FlowMark system.
- *IBM FlowMark*: Diagnosis Guide, SH19-8239, contains information on how you can correct problems when you install or use FlowMark.
- *IBM FlowMark: Application Integration V2R2*, SH12-6267, explains the service broker and building blocks concepts.

C.2.2 VisualInfo

- *IBM ImagePlus VisualInfo General Information and Planning Guide*, GK2T-1709, a overview of the VisualInfo system and information about planning your system.
- *IBM ImagePlus VisualInfo Installation Guide*, GK2T-1710, shows how you can install VisualInfo on your system.
- *IBM ImagePlus VisualInfo Administration Guide*, SC31-7661, explains how to administrate your VisualInfo system.

- *IBM ImagePlus VisualInfo User′s Guide*, SC31-7670
- *IBM ImagePlus VisualInfo Application Programming Guide, Volume 1*: Folder Manager, Application, and Library Interfaces, SC31-7662
- *IBM ImagePlus VisualInfo Application Programming Guide, Volume 2*: Image Services Interface, SC31-7664
- *IBM ImagePlus VisualInfo Application Programming Reference, Volume 1*: Folder Manager and Application Interfaces, SC31-7663
- *IBM ImagePlus VisualInfo Application Programming Reference, Volume 2*: Image Services Interface, SC31-7682
- *IBM ImagePlus VisualInfo Application Programming Reference, Volume 3*: Common Data Structures and Database Tables, SC31-7665
- *IBM ImagePlus VisualInfo Application Programming Reference, Volume 4*: Library Interfaces, SC31-7667

C.2.3 Other Publications

- *VisualView Image Client*: User′s Guide
- *VisualView Image Client*: Technical Reference Manual
- *Workflow Management Coalition Application Programming Interface 2 (WAPI) Specification*, document number WFMC-TC-1009
- *IBM Communications Manager/2*: V1.11 EHLLAPI Programming Reference, document number SC31-6163

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 URL 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 lists of redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  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:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page 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.

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** (Do not send credit card information over the Internet) — 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	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **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) 415 855 43 29 (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 Home Page	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).

IBM Redbook Order Form

Please send me the following:

Title	**Order Number**	**Quantity**

- **Please put me on the mailing list for updated versions of the IBM Redbook Catalog.**

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.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

List of Terms and Abbreviations

Note: For more information about terms, refer to the product publications found in Appendix C, “Related Publications” on page 119.

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 LOVEM-ProModeler: on a PLOVC or JLOVC, an activity is a manual process path component that receives input from an upstream component, acts upon it, and sends output to a downstream component. Activities can be added to manual bands.
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 runtime. The number of instances of an activity is determined during runtime by a special program activity called the planning activity. See also *planning activity*.

AIF. Application Integration Features: a CIC/ESA product which helps you integrate your existing applications without writing any code. Applications communicate across platforms using MQSeries for MVS/ESA, which keeps them insulated from network complexities.

AIR. See *assumptions/issues/recommendations*.

ALOVC. See *architecture line of visibility chart*.

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.

architecture line of visibility chart (ALOVC). The graphical representation of the essential customer and enterprise processes and key data needs and their main characteristics arranged in sequence. See also *enterprise architecture*.

As Is. The current state of a process or process path. See also *To Be*.

As Is view. A chart or diagram showing the current state of a process or process path. See also *To Be view*.

assumptions/issues/recommendations (AIR). In LOVEM-ProModeler: a business control parameter that can contain assumptions, issues, and recommendations.

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. The FlowMark Audit Trail server logs events that occur for each database tha contains process models and is started. These events are recorded throughout the day in an audit trail.

automation band. The horizontal section on a PLOVC or JLOVC that contains systems, system functions, or computer data stores.

Automation Manager. The Automation Manager is responsible for connection of remote clients to OLE servers.

B

band. The horizontal sections of an LOVC. Examples of bands:

- Customer band
- Internal/external organization band
- Internal business area or department and job bands
- Manual/automation band
- Automation band.

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, VisualInfo, Lotus Notes. This is the product called via the Service Broker Manager. Synonym for *cprtner product* and *companion product*.

benchmarking. Comparing something with a standard such as 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*.

blueprint. The exact graphical representation of **As Is** business processes or **To Be** designs. A blueprint can be created at the architecture, logical, physical, or job level. It can be used for implementing new processes and for ongoing process management.

blueprinting. The procedure used to document a company′s process design in graphical form using ALOVCs, LLOVCs, PLOVCs, or JLOVCs.

BOB. See *best of breed*.

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*.

Buildtime. In FlowMark: the component used to define processes. See also *Runtime*.

bundle. In FlowMark: a type of block that supports multiple instances of a single program or process activity at runtime. The number of instances of the activity is determined at runtime 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 Runtime. 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) that 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 area is part of the logical model (LLOVC).

business control parameter. Goals, predictions, targets, observations, and measurements, of the enterprise or individual organization units, such as CSFs, AIRs, and CMPs. These business objects are of primary importance to the purpose of the enterprise and an can be assigned to processes and activities in logical and physical models and blueprints.

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

CABE. Computer Aided Business Engineering.

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.

CAPI. The VisualInfo primary programming interfaces are called Common Application Programming Interface.

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.

change management bus. On the ALOVC, a continuum of repetitive and unpredictable processes for enabling customers to request and affect changes (for example, a proposal, a contract, or an order at any time during the relationship).

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*.

CIF. Common Interchange File; The CIF file specifies attribute names needed to import documents and folders to the VisualInfo system from other platforms.

CIU file. Common Interchange Unit File

client. Clients access shared network resources or functionality provided by a server.

COLD. Computer Output to Laser Disc

common specification language. A means of communicating across various functions, organizational units, or levels of management in a precise language using graphical representations, such as the four types of LOVCs.

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. Pertaining to, or characteristic of, the PLOVC and JLOVC. 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*.

control flow. In LOVEM: on PLOVCs and JLOVCs, a trigger that is generated by an activity. It shows the flow of control from one activity or task to another, provided the transition condition, if specified, is true. See also *information flow* and *material flow*.

In process-based applications: a control flow can consist of a workflow, a task flow, and an event flow. See also *workflow, task flow*, and *event flow*.

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 or not 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.

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 satisfaction can be graphically shown on LOVCs by an icon indicating whether a customer is ***happy*** or ***unhappy*** with the operations and results of an activity.

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

DASD. Direct Access Storage Device; A device in which access time is effectively independent of the location of the data

data bus. On an ALOVC or a LLOVC, a logical set of data. A logical, dynamic data store. It starts at the beginning of a logical model, such as ALOVC and LLOVC and continues until the end of the model. It reflects the most current data at any point in a relationship with a customer. Examples of data buses:

- Contact or customer data bus
- Offering or products and services data bus
- Promise or order and contract data bus

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 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 relation 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.

default connector. In FlowMark: the graphical representation of a special kind of control connector, shown as in a process diagram. Control flows along this connector if no other control path is valid. See also *connector, control connector*, and *transition condition*.

department. A subdivision of an enterprise that shows reporting lines and performs one or more activities. Department is part of the physical models, such as PLOVC or HSD.

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.

DLL. Dynamic link library. A module containing a routine that is linked at load time or runtime. 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. A VisualInfo document consists of a base part and the item information, notes with content class, annotation with content class, history and MGDS data (used with OCR). It is indexed by system-defined and user-defined attributes. A document can be contained in one or more folders.

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. Can be part of the physical models or blueprints, such as PLOVC and JLOVC. See also *data store*.

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*.

EDM. electronic document management.

electronic data interchange (EDI). A computer-to-computer connection between two companies. The transaction flow for EDI transaction is regulated through international protocol.

empower. To provide employees with the authority to make decisions.

enterprise. An organization, usually comprised of several lines of business, whose purpose it is to perform a mission and to achieve goals by working with customers and suppliers.

enterprise architecture. A business process architecture at the enterprise level as expressed through an ALOVC. It is at the logical, unconstrained level and shows the essential process and data needs in sequential form. See also *architecture line of visibility chart (ALOVC)*.

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 of a PLOVC or JLOVC 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. In a PLOVC or JLOVC, 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

FAT. OS/2 or DOS file allocation table.

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 Runtime 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 runtime database to a FRL file. Using the import utility program EXMPFRIM.EXE, a FRL file can be imported into a FlowMark runtime database.

folder. An item that can contain other folders or documents. In the VisualInfo system, a folder can be indexed by system-defined and user-defined attributes.

formalism. The strict attention to rules and symbols (for example, the LOVC rules and symbols).

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

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, such as 3270 or 5250 screen formats. FlowMark makes use of the HLLAPI building block under OS/2 or MS Windows.

HPFS. OS/2 high performance file system

HSD. See *hierarchical structure diagram*.

I

IBM. International Business Machines Corporation

icon. A picture that represents the actual image of information flow media or means of transportation, such as a telephone, truck, or computer terminal.

index class. A user-defined group of information used to store and retrieve an item or set of items. An index class identifies the type of key fields, automatic processing requirements, and storage requirement for a document or folder.

information. Facts or objects that have meaning to human beings; as opposed to *data*, which requires context and interpretation in order to become *information*. Information is formatted data. Business objects that are produced by or moved between activities and systems using information flows.

information flow. An ordered packet of data. Input to, or output from, any object on a PLOVC, or JLOVC, such as an order, a shipping document, or an invoice. Information flows can use various media, such as FAX machines, telephones, or electronic mail, which can be represented on the LOVCs by icons . See also *data flow*, *material flow*, and *control flow*.

information system. See *system*.

input container. In FlowMark: storage for data used as an input for activities, processes, or blocks. See also *output container*.

in-process server. A OLE server that is implemented as a dynamic link library is called in-process server, it runs in the client′s adress space.

inquiry management bus. On the ALOVC, a continuum of repetitive and unpredictable processes for enabling customers to request information on anything about the company, its products and services, or a past service encounter (for example, an inquiry about a proposal, a contract, or an order). See also *bus* and *change management bus*.

internal interface. An interface between two internal business functions, departments, or jobs where a dependency exists or a transfer takes place.

IOCA. Image Object Content Architecture; A collection of constructs used to interchange and present images.

issue. A controversial matter that needs to be discussed and resolved.

IT. Information Technology. The hardware, software, services and facilities which a company uses to store, process and retrieve data and information.

ITSO. International Technical Support Organization.

J

JLOVC. See *job line of visibility chart*.

job. The physical implementation of business processes, as expressed through manual activities and interfaces with customers, systems, and internal or external organizations. A series of one or more activities in a department that are performed by one employee. A job is represented by the JLOVC and can be part of a PLOVC. Examples of a job: marketing representative or customer service representative.

job line of visibility chart (JLOVC). A physical modeling and blueprinting technique that shows the *effectiveness and efficiency* of one particular job. Using the JLOVC, you focus on *how* an individual job is or will be implemented, and *who* is or will be executing the manual activities. It also shows the relationship of each activity to customer activities, systems, and internal or external organizations. See also *line of visibility chart*.

L

line of business (LOB). A family of either, or both, products and services, having common characteristics.

line of visibility (LOV). On a LOVC, the line between your customer and internal processes or activities where all points of contact (service encounters) are shown. See also *service encounter*.

line of visibility chart (LOVC). The graphical representation of all aspects of your business processes that are required to provide your customer with a specific product or service. It shows all points of contact with your customer. The LOVC is implemented in four different types of charts:

- ALOVC. See *architecture line of visibility chart*.
- LLOVC. See *logical line of visibility chart*.
- PLOVC. See *physical line of visibility chart*.
- JLOVC. See *job line of visibility chart*.

Line of Visibility Engineering Methodology (LOVEM). See *IBM LOVEM*.

Line of Visibility Enterprise Modeling (LOVEM). See *IBM LOVEM*.

LLOVC. See *logical line of visibility chart*.

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 line of visibility chart (LLOVC). A logical modeling or blueprinting technique that shows the *effectiveness* of the process path that you are studying. Using this technique, you focus on *what* needs to be done, not *how* it is done. See also *line of visibility chart*.

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 on a PLOVC or JLOVC. 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 on a PLOVC or JLOVC, which 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.

LOV. See *line of visibility*.

LOVC. See *line of visibility chart*.

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.

Note: There are two levels of IBM LOVEM:

1. Line of Visibility Engineering Methodology

   This level contains the full methodology, which is documented in the IBM LOVEM Consultant′s Guide, and which is only available to IBM consultants.

2. Line of Visibility Enterprise Modeling

   This level contains the graphics and applications of the methodology that are implemented in ProModeler. This level is documented in the IBM LOVEM User′s Guide and is available to the general public.

LOVEM. See *IBM LOVEM*.

LTL. See *logical transformation list*.

M

manual/automation interface. The manual-automation line on a PLOVC or JLOVC between manual activities and systems. Systems placed on this lines show user interactions with the systems. Systems placed below this line are batch systems with no direct user interaction.

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.

In LOVEM: input to, or output from, an activity or system on a PLOVC or JLOVC; for example, a car, a mortgage document, or a consultant′s report. The mode of transportation, such as person, airplane, or truck, can also be shown on the diagram as icons. See also *information flow, data flow, document flow*, and *control flow*.

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.

MGDS. Machine Generated Data Stream

mission. A general statement of the purpose and nature of an enterprise.

MO:DCA. Mixed Object Document Content Architecture. An IBM architecture developed to allow the interchange of object data among applications within the interchange environment and among environments.

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.

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*.

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.

O

OCR. Optical Character Recognition

OLE. Object Linking and Embedding

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.

ObjectStore. The database manager for FlowMark V2.2 and previous releases.

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. ObjectStore.

out-of-process server. An OLE server running in its own process space is called out-of-process server. This kind of OLE server is implemented as an executable file.

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*.

P

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*.

patch code. Industry standard pattern of horizontal lines; You can use patch codes to indicate the beginning of a new folder or document.

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*.

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 line of visibility chart (PLOVC). A business process modeling or blueprinting technique that shows the *effectiveness and efficiency* of a process path. A PLOVC is a sequence of physical business objects, such as activities and systems. See also *physical* and *line of visibility chart*.

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 runtime, 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.

PLOVC. See *physical line of visibility chart*.

PPDS. Page Printer Data Stream

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 *sub-process*.
In LOVEM-ProModeler: a process is a logical component on an ALOVC or LLOVC.
In FlowMark: a process is a 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 Runtime.

process management. In FlowMark: the Runtime 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 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 quality management bus. On the ALOVC, a continuum across the enterprise process path to capture any process quality parameters, such as CMPs, CSFs, goals, strategies, or policies in relation to customer service encounters. See also *bus*.

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 Runtime.

PROFS. Professional Office System. Predecessor to IBM OfficeVision.

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*.

proxy. A proxy is an interface specific object, that is able to package parameters for an interface to prepare for a remote method call. It runs in the address space of the caller and communicates with the corresponding stub on the remote computer.

Q

R

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.

REXX. Restructured extended executor language. A procedures language.

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.

RPC. Remote Procedure Call; Using RPC, an application is able to call routines belonging to an application hosted on another computer that is connected via a network.

runtime. In FlowMark: the component, used to execute process instances. The Runtime component consists of:

- Runtime server
- Program execution client
- Bundle server
- Notification service
- Delivery server
- Runtime client.

See also *Buildtime* and *Runtime client*.

runtime client. In FlowMark: the user interface for working with process templates, process instances, work lists, and work items. See also *Runtime*.

S

SB. service broker

SBM. Service Broker Manager

screen layout. The design and specifications of the image that the user sees on the screen of a system. See also *report layout*.

server. A server is a system that is designed to share data with client applications. Servers and clients are often connected via a network or may be simply located on the same computer.

service. A collection of related features that respond to requests for specific activities or yield information, is called service. The services are accessed through a consistent and published interface of the service that encapsulates its implementation.

In the Service Broker Architecture, a service DLL contains service functions which interface with the base product. A service function receives the user data from the Service Broker Manager and calls the appropriate product APIs to perform the work. The results are returned via the Service Broker Manager to the service requester and then back to the user application. a base product that is integrated. Logon session data is passed to the service via the Service Broker Manager. The services use this logon session data when invoking the product APIs.

Service Broker Architecture. The Service Broker Architecture 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. The aim is to allow 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.

Service Broker Manager. A FlowMark component that controls the operation of service broker sessions. This includes the interaction between service requester and services, between service broker and services, and also the intialization of the service brokers and services.

service encounter. Any point of contact with your customer. See also *moment of truth*.

service requester. A service requester is the interface to the user application. The user application calls the service requester APIs to request the base product to perform some work. The service requester formats the user data and issues a request to the service broker which invokes the appropriate service function via service threads.

service thread. One or more service threads are started for each service. Each thread receives information from the service requester to call the respective service function. When the function has been called, the thread returns information to the service requester.

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.

source. In FlowMark: the symbol that represents the input container of an activity, process, or block. See also *input container* and *sink*.

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*.

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*.

start criteria. A control setting for activities on PLOVCs or JLOVCs that determines when an activity or a process path can start. See also *exit criteria*.

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.

sub-process. 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 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.

TIFF. Tag Image File Format

time line. A notation on the PLOVC and the JLOVC that shows the time between individual activities as well as for the entire process path or job; for example, the order cycle time. The time line shows both actual (**As Is**) and target (**To Be**) times.

To Be. The desired state of a process or process path: how it could be or should be. See also *As Is*.

To Be view. A chart or diagram showing the desired state of a process or process path. See also *As Is view*.

TOC. Table Of Contents

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

unconstrained. Pertaining to, or characteristic of, the ALOVC and LLOVC techniques, or representative of the factors that define *what* a process is doing not *how* it is being done. See also *logical*.

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*.

V

VHLPI. VisualInfo high-level programming interface. The service broker for VisualInfo. It can be used to integrate FlowMark with VisualInfo for document management connectivity and services.

VisualBasic. A programming language under MS Windows.

VisualInfo. An IBM product for document management.

W

WISDDM. World-wide integrated solution design and delivery methodology. WISDDM is a set of I/T methodologies, such as:

- The application development methodology, which is based on:
  - Solution/2000
  - End-to-End
  - Full Life-Cycle Testing
  - Redevelopment
  - Package Selection.
- The project management methodology, which is based on:
  - Managing Implementation of the Total Project
  - Project Management for Project Executives
  - Other IBM project management approaches.

WF. workflow

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.

WFMC. Workflow Management Coalition.

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.

Index

Printed in U.S.A.

SG24-4712-00