Designing an Application Using IBM WebSphere Host Publisher

Before You Begin

This paper assumes that you have a general knowledge of IBM Host Publisher functions and terminology. Before continuing, you should at least understand these concepts:

- Integration Objects
- Macros: connect, data, and disconnect
- Integration Object chaining
- Connection pooling

If you are not comfortable with these terms, you may want to review the following chapters and sections of the Host Publisher Administrator's and User's Guide, available through a Host Publisher installation or from Host Publisher's external Web site (http://www.ibm.com/software/network/hostpublisher/library):

- Chapter 1. About Host Publisher, What is Host Publisher?
- Chapter 2. Using Host Publisher, Overview
- Chapter 2. Using Host Publisher, Using Host Publisher Studio, Creating an Integration Object for Host Access
- Chapter 3. Advanced Features, Using Integration Object chaining

Overview

Now that you have decided to use IBM Host Publisher to help you design and implement your Web or Java application for accessing and distributing host data, you are probably eager to jump right into Host Publisher Studio and begin the development process. Not so fast! You need to make sure you know what you are doing first.

If you decide to start designing your application using Host Publisher Studio, you'll probably find that your design changes several times and are spending most of your time trying to modify or fix what you've done so far. Host Publisher Studio is meant to help you implement your design, not to help you with the actual design.

There are two design points of which you must be absolutely certain before you begin using Host Publisher Studio. You must completely understand the host application you will be accessing and you must know what you want to do with the data once you've accessed it.

Understand the Host Application

You must understand the data source well enough to model a complete, repeatable, and predictable interaction with it. If you don't, you will end up spending a great deal of time learning about the data source while you are also learning Host Publisher. You will also end up wasting time debugging your mistakes, time that should have been spent carefully considering how the application should look and behave.

You are using Host Publisher because you have a task in mind. You probably have information stored in a host application, and you want more people to have easier access to this information. *What is this data? How do you get access to it? How will your end users get access to it?*

The first question is the easiest to answer: What is this data? This is the information that you want to expose using Host Publisher, whether it is a table of employee information, billing information, account information; it doesn't matter - it's all information.

The second question is more difficult to answer: How do you get access to it? Host Publisher helps you create entities called Integration Objects. These are JavaBeans which encapsulate interactions with your host application and are responsible for getting to and retrieving the information you want. You must, therefore, have to know how to get to it.

Integration Objects model tasks. You tell Integration Objects to perform the task, using certain information you provide as input (last names, menu choices, search criteria, and so on), and it returns to you the results of that task (tables of data, for example). You have to know what encompasses that task. What screens do you navigate for that task? What menu choices have to be made? What input do you require of the user to complete that task?

The last original question deals with connectivity issues: How will the end users get access to it? You might provide your own user ID and password to gain access to a host system to run the application. How will the general user of your new application get access to the system? Is the host application dependent on the identity of the user of the application? How you answer these questions not only affects how the application is deployed, but also how the Integration Objects are modeled. You have to consider where the process of connecting to the system ends and the application begins. Does the application include logging on to the system, or will the connect macro handle this?

We will explore the answers to these questions in more detail later.

Using the Data

What are you planning to do with the data once it has been retrieved by Integration Objects? If you plan on building a series of Web pages for interacting with this data, you have visual elements and logical flow to consider. If you are building a Java application to include Integration Object JavaBeans, you will be processing the data through another Java application and you won't be concerned with building Web pages for interacting with this data. There is no visual element or Web page flow to worry about, but aspects of understanding and designing the flow between Integration Objects is still vital.

One mistake that is easy to make is to attempt to use Host Publisher to create another interface to a host application with no attempt whatsoever being made to hide the complexity of the application flow. There is some usefulness to this: You don't have to train your end users how to use the host system and you are now making host applications available to end users using a more familiar and friendly medium, the Web browser. In reality, you are not doing the end user much of a favor.

Part of the advantage of using Host Publisher is the ability to hide the complexity of the back-end data source. You have the ability to condense multiple host screens down to one or two Web pages: one to perhaps ask a few questions, the other to show the results. The customer is not aware that several screens and decisions were made along the way to getting the resulting information.

After spending so much time studying the host application, it is now time to step away from the host environment and forget the host application's flow. If you don't, you are in danger of having it influence your Web application design.

You must now think about how the end user should interact with the data. How should the data be represented? What does the Web application need to know from the end user to get to that data? If you create enough Web pages to only answer these two basic questions, you'll give the user what he needs. You'll also make it easier on yourself in the development process by not trying to reproduce the host application screen by screen on the Web.

The Design Process

Designing the Integration Objects and the ultimate Web application involves these steps:

1. **Identify the tasks to be performed in the host application.** These tasks may or may not return data. They may instead involve updating information stored on the host, or execution of some function.
2. **Map the flows of the tasks.** Create a flow chart with every node in the chart either being a screen or a decision point (menu, for example).
3. **Identify the inputs and outputs in the flow.** What screens require information from the user? What data is being extracted?
4. **Identify the decision points of the task flows that require user feedback.** In other words, the decision cannot be made before the task begins, but instead must be made based on some results that the task has produced up to that point. The flows between such decision points are subtasks. There is a one-to-one mapping between Integration Objects and these subtasks.
5. **Map out the Integration Objects.** Now list the Integration Objects: where they start, where they stop, what they require as input, and what they provide as output. Also, uniquely name and label each one.
6. **Create the Web application page flow.** Using something similar to another flow diagram, create the page flow required to drive the Integration Objects you've mapped in number 5.
7. You are now ready to use Host Publisher Studio to implement your design.

If you do not have any plans to use Integration Objects in Web applications, but instead plan on incorporating them into other Java applications, you only require steps 1 through 5 and only part of step 7 (building the Integration Objects). We will also make an assumption that Integration Object chaining is required to accurately and efficiently model the host application. Most host applications are highly interactive, requiring user feedback throughout the application flow. Without Integration Object chaining, instead of each Integration Object picking up where the previous one left off, each Integration Object has to start over, reproducing the same steps as the previous Integration Object from the same starting point, adding on the additional steps required of its task. This is error-prone and highly inefficient; therefore, Integration Object chaining is the implementation of choice for these types of application. There are certainly times where Integration Object chaining is not required, such as simple queries or updates where all the required information can be gathered up front. There are also inherent complexities involved in designing Web applications using Integration Object chaining (such as keeping track of start and stop states), so first trying to apply Integration Objects to the task without using chaining is a good idea. The easiest way to explain these steps in more detail is to illustrate by example. We will use a library catalog host application as our example and we will create a design for a Web application that allows end users to look up book titles, reserve a title, and request new titles to be added to the catalog.

Identify the Tasks

The host application we are modeling in Host Publisher is a typical university library catalog application. Students, faculty, and staff of the university use this application to search for a book title, reserve the title for pickup later, and request that the library obtain a specific title.

The problem is that the 3270 terminals for accessing this application are only in the library itself. Library administrators want to use Host Publisher to create a Web application for use by students and faculty from their dorm rooms and offices. The university dormitories and office buildings are already wired with an Ethernet network with Internet access; such a Web application is a great fit.

There are three main tasks available through this catalog application, and want to make all of them available through the Web application. The tasks are:

1. Search for a book title based on various search criteria. The user selects a title from the search results and views details on the title and/or reserves that title.
2. View books currently reserved and/or checked out by a user.
3. Request that a title be added to the library.

The application doesn't require a user ID or password, nor does the host system. If you request that a book be reserved, check on what books are currently reserved, or request that a title be added to the library, you must specify your university identification number.

Map the Task Flows

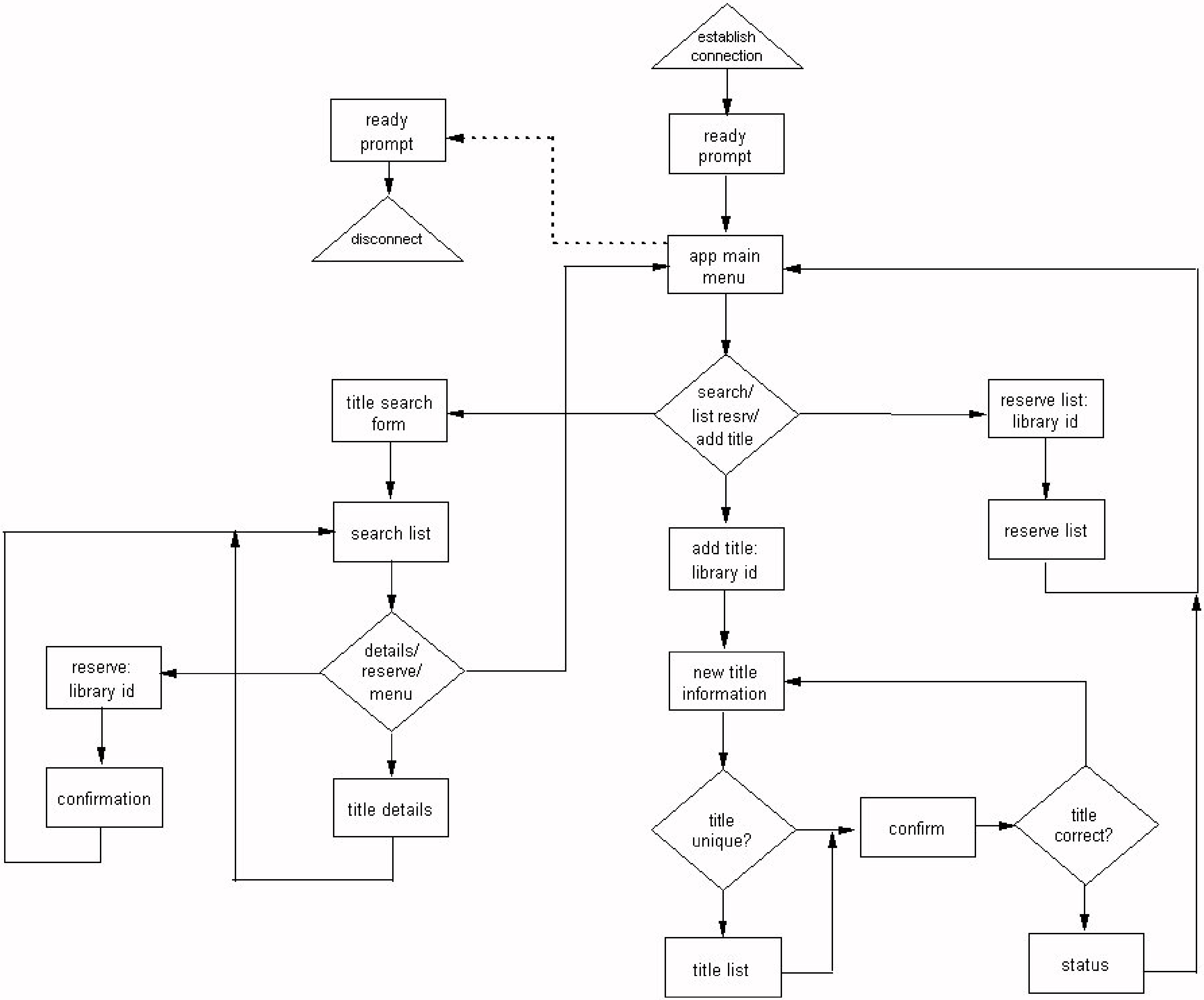
Now we know what our three primary tasks are, it's now time to map the flow of those tasks in terms of screens and decision points. We also need to consider the flow required to connect and log into, as well as disconnect from, the system.

Now is also a good time to begin thinking about how the application will be deployed. We highly recommend that you consider using connection pooling to improve end-user response time, which means you must determine how far down the connect path you can take a new connection so that it can be reused by many different users. If you don't use connection pooling, a new connection is created for every new application request. This means that the end user has to wait for extra processing to occur before the first Integration Object can be run.

As mentioned before, if your application relies on the identity of the person using it, you probably need to make the system logon part of your application. That means that the connect portion of the application (the flow that is done only once per connection before it is reused many times by the application) involves establishing the connection, not logging on. There is no connect macro. As an alternative, you may choose not to use connection pooling and contain the logon in the connect macro. Integration Objects must always request a new connection, instead of using one from a pool of available connections.

If your application is not dependent on the identity of the individual logging into the system, or if your system does not require a logon, such as in our example, then it is advantageous to take the connect macro as far into the application as possible. This reduces the number of screens that are required to be navigated by Integration Objects. In our example, when new connections are created by Host Publisher Server, the connection is established and the associated connect macro drives the connection to the host ready prompt. This is a good general starting point for every new instance of our application.

We will map the flow using a flow chart. The flow covers the entire application, including all three tasks as well as connection and disconnection from the system. The following flow chart contains the flow for the university library catalog application.



The arrows indicate direction of flow. The rectangles are screens, diamonds are decision points, triangles are the connection establishment and termination (non-visual activity), and the dotted line indicates the point from which the disconnection occurs. Users cannot choose to disconnect from the system; disconnection occurs when the connection is removed from the pool, or when Host Publisher Server is stopped.

The application works as follows: After connecting to the system, the user is presented with a welcome screen, or ready prompt, where the catalog application name is entered. The catalog main menu offers three choices, which are our three tasks: Search for a title, view titles currently on reserve, and request a new title to be added to the library.

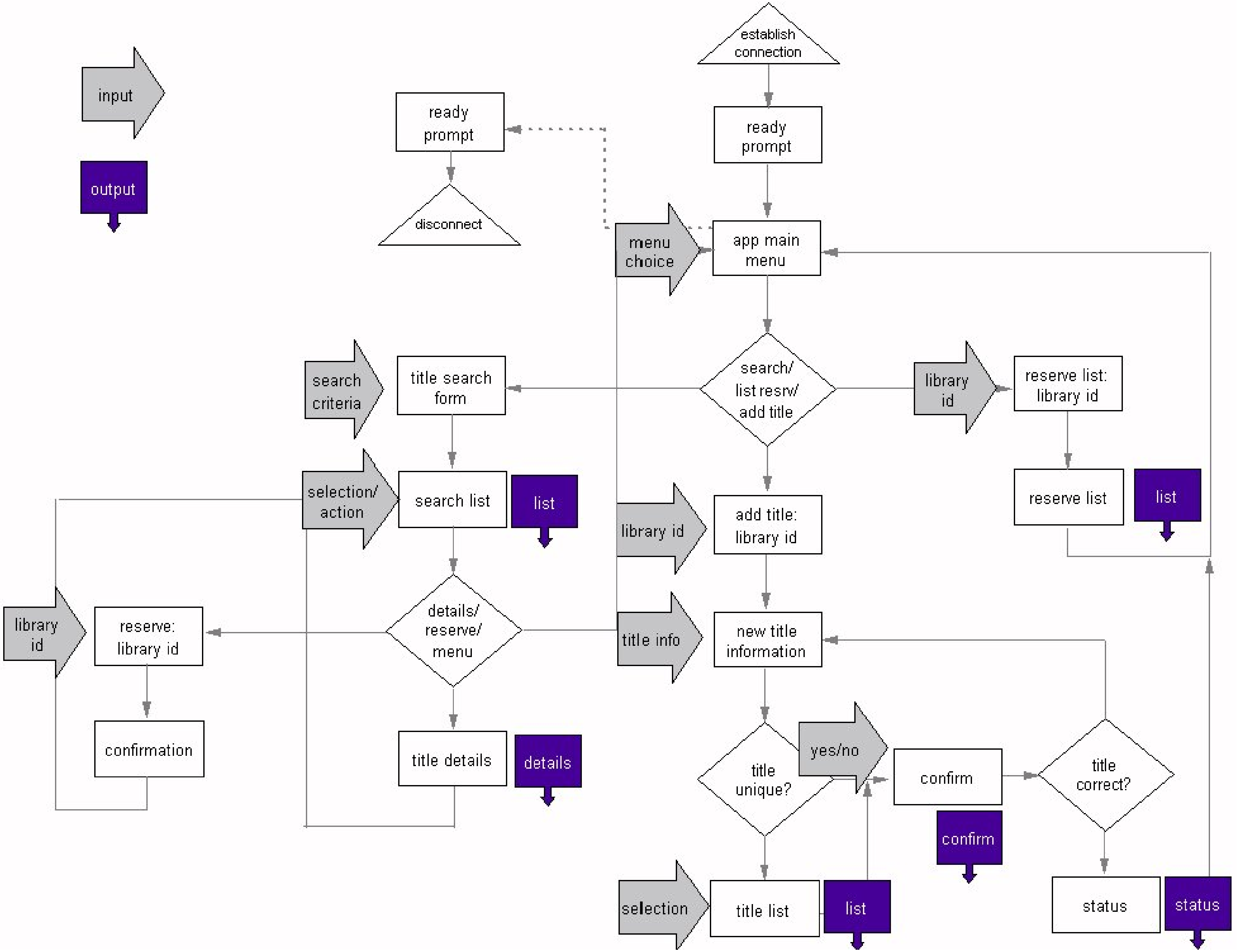
If the user selects to search for a title, he is presented with a search screen where he enters criteria for the search. Search criteria include any combination of title words, author, or topic keywords. The application returns a list of book titles meeting the specified criteria. From this list, the user may select to view details on a specific title and/or reserve a copy of the book for pickup later. After viewing title details or adding a title to the reserve list, the user is returned to the search results screen where he can either select another title or return to the main menu.

If the user selects to view the reserve list from the main menu, he is presented with a screen where he must enter his library ID. The resulting screen lists all of the titles which are currently reserved by that library ID. To remove a title from the reserve list, the user must see a library attendant; he cannot remove titles from the reserve list himself. After viewing the reserve list, the user is taken back to the main menu.

If the user selects to add a new title to the library, he is presented with a screen requesting his library ID. After the ID is validated, the next screen requests information about the book, such as title, author, topic, and any additional information which is available (publisher and year, for example). The application then searches its title database looking for matches. If more than one match is made, a list of matches appears, and the user is asked to select one. Once a unique choice is made, the application presents the title's details and asks the user to verify that the information is correct. If so, the application then verifies the status of the title: whether or not the title is already in the library's catalog, whether or not it is still in print, and whether or not a copy can be obtained, then instructs the user to check back later for status.

Identify Inputs and Outputs

Now that the application flow is understood, we must identify where input is required and output desired. We'll take our original flow diagram and add input and output indicators to it.

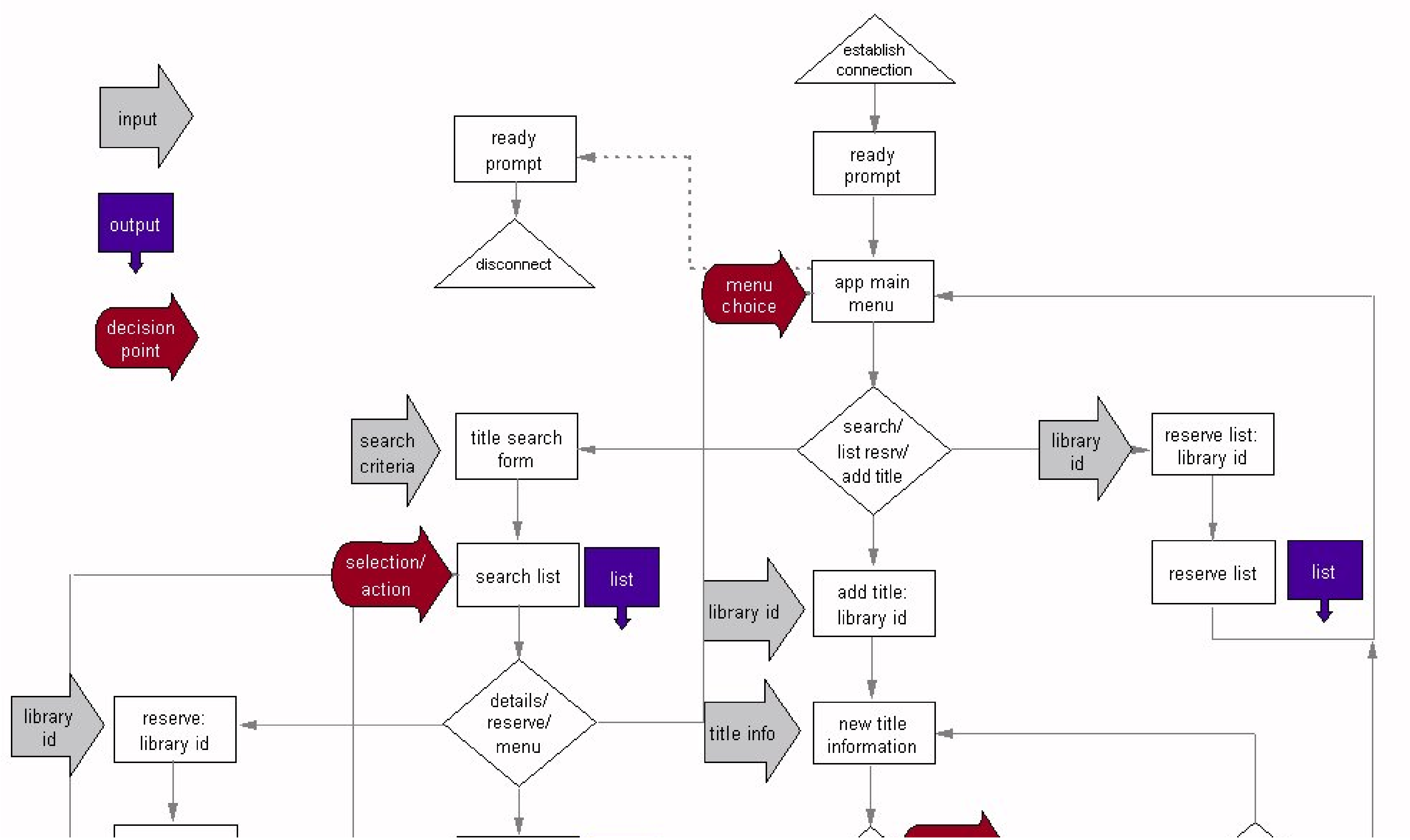


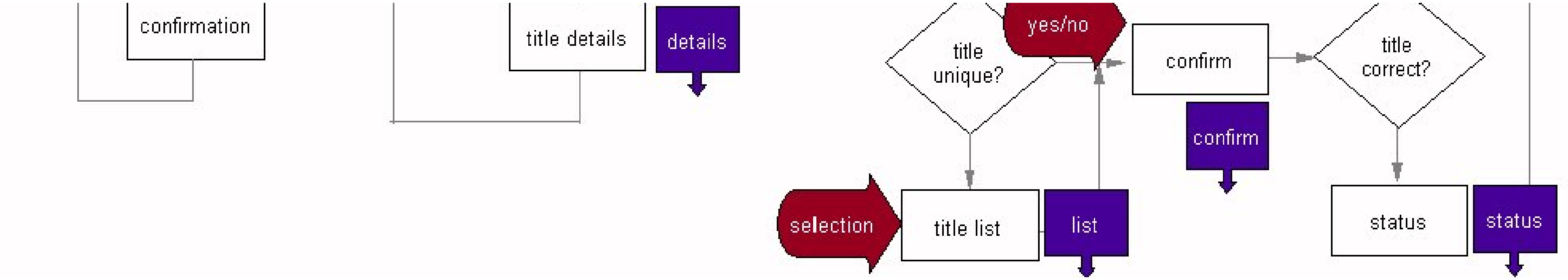
Notice that we don't extract output from all possible screens where output might be interesting; for example, the confirmation screen in the lower left hand corner of the diagram. You might think that extracting the confirmation message would be useful so you could display that message on the Web page and allow the user to react to it; however, this only causes unnecessary overhead, especially if you can reproduce the confirmation yourself. Don't get trapped by the assumption that everything the application creates should be reproduced on the Web page. You'll end up reproducing the host application instead of focusing on completing the task and extracting the data.

Now that we've identified what data musts be provided to the application from the user (input), and what data we are interested in getting in return (output), we are a step closer to defining the Integration Object boundaries within the flow. The next step involves grouping inputs that can be simultaneously requested, and identifying the major decision points requiring user interaction.

Identify Decision Points

Let's revisit the application flow diagram with the inputs and output. We must identify those inputs that require data based on provided output. For example, down the search path for a book title, a search result list is generated from which the user makes a selection and either views its details or reserves it. This is a decision point because a decision must be made (selection, and action) based upon data extracted as output (search results list). The only other decision point that doesn't act on output is the main menu selection; the difference here is that the menu is created manually in the Web page instead of being based on screen output. Here's the updated flow diagram:





Notice that we have five major decision points. The flows between the decision points represent the subtasks that the Integration Objects must perform. After counting the flows between the decision points, and taking into account the permutations caused by the conditional diamonds, we see that there are going to be eight Integration Objects in the application, plus one Integration Object to exit the application, for a total of nine Integration Objects. See if you can arrive at the same number.

Next, we'll split the flow into the three major tasks for clarity then begin designing the Integration Objects to map to the subtasks we've just identified.

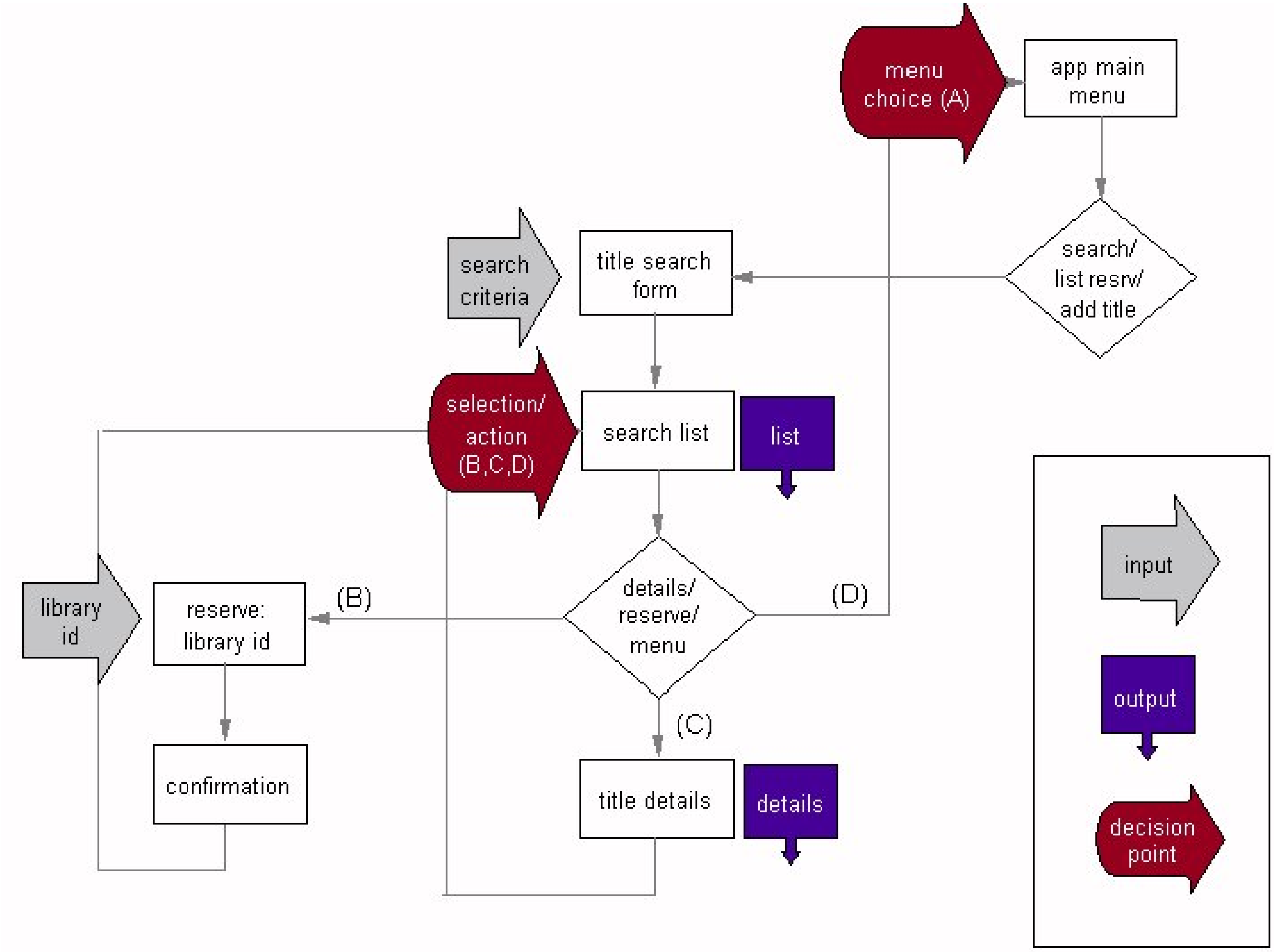
Map out the Integration Objects

When we design an Integration Object, we specify the following information:

- **Integration Object name**. It is important to choose a name that is meaningful for its task, as well as unique from any other Integration Object's name. Usually, you will include characters to identify the application you are building, in addition to the task the Integration Object performs, such as LCA for Library Catalog Application.
- **Start and stop state label**. These two labels identify the screen, or state, of the host application where the Integration Object starts and ends, respectively. The label does not have to coincide with screen names or titles, but you may want to consider doing so.
- **Inputs**. This is the list of items expected as input to this Integration Object. These inputs must be satisfied before the Integration Object can start its task (run its data macro).
- **Outputs**. This is the data expected to be returned by the Integration Object at the completion of its task.
- **Description**. Include a brief description of the task that the Integration Object will perform.

Before we begin designing the Integration Objects, let's break apart our flow diagram to match the three major tasks we identified in the beginning. In each diagram, we will label the starting point of an Integration Object with a letter. Letters appear as part of the major decision points in the flow diagrams. We also need to design the Integration Objects that will serve as the first and last Integration Objects of the chain. These are the Integration Objects that get you in to and out of the host application. We'll then design each Integration Object by letter and by task.

Task 1: Search for a Book Title



Integration Objects

A) **Name:** LCASearchForTitle

Start state: LCA Main Menu

Stop state: Search List Results

Inputs: title, author, topic, publisher, date

Outputs: search results (table)

Description: select search menu option; enter provided search criteria and start search; extract search results as a table

B) **Name:** LCAReserveTitle

Start state: Search List Results

Stop state: Search List Results

Inputs: title selection, library ID

Outputs: none

Description: select specific title from list; select to reserve that title; enter library ID; return to search results

C) **Name:** LCATitleDetails

Start state: Search List Results

Stop state: Search List Results

Inputs: title selection

Outputs: title details

Description: select specific title from list; select to show details on that title; extract detail text; return to search results

D) **Name:** LCASearchReturnToMenu

Start state: Search List Results

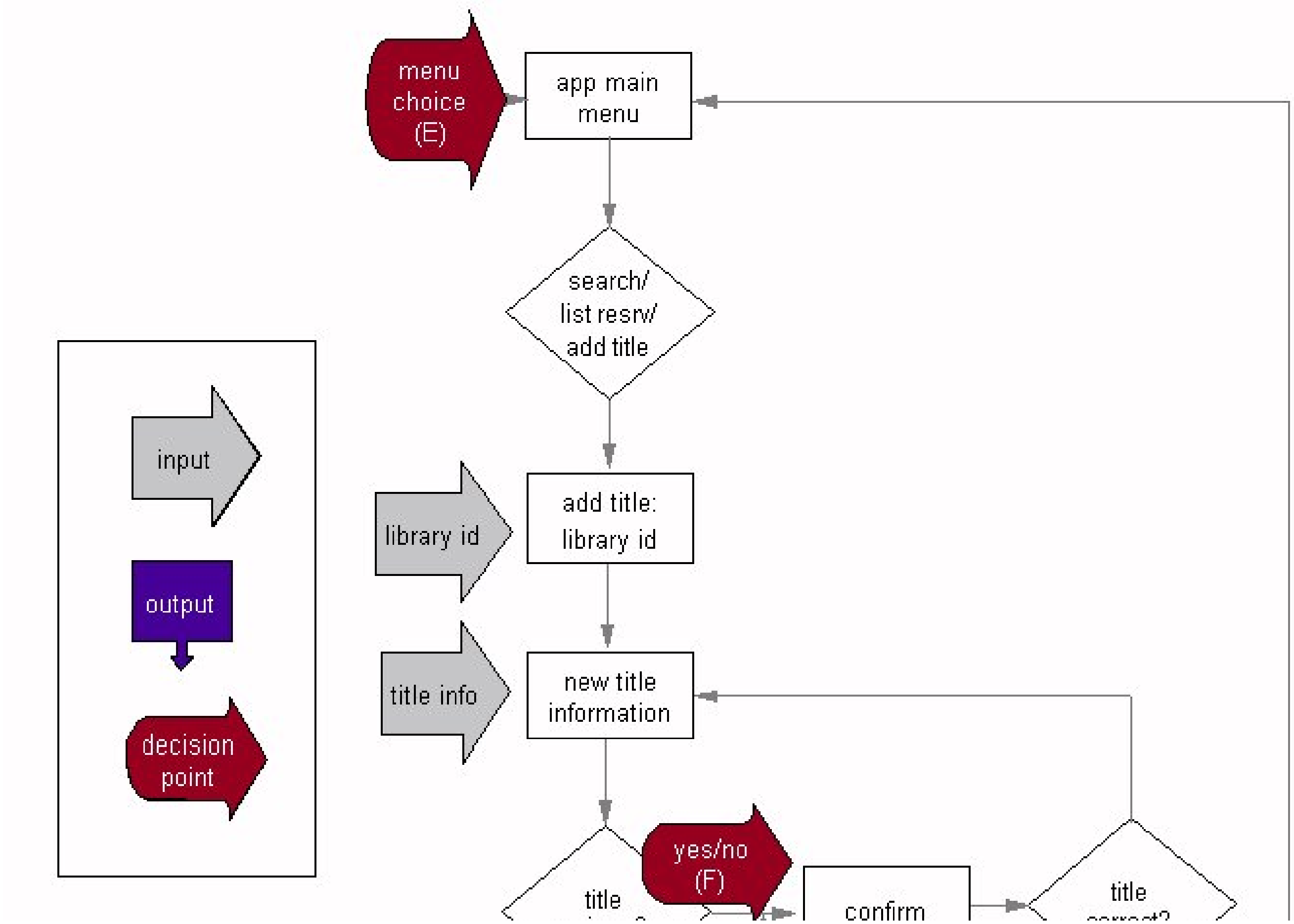
Stop state: LCA Main Menu

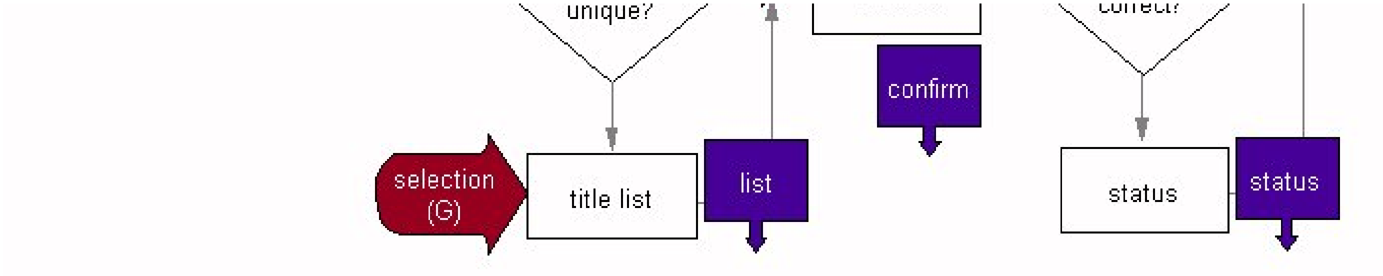
Inputs: none

Outputs: none

Description: select to return to the main menu

Task 2: Request a New Title





Integration Objects

E) **Name:** LCANewTitleRequest

Start state: LCA Main Menu

Stop state: New Title

Inputs: library ID, title, author, topic, publisher, date

Outputs: search results (table) or confirm details

Description: select request new title menu option; enter provided library ID and search criteria and start search; if the criteria was not unique enough, return table of title matches; if it was unique, extract confirm details

F) **Name:** LCANewTitleConfirm

Start state: New Title

Stop state: LCA Main Menu

Inputs: confirm

Outputs: status

Description: confirm that selected title is the appropriate title; if no, return to main menu; if yes, extract status message for display, then return to main menu

G) **Name:** LCANewTitleSelection

Start state: New Title

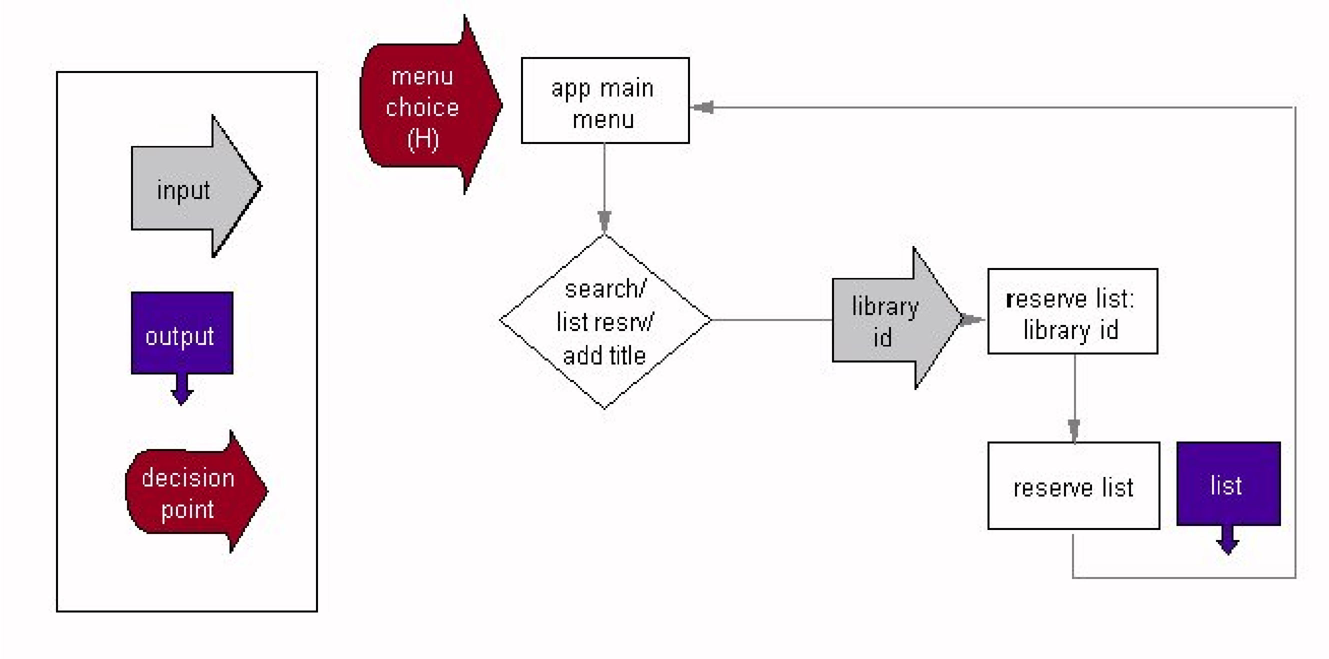
Stop state: New Title

Inputs: title selection

Outputs: confirm details

Description: selects title from a list of title matches; extract confirm details and automatically confirm that this is the correct title; extract status message for display, then return to main menu

Task 3: Review Reserve List



Integration Objects

H) **Name:** LCAReviewReserveList

Start state: LCA Main Menu

Stop state: LCA Main Menu

Inputs: library ID

Outputs: list of books on reserve (table)

Description: select reserve list menu option; enter provided library ID; extract list of titles currently reserved by the given ID; return to the main menu

Entering and Exiting the Application

Integration Objects

I) **Name:** LCAEnter

Start state: (n/a)

Stop state: LCA Main Menu

Inputs: none

Outputs: none

Description: enter the library catalog application at the ready prompt; take user to the application's main menu

J) **Name:** LCAExit

Start state: LCA Main Menu

Stop state: (n/a)

Inputs: none

Outputs: none

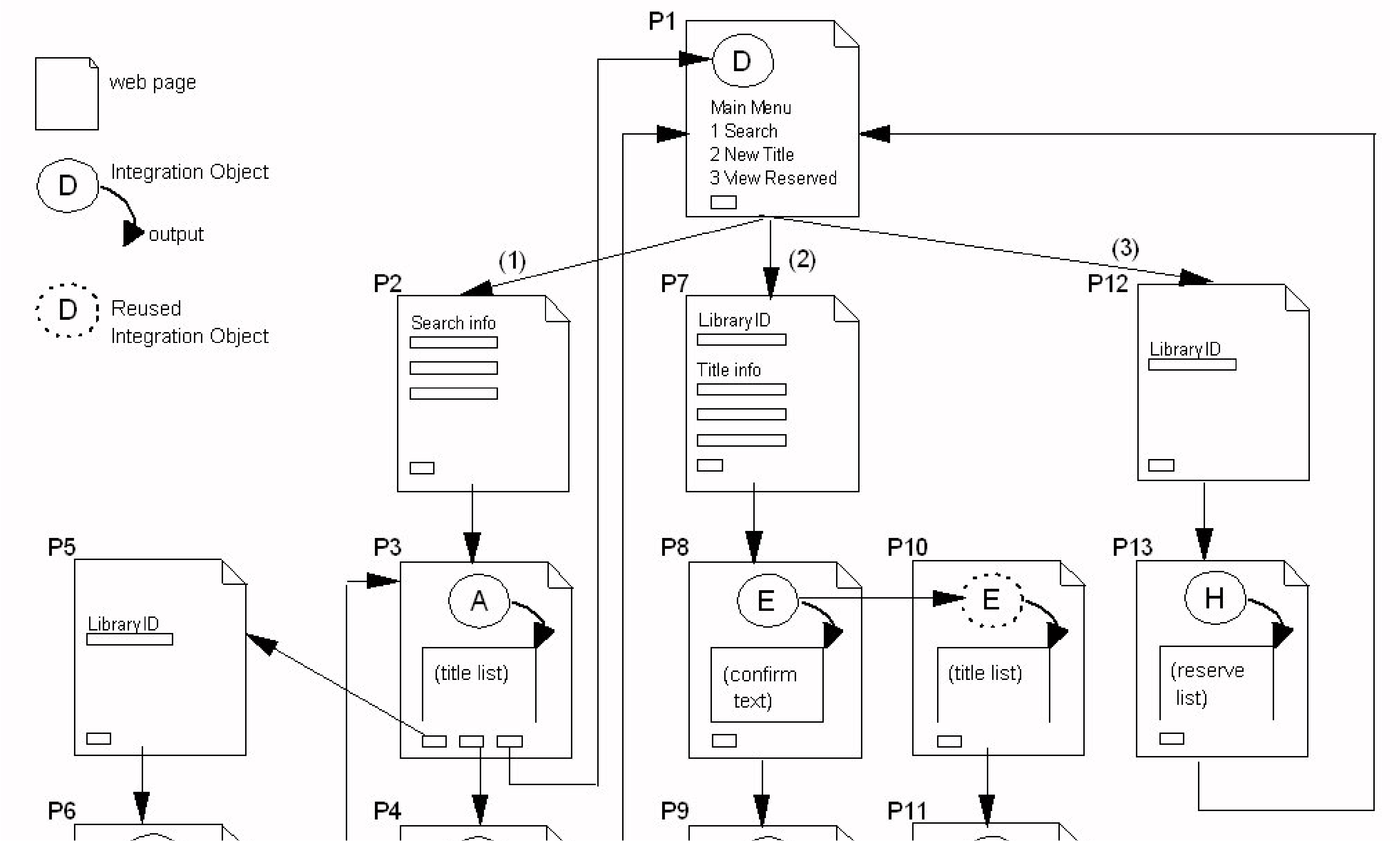
Description: exit the application's main menu and take user back to the ready prompt

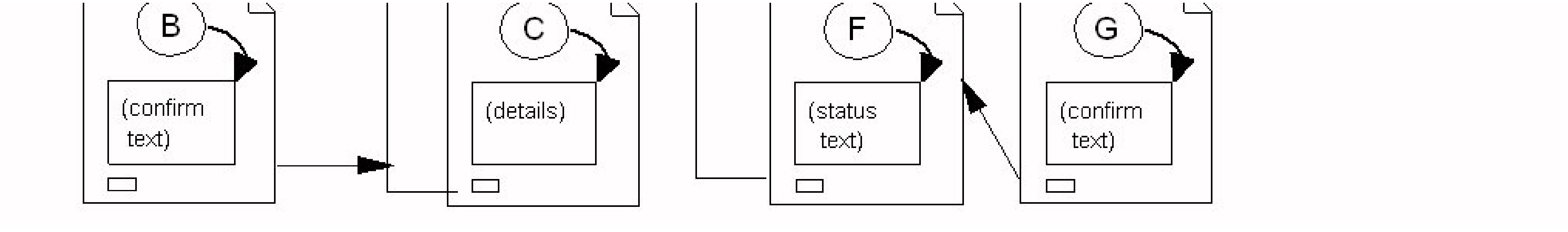
Designing the Web Application

Now that we understand the task flow and the Integration Objects necessary to accomplish the flow, we need a Web application design to drive the Integration Objects.

The design based on two sets of information: the Integration Objects' start and stop states, which govern the flow of Integration Object to Integration Object, and the inputs and outputs to each Integration Object. This should be tempered with any preconception you have as to how the Web application should look.

We will again represent the Web application as a flow chart, with Web pages being the nodes of the chart. Each page containing an Integration Object, or objects, represents them with the letter corresponding to the actual Integration Object designed in the previous section. The pages in the flow chart does not represent the complete layout or content of the page or page names, but instead contains notes and text pertaining to its contents, as well as a page number.





Notes on Web Application Design

Page flow is depicted by arrows pointing from one page to another. Flow is facilitated by HTML FORMs with the ACTION attribute specifying the target page. Integration Object output to the same page is shown as a curved arrow starting at the Integration Object and ending in an output area.

These pages contain the following design points:

P1) This is the first page of the application. When it is run for the first time, Integration Object I is run to enter the application. We know to run Integration Object I only once because the other pages that return to P1 (P3, P9, P13) specify a Web parameter that doesn't exist the first time in.

Integration Object D's responsibility is to return the user from the search list back to the main menu. To reduce the number of pages in the application, we didn't create a page for running Integration Object D because it doesn't have any output to show. So why not conditionally run it if we are returning from page P3? We can do that by detecting if the Return to Main Menu button was clicked on page P3 when P1 is entered, and if so, run Integration Object D to return the host connection back to the application's main menu.

P8 and P10) Integration Object E returns confirmation text, if the title information on page P7 was unique enough, or it returns a list of matching titles. Using Java on P8, we can detect if the confirmation text was returned and render it on P8, or redirect immediately to P10 and render the list of titles. On P10, we reference the instance of Integration Object E from P8 instead of having to recreate the Integration Object and rerun it. We can then access the title list that was returned when the Integration Object was run on P8.

Implementation Details: To facilitate the reference to an Integration Object from a previous page, you must do two things:

1. On the first page (P8), change the value of the SCOPE attribute of the <BEAN> JSP tag used to instantiate the Integration Object on the page, from *request* to *session*. This will cause the instance of the Integration Object to be stored within an HttpSession object which lives past the first page's lifetime.
2. On the next page (P10), create the same <BEAN> tag reference as on the previous page with the SCOPE attribute set to *session* and the CREATE attribute set to *false*. This causes WebSphere to search for a previous instance of the Integration Object (in the HttpSession object) to use instead of creating a new one.

P14) Integration Object J's responsibility is simply to exit the application and return to the ready prompt, at which time the connection is returned to Host Publisher Server. Some type of farewell message can display here. If any other exit points are desired within the Web application, they can all be forwarded to P14, but Integration Object J expects the start state to be LCA Main Menu. Other Integration Objects would have to be provided to exit the application from other states within the host application.

Building the Integration Objects and Web Application

OK, now that you have your design, exercise it on paper first. Come up with a few "use cases" that allow you to follow all the flows of each task, then walk through them using the paper models you've created.

Once you are comfortable that the design is sound, open Host Publisher Studio and use Host Access to create each of the Integration Objects. Be sure to use the macro playback function within Host Access to test the macros.

After the Integration Objects are complete, use Host Publisher Studio to generate the Web pages around the Integration Objects.

Where to Go for More Information

There are numerous resources at your disposal to help you in creating your Web application:

- Host Publisher Administrator's and User's Guide
- Building Integration Objects using IBM Host Publisher (Redbook)
- Samples and examples from Host Publisher's Web site (http://www.ibm.com/software/network/hostpublisher/downloads/)

Trademarks

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.