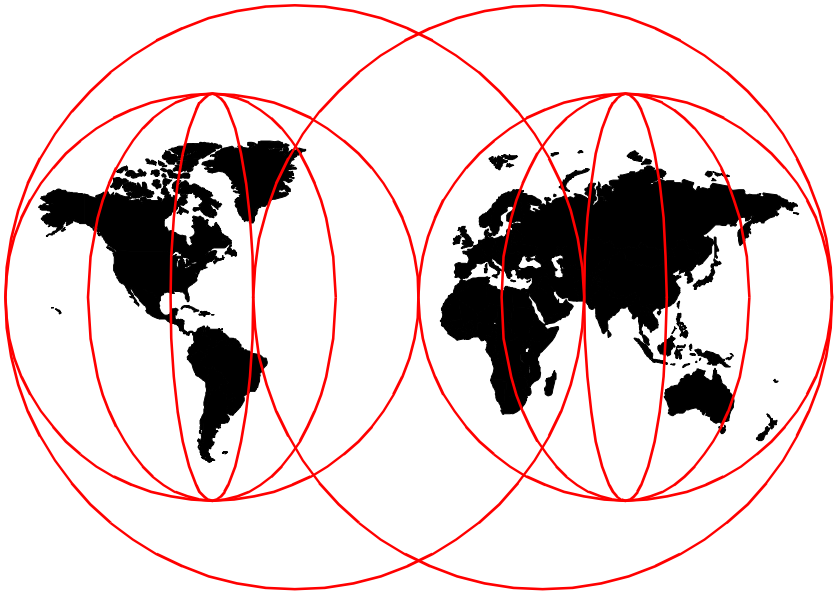
Programming with the Host Access APIs

Jason Bouknight



International Technical Support Organization

www.redbooks.ibm.com

SG24-5856-00

SG24-5856-00

International Technical Support Organization

Programming with the Host Access APIs

December 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix E, “Special notices” on page 139.

First Edition (December 1999)

This edition applies to Host Access Class Library shipped with IBM SecureWay Host On-Demand Version 4.0 and IBM SecureWay Personal Communications Version 4 Release 3.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

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

Contents

Preface

This redbook is designed to help programmers design and implement Web-to-host solutions using Host Access APIs. The material includes basic programming instructions, tips, and examples for Host Access Class Library (HACL) for Java applets and applications, ECLApplets, Host Access Beans for Java, Host Access Controls for ActiveX, and HACL Automation Objects. The basics of programming with each Host Access API discussed in this book are covered including setup information, system requirements, development tools, and examples. Using this information and the appropriate reference material, Java programmers will be able to create programs using the Host Access APIs discussed in this book.

Examples are discussed in the text and are included on the CD-ROM. They may also be downloaded from the Additional Materials section at `http://www.redbooks.ibm.com`.

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, Raleigh Center.

Jason Bouknight is a member of IBM's SWAT Team in the United States. Working in Research Triangle Park, NC, Jason offers assistance to the SecureWay Software Team by helping customers and fellow employees refine e-business solutions with customized programs and answers that supplement such products as Host On-Demand, Screen Customizer, and Personal Communications. Jason is nearing completion of his B.S. in Computer Science at North Carolina State University and may pursue additional certification in Business or Mathematics.

The advisor for this project was:

Carla Sadtler is a Senior Software Engineer at the International Technical Support Organization, Raleigh Center. She writes extensively in many areas including SecureWay Communications Servers, network integration, Web-to-host integration products, and the IBM Network Station. Before joining the ITSO 14 years ago, Carla worked in the Raleigh branch office as a Program Support Representative. She holds a degree in mathematics from the University of North Carolina at Greensboro.

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

Keith Rafferty
IBM Raleigh

Tom Brawn
IBM Raleigh

Scott Quint
IBM Raleigh

Ken Polleck
IBM Raleigh

Brian Web
IBM Raleigh

Greg Knowles
IBM Raleigh

Bill Reid
IBM Raleigh

Chip Mason
IBM Raleigh

Brian Gage
IBM Raleigh

Andy Chow
IBM Raleigh

Comments welcome

Your comments are important to us!

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

- Fax the evaluation form found in “IBM Redbooks evaluation” on page 149 to the fax number shown on the form.
- Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- Send your comments in an internet note to `redbook@us.ibm.com`

Chapter 1. Introduction

IBM's Host Access Class Library (HACL) is an architecture and methodology that allows application programmers to access host applications. The Host Access APIs are derived from HACL to provide the means for a developer to programatically create and interact with those host-based sessions. For example, a developer can connect to a host, copy text from the data stream, and more. Host Access APIs were included with Personal Communications (PCOMM) 4.2 as a way to let Visual Basic programmers interact with the emulator. Host Access APIs were also extended to the developer in Version 2.0 of Host On-Demand. However, IBM's intent was to support a broader range of development platforms for would-be host-based programmers to make HACL more prolific. Personal Communications 4.3 and Host On-Demand 4.0 include new versions of the Host Access APIs to further extend the developer's reach. Table 1 reflects various implementations of the Host Access APIs one would receive given a specific product was purchased.

Table 1. Implementations of the Host Access APIs provided by Host On-Demand 4.0 and Personal Communications 4.3

	Host On-Demand	**Personal Communications**
Java Applications	X	X
Java Applets	X	
ECLApplets	X	X
Host Access Beans for Java	X	X
HACL Automation Objects		X
Host Access Controls for ActiveX	X	X
HACL Lotus Script Extension (LSX)		X
HACL for C++		X

A developer who intends on writing Host Access-based programs should diligently define the screens the program should expect to come across. As you certainly would not want your program to hang because of an unrecognized screen, this includes any error messages the host might send . It only takes a little extra time to build this robustness into your program, but

the reward of consistent user satisfaction will greatly outweigh the cost of implementing these changes.

1.1 How does HACL compare to EHLLAPI?

The Emulator High-Level Language Application Programming Interface (EHLLAPI) was developed by IBM and succeeded its predecessor, HLLAPI, which had been created in 1984 to interface with DOS-based emulators. EHLLAPI became a standardized programming language for interacting with stand-alone emulators, such as Personal Communications. In fact, this is a fundamental difference between HACL and EHLLAPI; HACL can create a host-based session, whereas EHLLAPI must depend on an emulator to supply this connection. HACL also is implemented using an object-oriented hierarchy and object-oriented principles which make it more modern.

EHLLAPI packs all of its functions into a single *dynamically-linked library* (DLL). This single entry-point interface uses four changing parameters to identify specific functions; however, many developers find this to be a very complex interface. HACL inherently solves this problem by dividing up functionality into the individual objects associated with its model.

For a more detailed explanation of the differences between EHLLAPI and HACL, please refer to the section titled "Migrating from EHLLAPI" in *Host Access Class Library*, SC31-8685.

1.2 When and why would you use Host Access APIs?

It has been estimated that roughly 70% of all commercial data is found on AS/400 or S/390 mainframes. Using an emulator is the easiest way to retrieve this information, but IBM realized that for some applications,"green screens" provided by emulators were not the most efficient way to move and interpret data. Why not have the program which creates a session also present and manipulate the data so that the user can understand it better? The majority of operating systems today emphasize a graphical user interface (GUI). Why not let your experience with the host use the same principles?

An organization may find it too costly to replace a "legacy" system with one that many would find more intuitive and functional. Host Access APIs let developers write customized applications which can communicate with these legacy systems because the developer is essentially writing a "front-end". All of the original architecture remains untouched and intact.

Furthermore, most emulators must be installed on a each client's machine. The HACL for Java API supplied with Host On-Demand (HOD) provides the means to centralize your client-to-host solutions much like HOD itself. What if you needed 10,000 employees to sign their electronic time card? Maintenance costs (including installation) are sharply reduced if you deploy a program that runs in a Web browser because you would only need to update the files at the single Web server location.

Whatever the case may be, IBM provides the ability to quickly develop host-based programs in a variety of programming environments especially more popular ones such as Java and Visual Basic. IBM has also bridged the gap between programmers and host systems by building upon such programming standards as Microsoft's COM Automation Objects. We are confident that we can help provide a solution for a developer which will best fit your needs so that you can begin communicating with your host in a whole new way.

1.3 Implementations of the Host Access APIs

We previously mentioned that there are several Host Access APIs which supports a wide variety of programming environments and techniques; however, you will only be able to support particular environments depending on the product you purchased. The flowcharts below should help you decide which implementation of the Host Access APIs is best for you and your organization.

The flowchart in Figure 1 shows the options available when using the HACL supplied with Host On-Demand and when you would choose to use them.

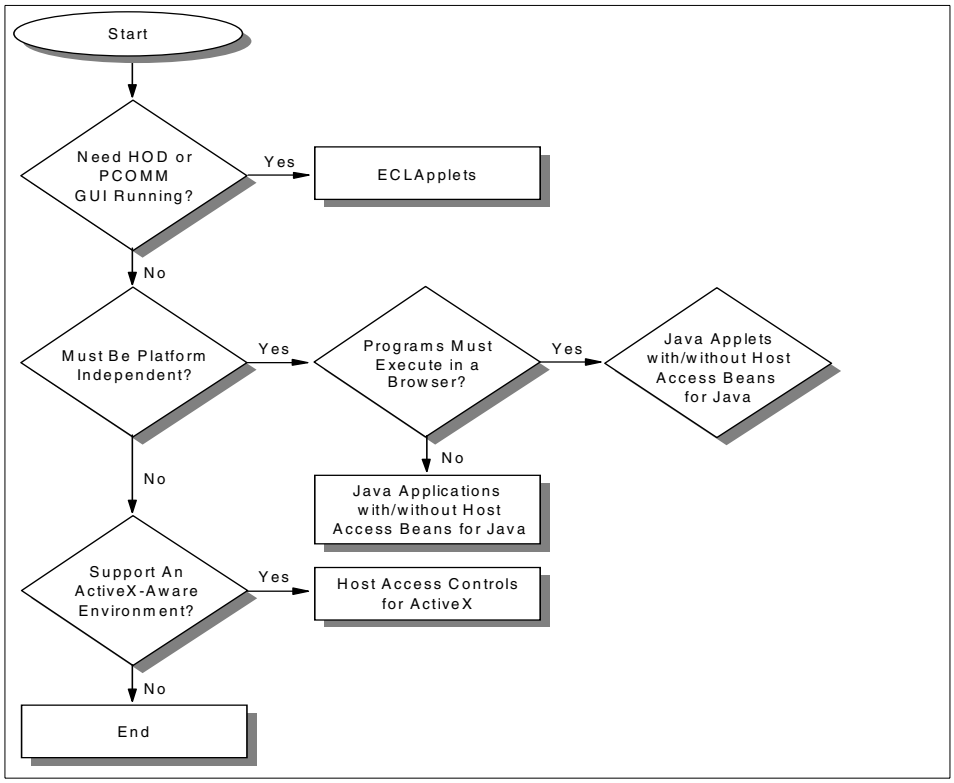


Figure 1. HACL with Host On-Demand

The flowchart in Figure 2 shows the options available when using the HACL supplied with Personal Communications and when you would choose to use them.

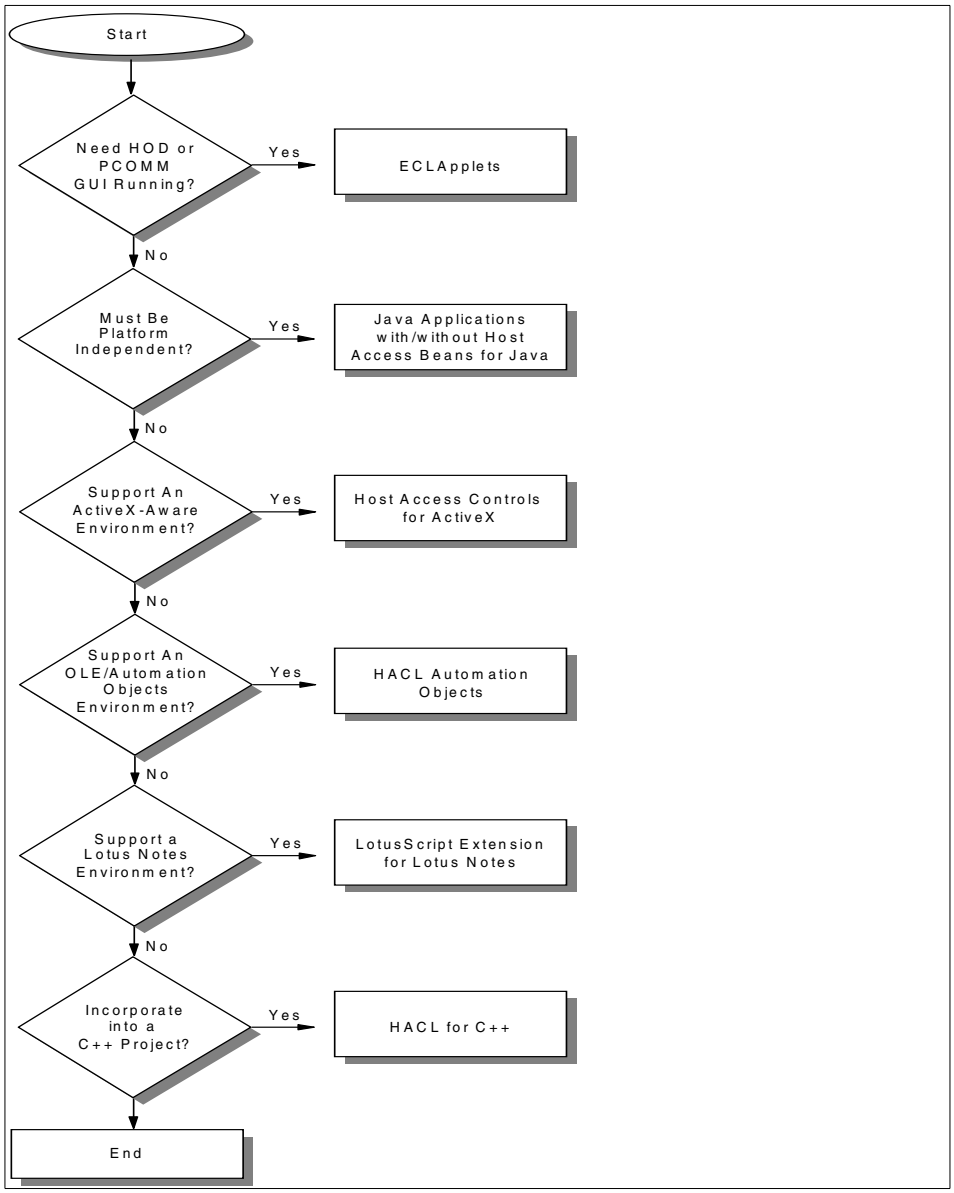


Figure 2. HACL with Personal Communications

We will be discussing the following environments:

- HACL for Java applets and applications
- ECLApplets
- Host Access Beans for Java

- Host Access Controls for ActiveX
- HACL Automation Objects

Environments not discussed in this book are described in *eNetwork Personal Communications V4.3 for Windows 95, Windows 98, and Windows NT: Host Access Class Library*, SC31-8685.

1.4 Host On-Demand vs. Personal Communications

Both Host On-Demand and Personal Communications provide Host Access APIs, but there are some minor differences between the two that are important enough to note.

1) Only the HACL for Java API with Host On-Demand supports creating applets which run in a browser. The only exception to this rule is when applets are created and executed with AppletViewer in unrestricted mode. Unrestricted mode can be set by setting the Network and Class Access fields in the AppletViewer's Properties dialog to Unrestricted. The Allow unsigned applets field must also be marked as Yes. This dialog is located under the File menu.

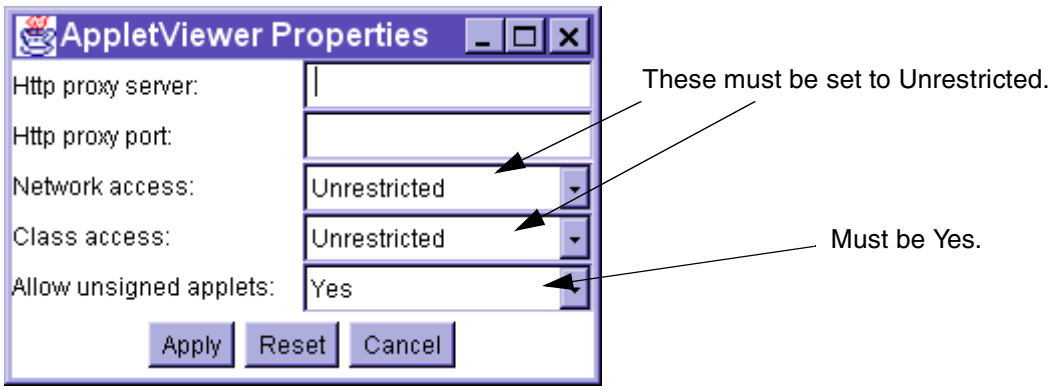


Figure 3. Configuring AppletViewer to execute an applet using HACL for Java

2) The Host Access APIs supplied with Host On-Demand only support TCP/IP.

3) Programs based upon Host Access APIs supplied with Host On-Demand can have their connectivity information configured directly on a per-property basis. Personal Communications-based programs use Personal

Communication config files (.ws) to set connection properties such as hostname and session type.

1.5 IBM's goal for HACL

Today, IBM is working with the Internet Engineering Task Force (IETF) and several companies, such as Attachmate, to create an advanced host API similar to the Host Access Class Library. This new API is called Open Host Interface Objects (OHIO).

In the mid 1980's, HLLAPI became the *de facto* standard for programatically controlling emulators in a DOS environment. IBM is continuing its leadership role in the host-to-client API market by developing OHIO; however, HACL is already emerging as another *de facto* standard for emulator programming.

1.6 Scope of this document

This document is divided into two main parts:

1. Host Access APIs supported by Personal Communications and Host On-Demand

   Four implementations of the Host Access APIs are common to both Host On-Demand 4.0 and Personal Communications 4.3; therefore, the following programming environments will be discussed:

   - Host Access Class Libraries for Java (applets and applications)
   - Host Access Class Libraries for Java (ECLApplets)
   - Host Access Beans for Java
   - Host Access Controls for ActiveX

2. Host Access API for Personal Communications only

   A subset of the Host Access APIs works only with Personal Communications. The second portion of this document will discuss the following:

   - Host Access Class Library Automation Objects for Personal Communications

Several chapters will address five of the most important functions in getting started with host-based programming:

- Connecting to a host/presentation space
- Disconnecting from a host/presentation space
- Searching for text
- Sending text

- Copying the presentation space to a string

Pay close attention to the examples that set these chapters apart from each other. These examples portray important scenarios that should prove each flavor of HACL invaluable for you as an emulator user and host-based programmer.

Note: All referenced source code is available on the included CD-ROM.

1.7 Required tools and programs

The following is a list of programs that you will need to re-create the material within this document.

For Java (applets, applications, ECLApplets) and Host Access Beans for Java:

At least one of the following Java 1.1 Integrated Development Environments (IDE) be must installed to build Java class files:

- Java Development Kit, V1.1.8 by Sun Microsystems (free download at `http://java.sun.com`)
- IBM VisualAge for Java, V2.0 or 3.0, Enterprise Edition
- Symantec Visual Cafe, V3.0
- **Only for Host Access Beans for Java:** Sun Microsystem's BeanBox (free download at `http://www.javasoft.com/beans`)

Java applets created with the HACL for Java API may be run with any browser that supports the Java 1.1 specification.

Additionally, for large deployable applets, it may be necessary to create Java archives for faster downloads and enhanced security. Please install the following kits for each browser you plan on using:

For Internet Explorer:

- Microsoft SDK for Java 3.2 (`http://www.microsoft.com/java/`)

For Netscape Communicator:

- Netscape's Signtool utility (`http://developer.netscape.com/software/signedobj/index.html`)
- Netscape's Capability API classes (`http://developer.netscape.com/docs/manuals/signedobj/capsapi.html`)

For Host Access Controls for ActiveX:

Microsoft has fully implemented ActiveX technology into a variety of its programs including the Win32 operating systems and Microsoft Office. For the purposes of this document, only the Microsoft ActiveX Control Pad will be needed. It can be downloaded from:

```
http://msdn.microsoft.com/workshop/misc/cpad/default.asp
```

Host Access Controls for ActiveX can be used in other IDEs, such as Visual Basic and Visual C++, by inserting them as new components.

For Host Access Class Library Automation Objects (Personal Communications only):

Microsoft Visual Studio also supports automation objects. This document will only discuss how to use Visual Basic 6.0 with HACL Automation Objects, but the same techniques apply when programming with Visual C++, for example. The example uses Microsoft Excel to send data from a worksheet over a network to a host.

1.8 Purpose and assumptions

This publication is intended to explain the basics of the Host Access APIs from the developer's perspective. It will emphasize how to set up each programming environment for use and give realistic, and sometimes *real,* examples of the particular Host Access API at work. Basic functions will be explained as well as the benefits/limitations of working with a particular implementation.

It is assumed that the programmer will have a general knowledge of object-oriented programming and has properly installed the Integrated Development Environment, if it is necessary. Having a basic understanding of the language(s) with which you choose to use the associated Host Access API will be a big help. Doing so will alleviate many of the difficulties associated with using external APIs.

The document assumes that Personal Communications and Host On-Demand were installed to the following directories:

```
C:\Personal Communications\
C:\Hostondemand\
```

Depending on how you installed HOD, the `lib` subdirectory may be `bin` instead; however, we will assume that you will be working with the `lib` subdirectory. Substitute `bin` with `lib` if appropriate.

The examples provided with the document are provided "as-is". **IBM assumes no liability**, indirect or direct, for the use of the programs on any given system.

Chapter 2. HACL for Java - applets and applications

The advent of Web-based intranets and the Internet coupled with Sun's Java programming language opened many new doors to developers. But host-based connectivity had no part in this evolution. It remained a separate entity unto itself.

That changed very quickly.

The Java subset of the Host Access APIs is arguably the most flexible and powerful way to develop host-based applications, or as the case may be, applets. Each class is reliant upon Sun's Java 1.1 specification. The classes provided with Host On-Demand also meet Sun's stringent 100% Pure Java specification, which means the product should function the same regardless of the platform. The Java classes supplied with Personal Communications do not meet this certification. See `http://java.sun.com/100percent/` for more details.

But how would these APIs fit in with Personal Communications or Host On-Demand? Why would a company utilize resources to develop new host-based applications when they just purchased emulator software? Simply put: IBM wants to provide its customers with as many options as possible to simplify the process of connecting and interacting with hosts. But emulators by nature cannot present information like most other graphical user interfaces (GUIs). Customers need to design programs to manage or incorporate information from a host so that the user can understand it better, and do so without modifying the legacy system; hence, there is a need for the Host Access APIs.

After demonstrating five basic functions, we're going to discuss a real-life application of the HACL for Java API in this chapter. The example is an internal IBM applet called TOTALS, a Java applet that provides a front-end to an otherwise "green screen" timesheet system. The applet simplifies the process of entering an employee's weekly hours by providing a GUI for input and submitting the hours without ever showing a green screen.

Note: This document will only discuss the development of applets in detail. The same techniques would apply for creating applications and even servlets. Chapter 3, "HACL for Java - ECLApplets" on page 51 discusses *Run Applets,* which are written in Java, but implement different programming and usability models.

2.1 Benefits and limitations

HACL for Java is a set of classes/objects that developers can use to represent a host-based session. As far as the end user is concerned, all of the session work is done internally - communication with the host is achieved without ever seeing a"green screen".

As long as the Java classes are executed with a compliant Java 1.1 Virtual Machine, there will not be any compatibility issues.

2.2 Establish the environment

To create an applet or application, libraries that the Java compiler is intended to reference must be listed in the classpath. This differs depending on whether you are using the Java libraries shipped with Personal Communications or with HOD.

2.2.1 Host On-Demand

When using the HACL for Java API supplied with HOD, there are two options:

1. Reference habeans.jar in your classpath. For example, enter the following command at a Windows command prompt:

   ```
   set classpath=%classpath%;C:\hostondemand\lib\toolkit\jars\habeans.jar;
   ```

   Note: EBCDIC is the character representation set for mainframes. So that information can be read by individuals, EBCDIC must be translated to unicode, but habeans.jar does not provide those necessary converter classes to translate EBCDIC to unicode. As a result, you must include these classes in subdirectories with respect to where the applet is launched or include them explicitly in the system classpath:

   ```
   set classpath=%classpath%;C:\hostondemand\lib\;
   ```

   If you would rather place the classes in subdirectories, copy the following directories from the Host On-Demand \lib directory so that your applet's directory follows the structure presented in Figure 4:

   - com\ibm\eNetwork\HOD\common
   - com\ibm\eNetwork\HOD\converters

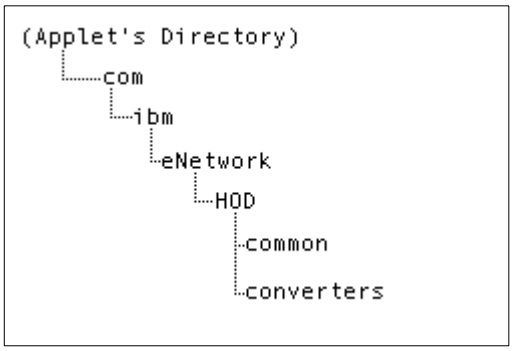


Figure 4. Directory structure for converter classes

2. A second file, habeansnlv.jar, contains the above HACL class files including the required converters. The only trade-off with referencing this file instead of habeans.jar is the size. Habeans.jar is approximately 2.0 megabytes whereas habeansnlv.jar is a larger 5.5 megabytes. The file is also located in the Host On-Demand\lib\toolkit directory:

```
set classpath=%classpath%C:\hostondemand\lib\toolkit\habeansnlv.jar;
```

Note 1: Because habeansnlv.cab is not supplied, Internet Explorer, which uses CAB, not JAR files, will report security violations when the applet is not run locally. Refer to Appendix A, “Signing applets” on page 123 for more information. Java applications do not adhere to this type of "security policy".

Note 2: For the remainder of this document, we will only refer to habeans.jar. Should you need the other converter classes for your applet/application, substitute habeans.jar with habeansnlv.jar.

2.2.2 Personal Communications

Because Personal Communications is a traditional emulator, it was designed to function independent of a Web browser. As a result, Personal Communications supplies a Java Runtime Environment for the development of Java applications with/without Host Access Beans and ECLAppets. Nonetheless, the HACL for Java API package, pcseclj.jar, is referenced in much the same way as Host On-Demand’s files:

```
set classpath=%classpath%;C:\Personal Communications\pcseclj.jar;
```

2.2.3 Loading the HACL for Java API with IBM's VisualAge for Java

IBM's VisualAge for Java is an IDE that provides a graphical user interface (GUI) for developing Java programs. Environments, such as VisualAge for Java, allow the developer to lay out visual Java components on-screen and focus more on the underlying code. VisualAge for Java is available through IBM or any IBM software distributor.

For Host On-Demand:

HACL for Java in conjunction with VisualAge *Enterprise* Edition provides the user two paths to take unlike the other IDEs discussed below. In addition to importing the habeans.jar or pcseclj.jar files, one can use the HOD *Connector* for VisualAge. The HOD Connector is an implementation of HOD which conforms to the Common Connector Framework (CCF) supported by VisualAge. CCF is a common tooling methodology which allows components to interact with the workspace in a consistent manner and vice versa.

The HOD Connector is further broken down into two pieces:

- HOD Common Connector Framework classes
- Host Access Beans for Java classes

The HOD Connector for Visual Age has been geared for developing servlets which facilitate communication with a host system.

VisualAge for Java 3.0 Enterprise Edition ships with the HOD Connector. Additionally, the HOD Connector can be purchased and subsequently downloaded for VisualAge 2.0 from the IBM VisualAge Developers Domain page at `http://www7.software.ibm.com/vad.nsf/`.

You will also need to purchase a license to deploy your applications, even if you received the HOD Connector in the 3.0 Edition of VisualAge. The deployment license is obtained by purchasing Host On-Demand.

Note: If the HOD Connector is installed and a developer imports habeans.jar/pcseclj.jar, VisualAge may report conflicts between the two sets of class files. Uninstall the HOD Connector if you want to import habeans.jar/pcseclj.jar files.

For more information, view \IBM Connectors\Hod\readme.txt under the VisualAge for Java directory. This location may be slightly different if you manually installed the HOD Connector.

For Personal Communications:

Any edition of VisualAge for Java can import the HACL for Java API class files supplied with Personal Communications. Follow these easy steps:

1. Select **File** from the Workbench window.
2. Click **Import.**
3. The SmartGuide window should now appear as shown in the figure below. Ensure that **Jar file** is selected. Click **Next**.

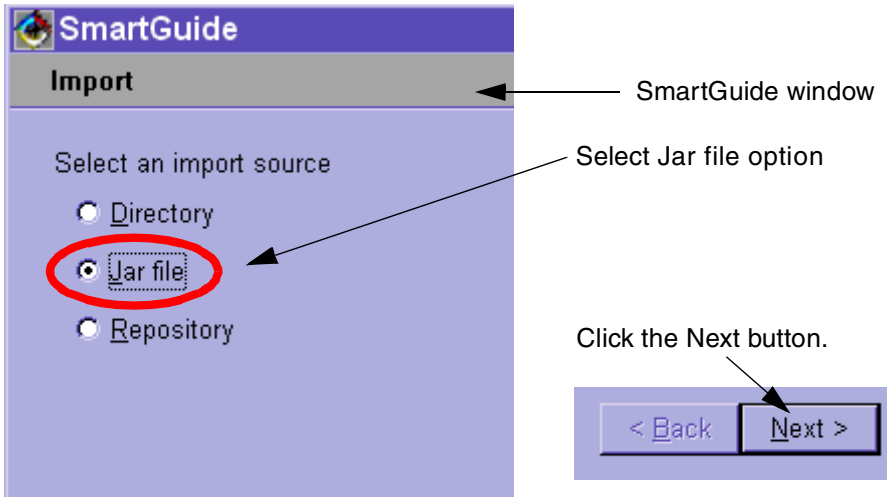


Figure 5. SmartGuide window with Jar file selected in VisualAge for Java

4. The SmartGuide window will now ask which file you wish to import. Click the **Browse** button and find pcseclj.jar located in your Personal Communications directory. Check the **.class** and **resource** checkboxes. Enter a name for a new project or click the **Browse** button to the right of the appropriate text field to select an existing one. Finally, click the **Finish** button to begin the importation process.

Figure 6. VisualAge for Java's progress bar for importing HACL for Java files from Personal Communications

Note: An Available Palette window will appear at the end of the import. This window inserts the Host Access Beans for Java into your project if you so desire. To add the Beans, check each component's checkbox and add a category. Click the **New Category** button and enter an appropriate name such as HACLJavaBeans. JavaBeans are discussed in Chapter 4, "Host Access Beans for Java" on page 63. Click **OK** to finish the process.

2.2.4 Loading HACL for Java with Symantec's Visual Cafe

Symantec's own Java environment follows the same principles used by IBM's VisualAge for Java. The "face" of a program can be laid out through Cafe's graphical user interface. Developers can then write code that manipulates and handles those components - making for a very simple process of developing Java programs.

A developer can point Visual Cafe to a file or directory and these referenced files and/or directories will be appended to the classpath *only* for the current project.

1. Select **Project.**
2. Click **Options.**
3. Select the **Directories** tab.

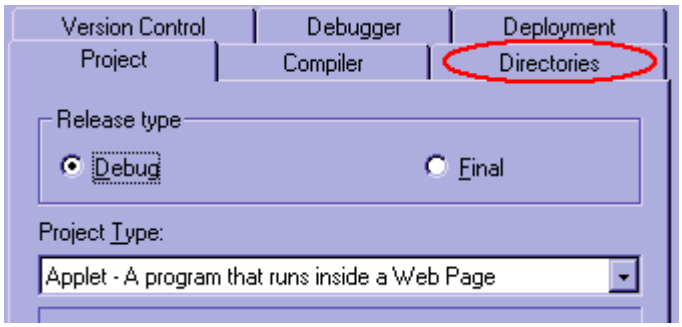


Figure 7. Selecting the Directories tab from Project/Options

4. Click the icon with an arrow pointing to the left.

Figure 8. Selecting the left arrow icon

5. An entry will be inserted into the list. Click **file**.

Figure 9. Selecting the file button to choose which file to reference

6. A file dialog box should now be shown. Find the pcseclj.jar file in your Personal Communications directory or habeans.jar in your Host On-Demand directory. Click **Open**.
7. Visual Cafe will have added the package to the current project.

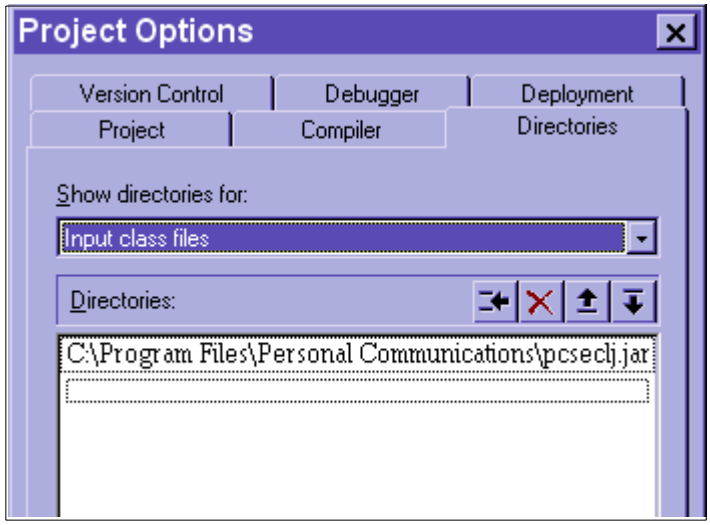


Figure 10. Symantec's Visual Cafe with pcdeclj.jar added to the current project

2.3 Getting started

Note: Due to the fact that applets are much more difficult to set up because they must conform to Web browser requirements, the remainder of this document will focus mainly on creating Java applets. A programmer familiar with Java should be able to make the necessary changes to create an application or servlet.

Creating a HACL for Java program follows the same 3-step process all other Java programs adhere to:

1. Import the necessary class files.
2. Create your objects.
3. Access the properties and methods of those objects.

We will now discuss in detail this 3-step process .

2.3.1 Import the necessary class files

The HACL for Java API provides six packages with which one may import to add functionality to their program. Those packages are as follows:

- package com.ibm.eNetwork.ECL
- package com.ibm.eNetwork.ECL.event
- package com.ibm.eNetwork.ECL.screenreco
- package com.ibm.eNetwork.ECL.screenreco.event
- package com.ibm.eNetwork.ECL.trace
- package com.ibm.eNetwork.HOD.trace

These packages are immediately accessible if the HACL for Java files have been referenced properly. Refer to 2.2, “Establish the environment” on page 12 for more information.

However, the most common functions are located in the first package and can be accessed with a single import statement:

```
import com.ibm.eNetwork.ECL.*;
```

2.3.2 Create HACL for Java Objects

An applet will need objects to represent ECLSession properties, the ECLSession itself, and a presentation space:

```
Properties p = new Properties();
ECLSession s = null;
ECLPS ps = null;
```

The Properties object includes information to define ECLSession parameters. For example, the TCP/IP hostname can be assigned by using the following command in which a valid string is supplied:

```
p.put(ECLSession.SESSION_HOST, "A Valid Hostname") );
```

Once an ECLSession’s properties have been defined, an ECLSession object can be instantiated and subsequently referenced:

```
s = new ECLSession(p);
```

Tip: Your program will have increased flexibility and usability if this string is entered by the user in a TextField instead of hard-coding it like the previous lines showed.

A connection has not been started yet - only defined. Communication can be initiated with the following command:

```
s.StartCommunication();
```

At this point, the ECLSession object should now represent an active session, assuming a valid hostname was supplied.

The final important object to create is the Presentation Space object. A Presentation Space represents a virtual screen containing text and other attributes. The Presentation Space *object* contains the virtual screen plus any functions used to manipulate the information. Simply call the `GetPS()` function from the ECLSession object to return a handle to the current Presentation Space object:

```
ps = s.GetPS();
```

2.3.3 Access the properties and methods of those objects

Now that a connection with the host has been established and the Presentation Space object can be referenced, many functions are available for use. The remainder of this section will explain how to execute four main functions not yet discussed. Additional information concerning the functions available from the ECLSession and Presentation Space objects can be found in the Personal Communications or Host On-Demand Help directories under these names:

- com.ibm.eNetwork.ECL.ECLPS.html
- com.ibm.eNetwork.ECL.ECLSession.html

The files will be located in one of the following directory paths depending on whether HACL was installed with PCOMM or Host On-Demand:

- \PersonalCommunications\lib\en\doc\hacl\
- \Hostondemand\lib\en\doc\hacl\

2.3.3.1 Disconnecting from the host/presentation space

A program must call the `StopCommunication()` function from the ECLSession object to end the connection with the host. For example:

```
s.StopCommunication();
```

2.3.3.2 Searching for text

To touch on a bit of program logic, how will a program know if it has successfully arrived at the correct screen? There are several methods available from the Presentation Space object (ECLPS) to let the user search for a specific string. The most common form is `SearchText()` which takes a

string and search direction as its parameters and does not throw an exception. Of course, the program should search for text that is *unique* for that particular screen, for example, "NEWS":

```
int return_code = ps.SearchString("NEWS", ps.SEARCH_FORWARD );
```

Note about the return code: If the text is found within the Presentation Space, the linear position of where the string begins will be returned. If the text was not found, the return code equals 0.

We can also use the `WaitForString(String s)` function to wait for a particular string of text to appear inside the presentation space:

```
boolean wasFound = ps.WaitForString("NEWS");
```

Refer to 2.4.1, “Screen recognition” on page 21 for a different way to navigate through host screens using Screen Recognition technology.

2.3.3.3 Sending text

To navigate between screens, a program must be able to send text to the host. Sending text is just as simple as searching for it. Again, the Presentation Space object contains a variety of methods used to accomplish this task. One such method is `SendKeys()` which only needs a string as its parameter:

```
try {
     ps.SendKeys("Hello World");
} catch (ECLErr e) {
      e.printStackTrace();
}
```

The `SendKeys()` function also supports mnemonics such as "[enter]" to mimic a user pushing the Enter key. See Appendix B, “SendKey() mnemonics” on page 127 for a full list of supported key mnemonics.

Using the above `SendKeys()` function places the string of text at the current cursor location. The are additional `SendKeys()` definitions that will place your text in more specific locations. See the documentation for a full list.

Tip: Before you try to send text to the presentation space, use `IsCommReady()` from the ECLSession object to determine if the program may send keys to the host. Calling such functions as `SendKeys()` when `IsCommReady()` returns false will result in your or the program's keystrokes being ignored. Add a simple *while* loop to your program to "wait" until the appropriate time to use the ECLSession object:

```
int counter = 0;
while ((!s.IsCommReady() ) && (counter < 5000) )
{
  try {
    Thread.sleep(50);
    counter++;
  } catch (Exception e) { }
}
```

2.3.3.4 Copying the presentation space to a string

How would we copy all or some of the text in a presentation space so that it can be presented to the user? It is possible to copy the entire contents of the presentation space to a string. Usually, the presentation space object's (ECLPS) `GetString()` method is called with a character array and the array's length as an integer for parameters.

Note: The `GetString()` method does not actually return a string - the text is actually placed within a character array; however, this array is easily converted to a string by passing it into a string's constructor:

```
char[] buff = new char[1920];
try {
    ps.GetString(buff, buff.length );
        } catch (ECLErr e) {
 e.printStackTrace();
        }
String psString = new String(buff)
```

2.4 Beyond the basics

HACL for Java, or any of the other Host Access APIs, consist of more than just five functions. There is an incredible amount of functionality built into the available objects. Chapter 3, "HACL for Java - ECLApplets" on page 51 discusses the ECLApplet feature of the HACL for Java API. One feature that should be discussed here is screen recognition.

2.4.1 Screen recognition

The HACL for Java API includes a much improved method of determining if a particular screen has been reached: screen recognition technology. This patented technology fires events when a screen is loaded into the presentation space and that screen meets a predefined set of criteria. The amount of code and time one normally spends to write a Host Access-based program can be significantly reduced using this technology.

The ability to monitor and respond to incoming screens requires three additional objects:

1. **Screen Description object (ECLScreenDesc)**: Properties within this object can be set so that a unique "picture" of a screen can be generated. The properties which can be monitored for are:
   - Any attributes at any position on any plane
   - A specific cursor position
   - Total number of fields or the number of input fields
   - A string of text at any position or in any screen rectangle (case sensitive or not)
   - Wait for operator information area (OIA) inhibit status
2. **Callback Class (implements ECLRecoNotify):** An action must be performed when an appropriate screen has been recognized. As a result, a "callback class" must be defined for this purpose. Callback classes must implement ECLRecoNotify which means definitions must be provided for the following three functions:
   - public void NotifyEvent(ECLPS ps, ECLScreenDesc sd)
   - public void NotifyError(ECLPS ps, ECLScreenDesc, ECLErr e)
   - public void NotifyStop(ECLScreen sd, int Reason)
3. **Screen Recognition object (ECLScreenReco):** A Screen Recognition object takes a Screen Description object and executes its associated callback class.

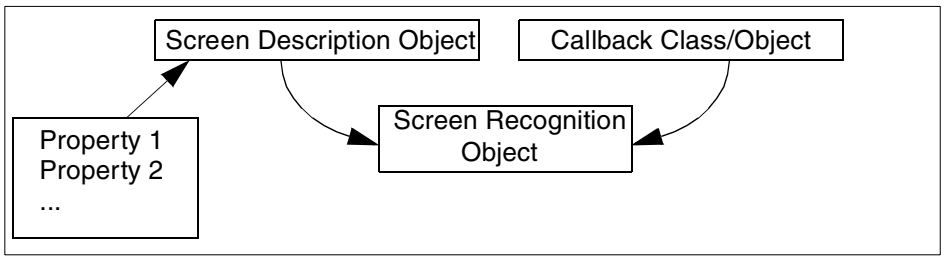


Figure 11. Diagram of three objects required for screen recognition

The following code demonstrates how to add Screen Recognition technology to your HACL for Java applet. The first piece of code creates all of the

necessary objects and, for demonstration purposes, adds a few properties to the Screen Description class.

```
ECLScreenDesc myScreenDesc = new ECLLScreenDesc();                    A
myScreenDesc.AddCursorPos(20, 16);
myScreenDesc.AddString("WELCOME TO");
myScreenDesc.AddOIAInhibitStatus(ECLScreenDesc.DONTCARE);

SimpleRecoCallback myCallback = new SimpleRecoCallback();             B

ECLScreenReco myScreenReco = new ECLScreenReco();                     C
myScreenReco.AddPS(ps);
myScreenReco.RegisterScreen(myScreenDesc, myCallback);
```

Figure 12. First piece of code needed to add screen recognition technology

A) A Screen Description object named `myScreenDesc` is created and three properties to define the screen are added. A "match" will be made if a screen sets the cursor to position 20,16 and the string "WELCOME TO" appears anywhere in the screen. Additionally, the OIA Inhibit Status should not make a difference in defining the screen for this example.

B) This line of code creates the callback object. `myCallback` is not a standard HACL object, rather it implements ECLRecoNotify. A sample SimpleRecoCallback class will be provided in just a moment.

C) At this point, a Screen Recognition object can be instantiated. It is given a reference to the current presentation space for monitoring purposes with the `AddPS()` function. Finally, the screen defined in section A is added to the Screen Recognition object by the `RegisterScreen()` function. The Screen Description object and the callback object are the only parameters.

There is still the matter of defining the callback class. Without a callback class, there would be no way to determine what action(s) to perform once a screen has been recognized. The code below is one example of a callback class.

```
import com.ibm.eNetwork.ECL.*                                          A
class SimpleRecoCallback implements ECLRecoNotify
{
   public void NotifyEvent(ECLPS ps, ECLScreenDesc sd) { }
   public void NotifyError(ECLPS ps, ECLScreenDesc sd, ECLErr e) { }   B
   public void NotifyStop(ECLScreenDesc sd, int Reason) { }
}
```

Figure 13. Second piece of code needed to add screen recognition technology

A) As with any class, import the appropriate package. For the Screen Recognition object to correctly execute a callback class, the class must implement ECLRecoNotify to ensure the proper functions are defined.

B) The functions defined by ECLRecoNotify are given method bodies.

Of course, your program may have several different callback classes for different screens. Your program may also use only one callback class with logic inside to handle several different callback screens.

2.5 Deployment and testing

Writing a HACL-based Java program is only *half* the job. The following sections are intended to help your Java applet execute smoothly.

2.5.1 Reference libraries in HTML

Jar and CAB files are two standards which exist for archiving Java applets. Netscape only recognizes the JAR format. Internet Explorer, uses JAR and CAB files, but only recognizes digital signatures in CAB files. This "archiving duality" must be handled in the applet tag that identifies the Java applet. For example:

```
<Applet Archive="habeans.jar" Code="JavaSample.class"
   Width=500 Height=300>
<param name="cabinets" value="habeans.cab">
</Applet>
```

The first line alerts the browser that an applet is about to be defined. It also tells the browser to load habeans.jar in addition to executing JavaSample.class. Height and width parameters define the area a browser should assign within the Web page for the applet.

The second line of the applet tag is specifically for Internet Explorer. The browser will ignore the Archive statement in the tag and load any CAB files referenced by the Cabinets parameter.

2.5.2 Security and signing

Each browser defines what is and what is not acceptable default behavior for an applet. An applet which attempts to extend its access, such as connecting to a host other than where it was downloaded from, must request permission from the user. If your applet begins to throw SecurityExceptions in the Java console, then your program needs to ask for these permissions. See Appendix A, “Signing applets” on page 123 for more information on how to correct this problem.

2.5.3 Browser compatibility

While the HACL for Java API meets Sun’s Java 1.1 specification, not all browsers execute the associated functions properly. Make sure your browser is Java 1.1 compliant and your applet will execute as expected.

2.6 Example 1: JavaSample applet

This example illustrates a simple HACL for Java applet using the HACL for Java API supplied with Host On-Demand 4.0. The applet will present a graphical presentation to the user, providing choices of actions and input. A terminal screen will also be provided to the user by using the terminal bean. The user will have the following choices:

- Connect to a host.
- Disconnect from the host.
- Search for text in the screen.
- Send text.
- Copy the presentation space to a string.

2.6.1 Step 1: Set up the environment

Following the instructions in 2.2.1, “Host On-Demand” on page 12, set up the environment:

- Set classpath to contain hostondemand\hod\lib\toolkit\jars\habeans.jar
- Make the converter classes available by either:
  - Using habeansnlv.jar instead of habeans.jar, or

- By copying the converter classes from the HOD libraries to subdirectories under the directory the applet will be launched from, preserving the correct directory structure.

  c:\hostondemand\lib\com\ibm\enetwork\hod\common*.* ->
  c:\java_source\hacl\com\ibm\eNetwork\Hod\common*.*

  c:\hostondemand\lib\com\ibm\enetwork\hod\converters*.* ->
  c:\java_source\hacl\com\ibm\eNetwork\Hod\converters*.*

2.6.2 Step 2: Create and compile the source code

This sample applet has two Java source files:

- JavaSample.java
- SimpleRecoCallback.java

The source for each will be listed later.

Compile the source:

```
javac JavaSample.java SimpleRecoCallback.java
```

2.6.3 Step 4: Create and sign the JAR file for Netscape

This example will be run using Netscape, so we need to make a JAR file. Following the instructions in A.2, “How to sign JAR files for Netscape Navigator” on page 123, create a JAR file and sign the contents. If you will be using Microsoft Internet Explorer, follow the directions in A.1, “How to sign CAB files for Internet Explorer” on page 123 to create the CAB files.

For this example, we will create a test certificate to sign with, called mycert.

1. The Sign tool should be downloaded. We put ours in c:\java_signtool:

   ```
   http://developer.netscape.com/software/signedobj/jarpack.html#signtool1.1
   ```

2. Use the security icon in Netscape to assign a password (we called ours password).
3. Create a certificate called mycert. **Note:** this certificate is for testing and will only work on the PC it is created on. Normally, you would buy a certificate.

```
c:\java_signtool>signtool -G mycert -d "c:\Program Files\Netscape\Users\default"
using certificate directory: c:\Program Files\Netscape\Users\default

WARNING: Performing this operation while Communicator is running could cause
corruption of your security databases. If Communicator is currently running,
you should exit Communicator before continuing this operation. Enter
"y" to continue, or anything else to abort: y


Enter certificate information.  All fields are optional. Acceptable
characters are numbers, letters, spaces, and apostrophes.
certificate common name: mycert
organization: IBM
organization unit:
state or province:
country (must be exactly 2 characters): US
username: Carla Sadtler
email address:
Enter Password or Pin for "Communicator Certificate DB": password
generated public/private key pair
certificate request generated
certificate has been signed
certificate "mycert" added to database
Exported certificate to x509.raw and x509.cacert.
```

4. Create the JAR file:

   - Copy the class files to a unique directory (no other class files and no subdirectories with class files). The sign tool looks for all class files in that directory and any subdirectories when building the JAR file. Having the class files in a unique directory keeps you from getting unneeded class files in the JAR file. We copied the two class files (JavaSample.class and SimpleRecoCallback.java) into a directory called c:\java_source\hacl\sign.
   - Use the `signtool` command to create the JAR file.

```
c:\java_signtool>signtool -k mycert -d "c:\Program Files\Netscape\Users\default"
 -Z "JavaSample.jar" c:\java_source\hacl\sign
using certificate directory: c:\Program Files\Netscape\Users\default
Generating d:\java_source\hacl\sign/META-INF/manifest.mf file..
--> JavaSample.class
adding c:\java_source\hacl\sign/JavaSample.class to JavaSample.jar...(deflated 4
8%)
--> SimpleRecoCallback.class
adding c:\java_source\hacl\sign/SimpleRecoCallback.class to JavaSample.jar...(de
flated 50%)
Generating zigbert.sf file..
Enter Password or Pin for "Communicator Certificate DB":
adding c:\java_source\hacl\sign/META-INF/manifest.mf to JavaSample.jar...(deflat
ed 29%)
adding c:\java_source\hacl\sign/META-INF/zigbert.sf to JavaSample.jar...(deflate
d 35%)
adding c:\java_source\hacl\sign/META-INF/zigbert.rsa to JavaSample.jar...(deflat
ed 31%)
tree "c:\java_source\hacl\sign" signed successfully
```

5. Copy the JAR file from c:\java_signtool\JavaSample.jar to the directory you want to invoke the applet from (d:\java_source\hacl\JavaSample.jar).
6. Remove the class files from this directory so Netscape will execute the class files in the JAR file, not the original class files.
7. Create an html file (the CAB file is used for Internet Explorer, the JAR file for Netscape). The example below uses habeansnlv.jar instead of habeans.jar and assumes you created a corresponding CAB file (discussed in Appendix A, “Signing applets” on page 123).

```
<Applet Archive="JavaSample.jar, habeansnlv.jar"
Code="JavaSample.class" Width=500 Height=500>
<param name="cabinets" value="JavaSample.cab, habeansnlv.cab">
</Applet>
```

Figure 14. JavaSample.html

8. Call the applet from a Netscape Web browser:

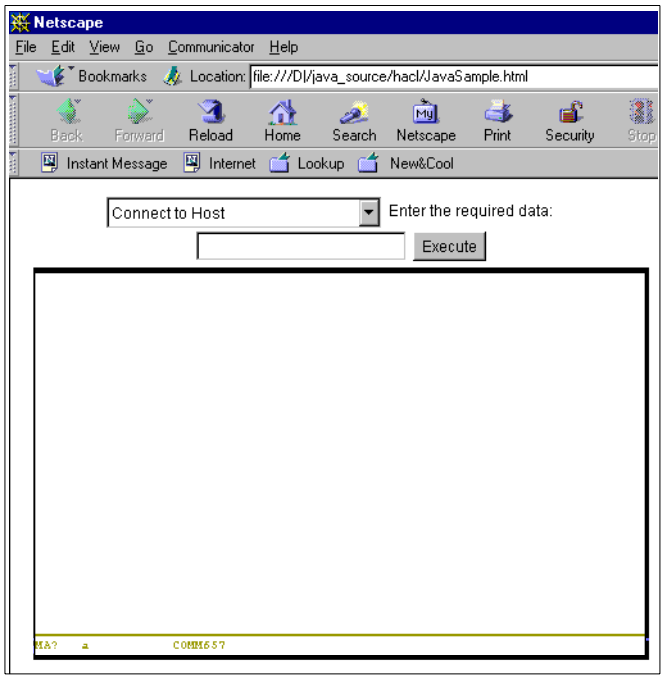


Figure 15. JavaSample: initial interface

When the user enters the URL of the JavaSample applet, a screen is built containing a choice field, a text input field, and a terminal screen (using the Terminal bean). The code that does this can be seen in Figure 20 on page 34, starting at marker **A**.

From this screen, the user can choose an action and provide input if necessary.

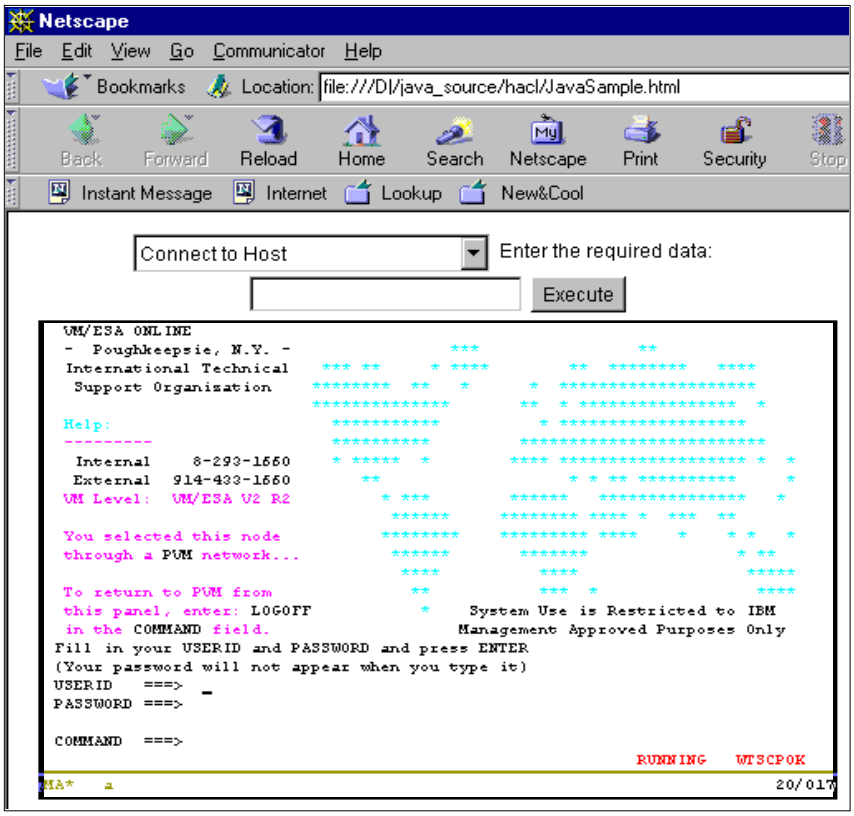


Figure 16. Connecting to the host

The first logical action would be to connect to a host. The text input must be the IP address or hostname of the host to connect to since were are using the HACL for Java API supplied with Host On-Demand in this example. Once the Connect to Host choice is selected, the IP address/hostname is entered, and Execute is clicked, the applet will establish a connection with the host.

The code that does this can be seen in Figure 21 on page 35, starting at marker **B**.

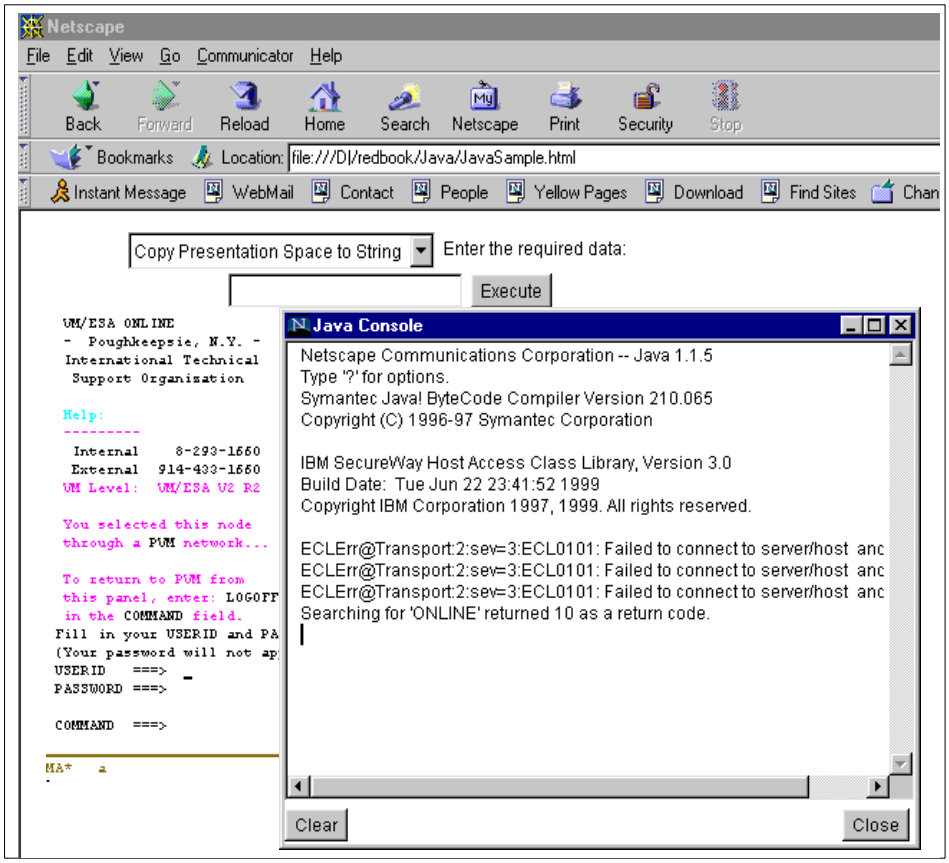


Figure 17. Searching for text

To search for text in the presentation space, choose the **Search for Text** action, enter the desired text in the input field, and click **Execute**. Open the Java console to see the results. The linear location of the text will be returned.

The code that does this can be seen in Figure 22 on page 36, starting at marker **D**.

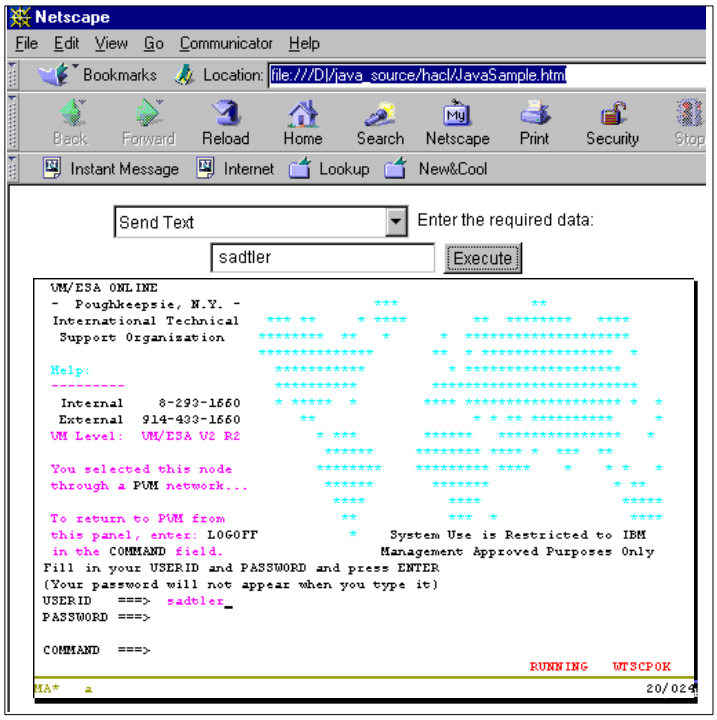


Figure 18. Sending text

To send text to the presentation space, choose the **Send Text** action, enter the desired text in the input field, and click **Execute**. The text will be placed in the presentation space at the cursor location. In this example, the text is only placed in the presentation space and not sent to the host. To take this one step further you could use the combination of text and a SendKey mnemonic (listed in Appendix B, “SendKey() mnemonics” on page 127).

The code that does this can be seen in Figure 22 on page 36, starting at marker **E**.

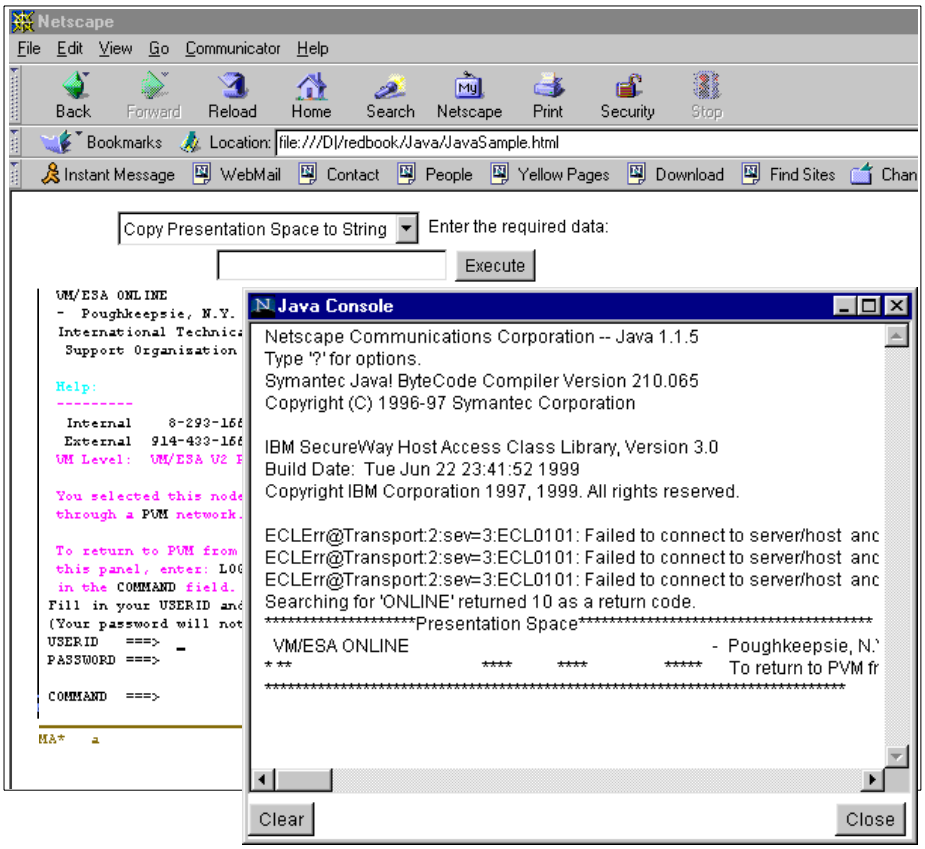


Figure 19. Copy the presentation space to a string

To copy the presentation space to a string, choose the **Copy Presentation Space to String** action and click **Execute**. Open the Java console to see the results.

The code that does this can be seen in Figure 22 on page 36, starting at marker **F**.

2.6.4 Source Code for JavaSample.java

```
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import com.ibm.eNetwork.ECL.*;
import com.ms.security.*;

// For Terminal Only - Used only to display HACL at work
import com.ibm.eNetwork.beans.HOD.Terminal;

public class JavaSample extends java.applet.Applet implements ActionListener
{
    // Required components for host-based communication
    Terminal term;
    ECLSession s = null;
    ECLPS ps = null;
    Properties p = new Properties();

    // Components only needed for Graphical User Interface
    Choice commandChooser = new Choice();
    Label message = new Label("Enter the required data: ");
    Button executeButton = new Button("Execute");
    TextField info = new TextField(20);

    public void init()                                                    (A)
    {
        executeButton.addActionListener(this);
        commandChooser.add("Connect to Host");
        commandChooser.addItem("Disconnect from Host");
        commandChooser.addItem("Search for Text");
        commandChooser.addItem("Send Text");
        commandChooser.addItem("Copy Presentation Space to String");

        //Create the Terminal for display
        try {
        term = new Terminal(p);
        } catch (Exception e) {
        e.printStackTrace();
        }
```

Figure 20. JavaSample.java

```
        // Adding Components for Graphical User Interface (GUI)
        add(commandChooser);
        add(message);
        add(info);
        add(executeButton);
        add(term);

    }

    public void connect()                                                    (B)
    {
        //For use with the Terminal only
    try {
        term.setHost(info.getText() );
        term.startCommunication();
        } catch (Exception e) {
        }
        s = term.getECLSession();


        // Get the Presentation Space Object
        ps = s.GetPS();

        // FOR SCREEN RECOGNITION ONLY
        ECLScreenDesc myScreenDesc = new ECLScreenDesc();
        myScreenDesc.AddCursorPos(20, 16);
        myScreenDesc.AddString("WELCOME TO");
        myScreenDesc.AddOIAInhibitStatus(ECLScreenDesc.DONTCARE);

        SimpleRecoCallback myCallback = new SimpleRecoCallback();

        ECLScreenReco myScreenReco = new ECLScreenReco();
        myScreenReco.AddPS(ps);
        myScreenReco.RegisterScreen(myScreenDesc, myCallback);

    }
```

Figure 21. JavaSample.java

```
    public void disconnect()                                                    C
    {
        // Next line for the Terminal only
        term.stopCommunication();

        //s.StopCommunication();
    }

    public void search()                                                        D
    {
        try {
          int return_code = ps.SearchString(info.getText(), ps.SEARCH_FORWARD);
          System.out.println("Searching for '" + info.getText() + "' returned " +
return_code + " as a return code.");
    } catch (Exception e) { }

    }

    public void send()                                                          E
    {
        try {
            ps.SendKeys(info.getText() );
        } catch (ECLErr e) { e.printStackTrace();
        }
    }

    public void copy()                                                          F
    {
        char[] buff = new char[1920];
        try {
            ps.GetString(buff, buff.length );
        } catch (ECLErr e) { e.printStackTrace();
        }
          System.out.println("****************Presentation Space*****************");
          System.out.println(buff);
          System.out.println("***************************************************");
    }
```

Figure 22. JavaSample.java

```
public void actionPerformed(ActionEvent e)                                    G
        // Handle ActionEvents according to the Java 1.1 model.
        // Only only for purposes of the Graphical User Interface
    {
        Object anchorpoint = getParent();
        Object src = e.getSource();

        if (src == executeButton)
        {
            if (commandChooser.getSelectedItem() == "Connect to Host")
            {
                connect();
            }
            else if (commandChooser.getSelectedItem() == "Disconnect from Host")
            {
                disconnect();
            }
            else if (commandChooser.getSelectedItem() == "Search for Text")
            {
                search();
            }
            else if (commandChooser.getSelectedItem() == "Send Text")
            {
                send();
            }
    else if (commandChooser.getSelectedItem() == "Copy Presentation Space to String")
            {
                copy();
            }
        }
    }

}
```

Figure 23. JavaSample.java

2.6.5 Source code for SimpleRecoCallback.java

```
import com.ibm.eNetwork.ECL.*;

class SimpleRecoCallback implements ECLRecoNotify
{
        public void NotifyEvent(ECLPS ps, ECLScreenDesc sd)
        {
            // Code here for the specific screen
            //System.out.println("The login screen was found!!!!");
        }

        public void NotifyError(ECLPS ps, ECLScreenDesc sd, ECLErr e)
        {
           // Error Handling
           //System.out.println("Error Handled");
        }

        public void NotifyStop(ECLScreenDesc sd, int Reason)
        {
           // possible stop montioring
           //System.out.println("Stop called");
        }
}
```

Figure 24. SimpleRecoCallback.java

2.7 Example 2: TOTALS applet

The following example is a not a hypothetical situation. It is an actual program used by IBM for non-salary employees to interact with the timesheet system. This example is a little more advanced and is for experienced Java programmers. Parts of the code are shown here. The entire applet is included in the CD-ROM that comes with this redbook. The TOTALS applet is property of IBM and may only be used for purposes of education.

Terminal Oriented Time and Labor System (TOTALS) is IBM's answer to electronically managing time worked on a weekly basis. A user can access TOTALS through a 3270-style VM system and navigate through the "green screens". In the summer of 1999, IBM management decided to demonstrate the power behind the HACL for Java API by building a front-end program for the TOTALS system. This program was intended to simplify the process of

filling out a timesheet by automating each step. The result: IBM's TOTALS applet.

In the pages to come, you will understand three things:

1. How TOTALS was originally used with an emulator
2. How the TOTALS applet operates, internally and from the user's perspective
3. The benefits realized by using the HACL for Java API in this scenario and how they are easily transferable to other situations

2.7.1 Old method: using the TOTALS host program with an emulator

In order to show the contrast between using host green screens and the newer user friendly applet described in this example, we will take you through the process an employee would use to submit a timecard using the original VM program. An employee must first connect to a host, log in, and execute the TOTALS program. These tasks alone are repetitive in nature except for the unique user name, password, and weekly information. The first three steps explain how one would achieve this using the original method.

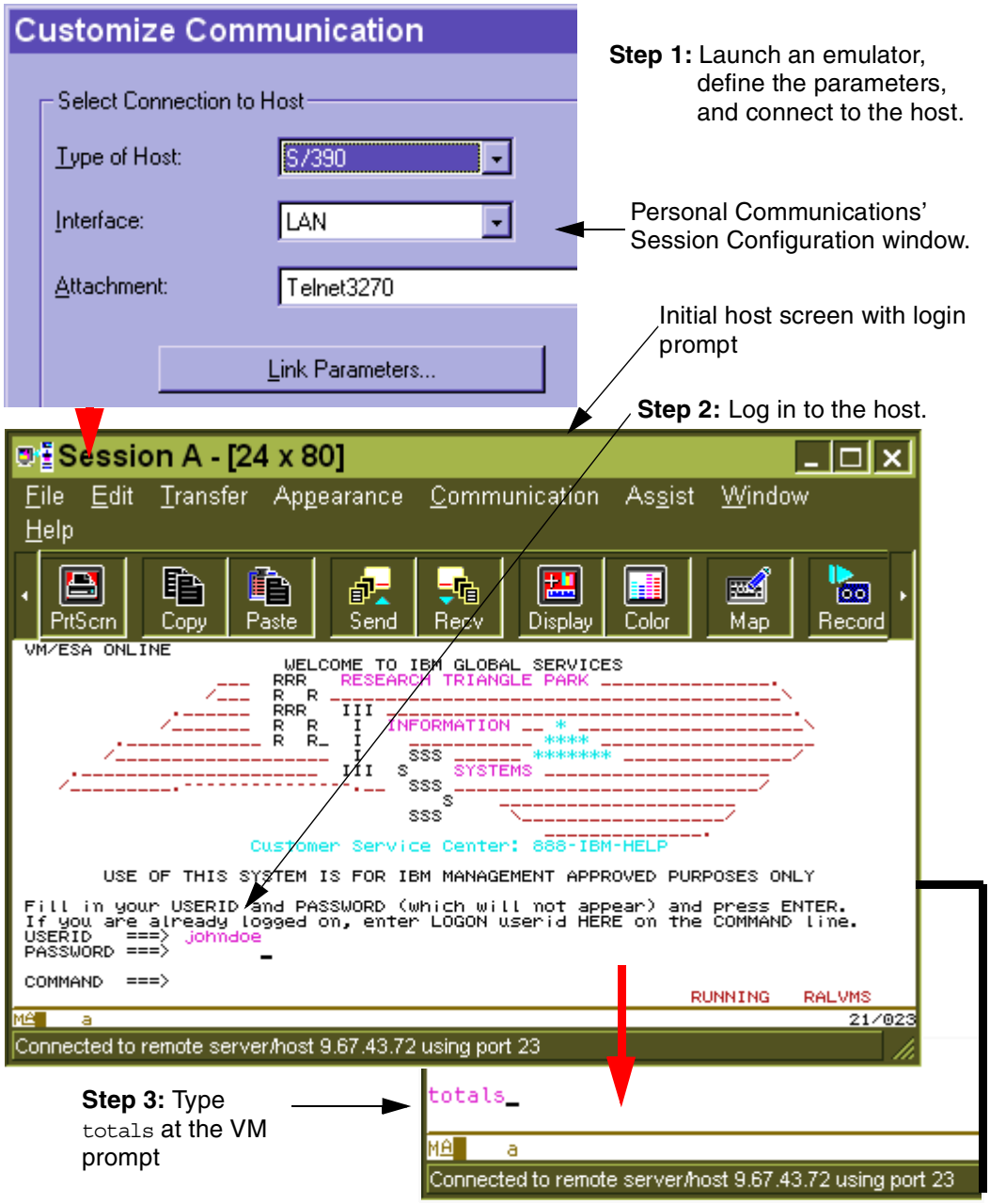
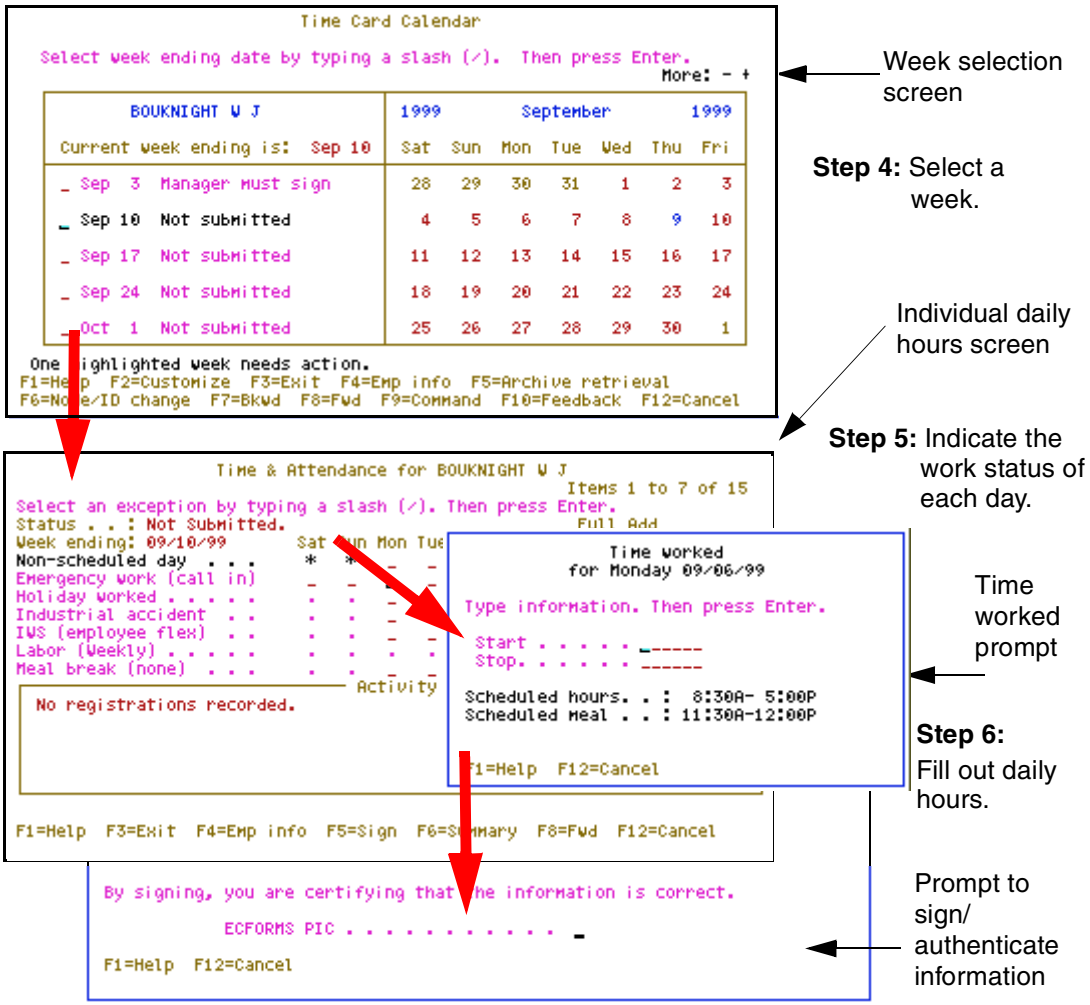


Figure 25. Executing TOTALS with the Personal Communications emulator

After navigating past any information screens, the Week Selection screen appears. The user should select the current week to begin entering hours for each day. Having completed each day, an employee must "sign" or authenticate the timecard with a Personal Identification Code (PIC). The figure below displays each screen a user sees during these next four steps.



Step 7: Enter Individual's PIC to Sign the Timesheet.

Figure 26. Entering information into TOTALS and "signing" the timesheet

Of course, the final steps include exiting from TOTALS and disconnecting from the host, which we will not show in detail here.

2.7.2 New method: the TOTALS applet

Now let's take a look at the TOTALS applet. The applet simplifies the steps for the user and presents the information in a user friendly GUI. First, there is no "green screen". The user opens a Web browser and enters the URL for the applet.

2.7.2.1 Connecting to the host

Much of the work explained in the previous section is automatically executed with the TOTALS applet. In fact, after the user profile and timesheet are completed, the applet will have all the information it needs to start communicating with a host.

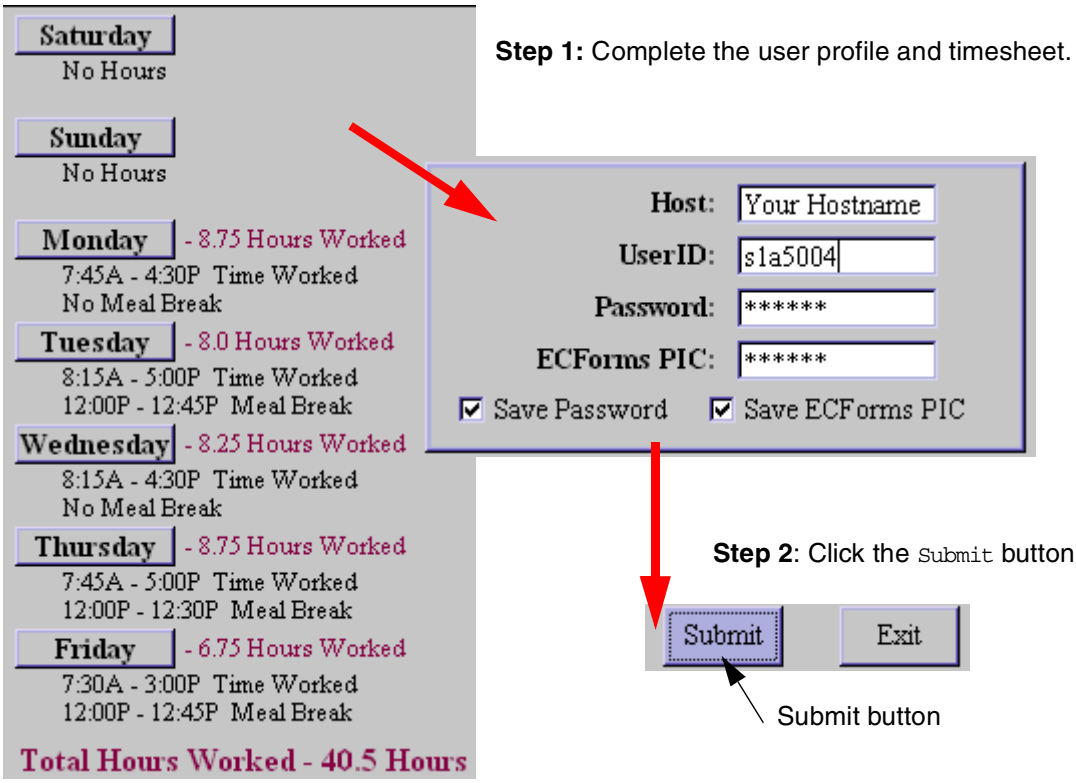


Figure 27. TOTALS applet user profile and Submit button

What happens when the Submit button is clicked? Visually, the following status message appears to alert the user that the applet is attempting to connect to the specified host.

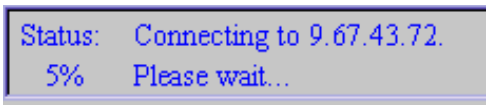


Figure 28. Status message displayed while the TOTALS applet initiates communication

But what about internally? The code provided below is a snippet of the TOTALS applet's code which begins communications with a host. The highlighted lines of code are explained in more detail.

```
try {
    Properties PR = new Properties();                          A
    PR.put(ECLSession.SESSION_HOST, new String(UI.Host));
    try {
        s = new ECLSession(PR);                                B
        s.StartCommunication();
        setVisible(true);
        requestFocus();
        ps = s.GetPS();                                        C
    } catch (ECLErr e) {                                       D
        System.out.println(e.GetMsgText());
```

Figure 29. TOTALS applet code that initiates communication with the defined host

A) A Properties object is created and the host name is set from a string that is a member of an internal profile object created by the TOTALS applet. The host name could also be hard-coded for a less flexible means of setting the parameter.

B) The Properties object, PR, is passed into the constructor for the ECLSession object. Default settings are used for any properties not manually defined. Once the ECLSession object is created, `StartCommunication()` can be called to begin communicating with the host.

C) As you may already know from 2.3.3, "Access the properties and methods of those objects" on page 19, most functions are available from the Presentation Space object. The TOTALS applet receives a handle to the current presentation space with this line of code.

D) Several HACL for Java functions throw exceptions and errors. It is necessary for the TOTALS applet to catch any errors that might occur.

2.7.2.2 Logging in

Once the TOTALS applet connects to the host, it must log in the user, navigate to the VM prompt, execute TOTALS, and display the available weeks for selection. In essence, the TOTALS applet is navigating between the

various screens sent by the host. But how? Look at the code to send the user's name and password:

```
try {
    ps.SendKeys(UI.UserID, 20, 16);
    ps.SendKeys(UI.Password+"[enter]", 21, 16);
```

Figure 30. TOTALS applet code that sends a user name and password

Notice that `SendKeys()` uses additional column and row parameters. The applet sends data that is part of the internal user profile (created by the TOTALS applet) to the respective fields and the host determines if a valid user name and password have been supplied. This section of code also demonstrates an actual use of the key mnemonics that are a part of HACL. The "[enter]" substring indicates for the host that an Enter key equivalent has been pushed/sent. We could have also used the ECLConstants.ENTER_STR value in place of our mnemonic.

At this point, the TOTALS applet can truly begin its screen navigation. This is accomplished by searching for text unique to each screen and responding by sending the appropriate text. Another methodology is the patented Screen Recognition Technology which is discussed in 2.4.1, “Screen recognition” on page 21.

Below is a section of code that the TOTALS applet uses to determine if it has arrived at the correct screen. Over a ten second period, allowing time for slow network access, the applet checks to see if the host has sent the text characteristic of a particular screen. If all of the text is found, the applet launches the TOTALS program. If not, the program will use coded logic to handle a failed navigation, such as a timeout notification.

```
for (int i = 0; i < 10 && !ready; i++)
{
   Ready = ps.SearchText("Ready", ps.SEARCH_FORWARD);
   OfVi = ps.SearchText("OfficeVision", ps.SEARCH_FORWARD);
   OV = ps.SearchText("OV/VM", ps.SEARCH_FORWARD);
   if ( | (OfVi != 0) | (OV != 0)) ready = true;
   java.lang.Thread.sleep(1000);
   }
if (!ready)
{
  // Handle Failed Navigation
}
else
   ps.SendKeys("totals[enter]");   // enter totals
```

Figure 31. TOTALS applet code that demonstrates internal screen navigation

2.7.2.3 Entering data

Persistent use of the previous technique will ultimately present the applet with the Week Selection screen. In response, the TOTALS applet executes two actions:

- Weeks are pulled off the screen and presented to the user for selection.
- Data is sent back to the host to signify what choice the user made.

The following code shows how the TOTALS applet captures the screen text and presents it to the user.

```
char ch3[] = new char[20];
char ch5[] = new char[20];
ps.GetString(ch3,7,9,33,7);
ps.GetString(ch5,20,4,6,20);
// Translate Strings to Dates
...
// Create a Window for Display
...
```

Figure 32. TOTALS applet code to pull dates out of the presentation space

You may notice that the `GetString()` function accepts four parameters in the above code. Like most other APIs, HACL provides function with multiple parameter lists so that the developer has greater flexibility in writing the corresponding code. The Personal Communications and Host On-Demand documentation will provide you with a complete list of functions.

But business logic is truly the *key* to manipulating and interpreting data from the host. In this example, the applet pulls strings out of the presentation space and uses logic to convert the data into actual dates. A window to display the dates is subsequently created and is intended to let the user choose which week they are entering time for. This brings us to the user's second step.

Step 2: Choose the week for which data is to be entered and click the **Continue** button.

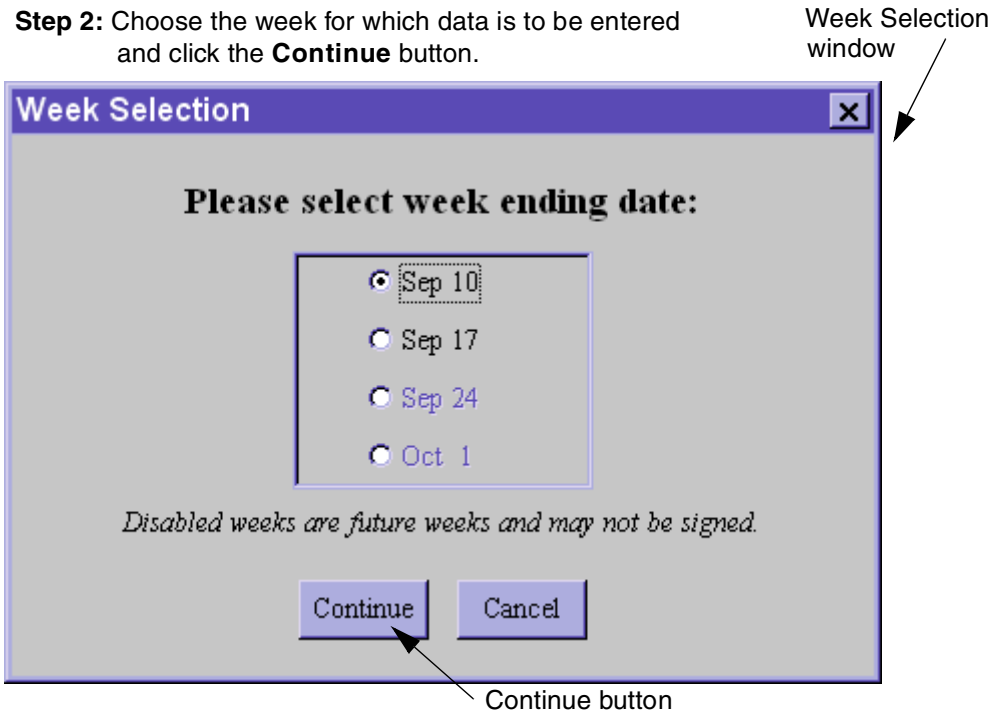


Figure 33. Week Selection window presented by the TOTALS applet

Remember that Step 2 for the original method was to log on to the host. As you can see, the TOTALS applet has already simplified the process of entering data tremendously by automating most of the steps needed to use TOTALS.

Once the Continue button is clicked, the applet takes the data previously entered and places it within the respective fields. For a simplified example, see the following:

```
for (int i = 0; i < 7; i++)// For Each Day
{
  if (D[i].Worked)
  {
//Position cursor
...
    ps.SendKeys("/[enter]");

// Error Check
....
ps.SendKeys(D[i].HoursStart,12,55);
ps.SendKeys(D[i].HoursStop,13,55);
ps.SendKeys("[enter]");
```

Figure 34. TOTALS applet code that demonstrates sending daily information using business logic

For each applicable day of the week, the TOTALS applet marks the day worked with a "/" and sends the Enter key for confirmation. When TOTALS asks for the exact hours worked, the applet sends a set of strings that represent the start and stop times. The TOTALS applet needs to "sign" the timesheet for the user; hence, the third step is presented to the user.

2.7.2.4 Step 3: signing the timesheet

The applet presents the user with a dialog of summarized data. It is the user's last chance to make any corrections before the data is officially submitted to payroll.

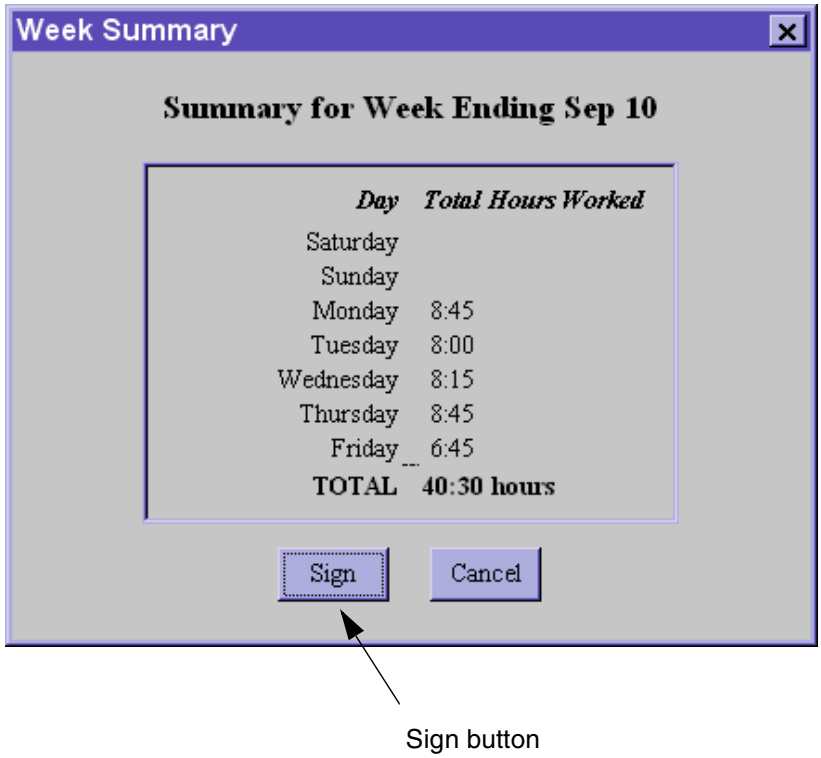


Figure 35. Final confirmation dialog with hourly summary

Clicking the **Sign** button sends the user's PIC to the host. The same techniques regarding sending text to particular locations on the screen still apply. Additionally, the applet exits the TOTALS program and sends the "logoff" string to the host to end communications.

2.8 Summary

Clearly, using the TOTALS applet simplified the entire process of signing a timecard. By this example alone, IBM employees achieved:

- Accuracy
- Speed
- Simplicity

But these traits are easily transferable to other programs written with the Host Access APIs, such as Host Access Controls for ActiveX or HACL Automation Objects, for that matter. All of this functionality and more is just as easy to

implement as it looks. If you provide the logic surrounding your communications, HACL will provide the functions needed to interact with the data.

The source code and all associated files are located on the CD-ROM included with this redbook.

2.9 Documentation and interesting reading

The HACL for Java documentation is provided in HTML format with Personal Communications and Host On-Demand. This reference provides an excellent HTML-style reference for the HACL classes.

- For Host On-Demand the documentation is in:

  c:\hostondemand\lib\en\doc\hacl\API_users_guide.html

- For Personal Communications the documentation is in:

  c:\Program Files\Personal Communications\doc\hacl\ECLReference.html

Another source of information is the documentation placed on the IBM Web site. Use the following link to supplement the information above:

```
http://www-4.ibm.com/software/network/technology/hacl/
```

Other good references:

- *Personal Communications Version 4.3 for Windows 95, 98, and NT,* SG24-4689
- *IBM SecureWay Host On-Demand: Enterprise Communications in the Era of Network Computing*, SG24-2149-00
- *eNetwork Personal Communications V4.3 for Windows 95, Windows 98, and Windows NT: Host Access Class Library*, SC31-8685

Chapter 3. HACL for Java - ECLApplets

ECLApplets are created with the same HACL for Java classes as seen in Chapter 2, “HACL for Java - applets and applications” on page 11; however, the programming and usage models are different enough that we found it necessary to devote this small chapter to the subject.

The Run Applet feature of Host On-Demand and Run Java Applet feature of Personal Communications execute user-defined applets which run concurrently with the HOD or PCOMM GUI. Of course, this is different from creating independent programs that do all of the emulation internally. ECLApplets interact with the emulator’s current ECLSession by passing a reference to the applet’s constructor.

So how might ECLApplets benefit the user? One way is to design them to function like macros. For example, an ECLApplet could enter a user’s name and password when it sees a prompt. Perhaps you need a program that counts how many times a certain string of text appears on a screen? The possibilities are truly endless.

This chapter will discuss how to develop an ECLApplet class in two ways: macro-like and monitor-like. Both structures essentially create "helper applications" in which tasks are automated, data from the session is processed, or both. It may be better to consider this chapter as an addendum to Chapter 2, “HACL for Java - applets and applications” on page 11. Much, if not all, of the information from that chapter will be required to fully understand the material contained herein.

In this chapter, we will use an ECLApplet developed for internal IBM use, called SessionAssist!. Imagine logging into a host that you had never used before. What commands do you use? How would you go about getting help? The idea to create a program that provided on-the-fly "cue cards" to navigate through a host-based session was finally realized in the summer of 1999 with the development of SessionAssist!. Founded upon the ECLApplet programming model, SessionAssist! demonstrates how a fully customizable help system can be used with sessions running in Host On-Demand or Personal Communications. We will discuss at length the SessionAssist! applet.

3.1 Establish the environment

The environment to *develop* ECLApplet classes should mirror that of Chapter 2, “HACL for Java - applets and applications” on page 11. Since both Host On-Demand and Personal Communications support ECLApplets, referencing habeans.jar or pcseclj.jar in the classpath will suffice. VisualAge for Java and Visual Cafe can be used to create ECLApplets as long as the class files are imported into each IDE in the proper manner. Refer to 2.2, “Establish the environment” on page 12 and follow the steps that apply to your choice of development environments.

To *execute* the ECLApplets, simply use the appropriate pull-down menu as discussed in 3.3, “Deployment and testing” on page 56.

3.2 Getting started

The key to creating an ECLApplet class for Host On-Demand or Personal Communications is the ECLAppletInterface provided by HACL for Java. This interface defines a special `init()` function which accepts a parameter of type ECLSession. After the emulator creates the ECLApplet object, the `init()` function is called immediately afterwards.

There are five basic steps to creating an ECLApplet:

1) The ECLAppletInterface will have to be imported like any other HACL class file:

```
import com.ibm.eNetwork.ECL.*;
import com.ibm.eNetwork.ECL.event.*;
```

2) A Java interface provides a consistent way for other programs or class files to manipulate an object. Any class file that utilizes a specialized interface, such as ECLAppletInterface, must implement it on the class definition line:

```
public class interfaceSample implements
   com.ibm.eNetwork.ECL.ECLAppletInterface
```

3) It would be wise to include a global member variable for the ECLSession and the subsequent ECLPS objects:

```
ECLSession s = null;
ECLPS ps = null;
```

4. Host On-Demand and Personal Communications use a no-parameter constructor for your class file. Ensure that this type of constructor is placed within your class file:

```
public interfaceSample()
{
}
```

5. The final required piece of code is the definition for the `init()` function. The ECLSession object being passed into this function is the class file's link to the current session. It is imperative to set the values of your two global variables within this body as well:

```
public void init(ECLSession session)
{
   s = session
   ps = s.GetPS();
}
```

At this point, your class file can be created by Host On-Demand and Personal Communications, but it does not do anything. Further development of your class file will lead to one of two models: macro-like or monitor-like ECLApplets.

Tip: Recall that the `IsCommReady()` function will return whether or not the ECLSession object is able to use such functions as SendKeys. It is even more important to add the associated while loop to the `init()` body of an ECLApplet or much, if not all, of the associated functionality of the class file may be lost:

```
while (!s.IsCommReady())
{
    continue;
}
```

3.2.1 Macro-like ECLApplets

A macro-like ECLApplet class can execute a set of functions upon the active session and terminate - similar to that of a macro or script. To achieve this, these functions would be placed within the body of the `init()` function. Both Host On-Demand and Personal Communications wait for this function to finish executing before continuing on with their own internal routines. An example of this is the previously mentioned user name and password "automator". Once the ECLApplet class completes its routine, it is terminated, but until then, the emulator waits. Look at the next example:

```
public void init(ECLSession session)
{
    s = session;
    ps = s.GetPS();
    ps.SendKeys("myUsername", row, col);
    ps.SendKeys("myPassword[enter]", row col);
}
```

3.2.2 Monitor-like ECLApplets

Whereas the macro-like ECLApplet terminated after execution, you can create a class that executes for an indefinite period of time. The key is to make the class file run on its own thread. Host On-Demand and Personal Communications will proceed as normal; however, a program attached to the active session will be running concurrently in the background, either visibly or invisibly.

The process of writing threaded ECLApplets seems complicated at first, but the programming structure is relatively simple and easy to follow. Read the steps below for a better understanding:

Changes to original class file

1. Import the appropriate Java thread package into your project by adding the following line:

   ```
   import java.lang.Thread;
   ```

2. The `init()` function must create a Thread object at this point or else the class will terminate. The Thread object should actually be a subclass of Thread because you will have to define what actions are performed when it is executing. Step 3 will help you to define the Thread subclass, but for now, create your new Thread within the `init()` function:

   ```
   public void init(ECLSession session)
   {
      s = session;
      ps = s.GetPS();
      subclassThread sThread = new subclassThread(s, ps);
      sThread.start()
   }
   ```

Defining the Thread subclass - second class file

3. As usual, import the necessary class files, define ECLSession and ECLPS member variables at a global level, and set those variables equal to the

constructor's parameters. Additionally, since this is a subclass of the Thread class, the class definition must extend its parent and call `super()`.

The following demonstrates step 3 in its entirety:

```
import java.lang.Thread;
import com.ibm.eNetwork.ECL.*;
import com.ibm.eNetwork.ECL.event.*
public class subclassThread extends java.lang.Thread
{
    ECLSession s = null;
    ECLPS ps = null;
    public subclassThread(ECLSession session, ECLPS newps)
    {
        super();
        s = session;
        ps = newps
    }
}
```

4. A thread, as you saw in the main class file, has a `run()` method. Within this function's body should be any session work that needs to be executed. Our example uses specific intervals to execute the instructions. The example below monitors the presentation space for the string "You May Disconnect Now...". When it finds the text within the presentation space, the ECLApplet class replies "logoff" and terminates. If the text is not found, the program sleeps for 500 milliseconds and checks again, in essence, polling the active session. Of course, your program may search for different text, respond differently, and sleep for different intervals.

```
public void run()
{
    if (s.SearchText("You May Disconnect Now...") )
    {
        ps.SendKeys("logoff");
        currentThread().stop();
     }
     try {
          Thread.sleep(500);
     } catch (Exception e) {
            e.printStackTrace();
     }
}
```

3.2.2.1 Notes and tips

The term "ECLApplet" can be misleading because Host On-Demand and Personal Communications do not actually create an applet class. Neither

application hands down a handle to a parent Applet either. As a result, those methods specific only to the Applet class cannot be referenced. With respect to creating and showing visual components, one can use frame objects and set them visible to add a graphical user interface to an ECLApplet class. The SessionAssist! example provided later in this chapter is a demonstration of an ECLApplet class with a graphical user interface.

3.3 Deployment and testing

Once you have created your ECLApplet class files, they must be placed in the appropriate locations for Host On-Demand and Personal Communications to find them.

For Personal Communications:

Copy the class file(s) into the private directory under the Personal Communications main directory. To execute your ECLApplet:

1. Click **Assist.**
2. Select **Run Java Applet.**
3. Enter the name of the class file that implemented ECLAppletInterface. In the example above, one would enter `interfaceSample`. The .class extension is optional.

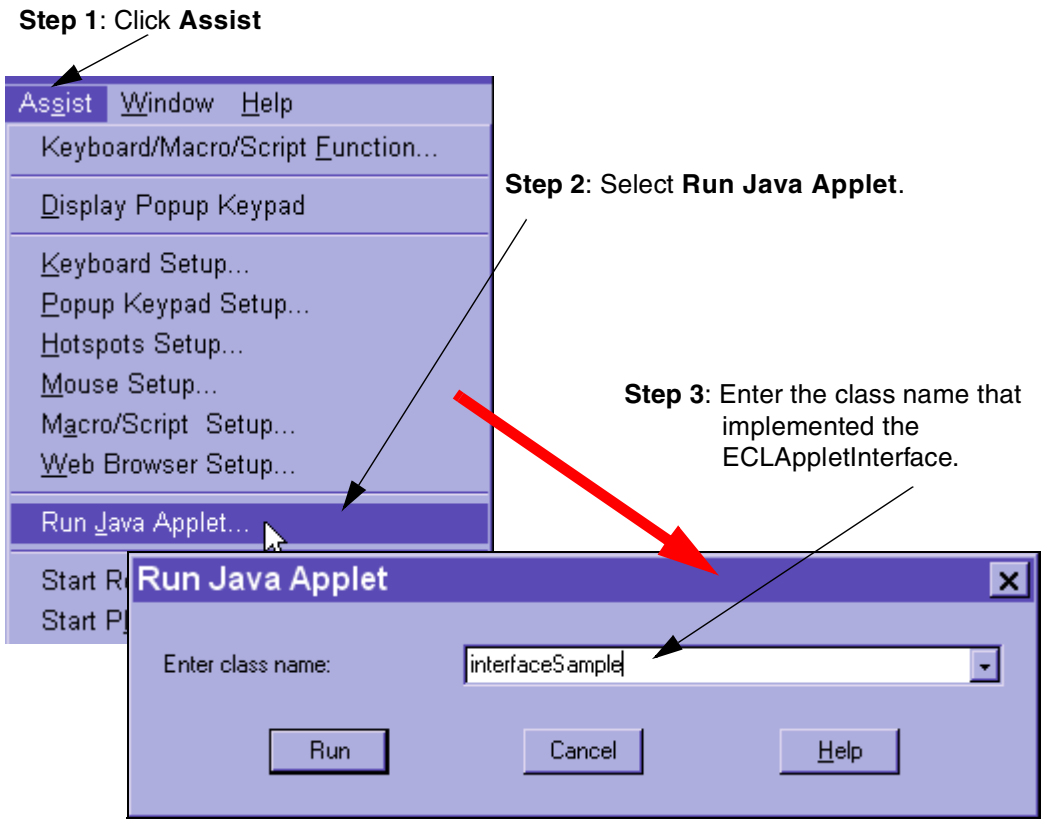


Figure 36. Executing an ECLApplet with Personal Communications

Unfortunately, there is no Java console with Personal Communications, which limits any serious debugging efforts. The best solution is to test your ECLApplet in a Host On-Demand environment, if possible, and use the browser's internal Java console.

For Host On-Demand:

The technique used to execute an ECLApplet with Host On-Demand is almost identical to the procedure used for Personal Communications. The primary difference is that Host On-Demand searches for the ECLApplet class files in the system's classpath instead of a private subdirectory. One suggestion is to append to the system's classpath the directory where all of your ECLApplet classes were compiled. The next three steps explain how to launch your ECLApplet in Host On-Demand once the files are located in the classpath:

1. Select **Assist.**
2. Click **Run Applet.**
3. Enter the name of the class file which implemented ECLAppletInterface. The .class extension is optional.

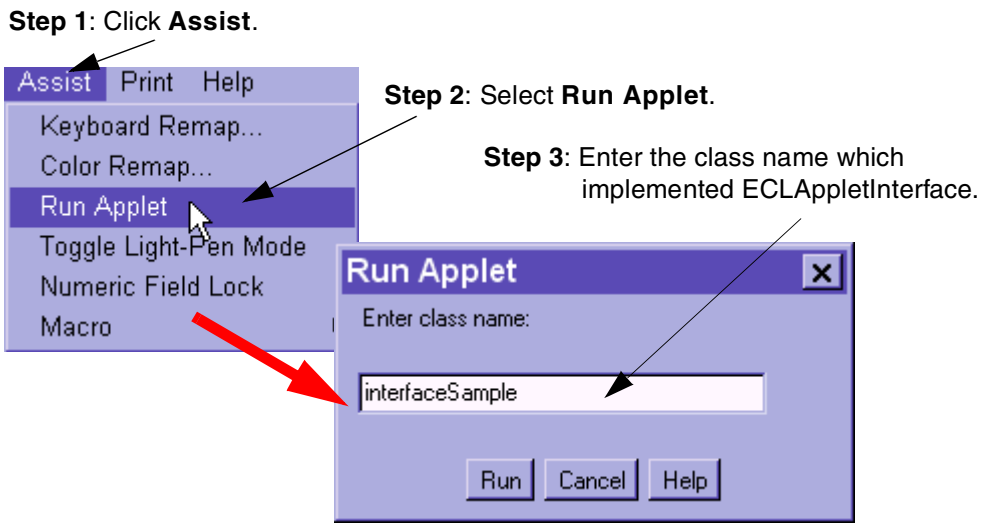


Figure 37. Executing an ECLApplet with Host On-Demand

3.4 Example: SessionAssist! ECLApplet

The following example, SessionAssist!, is a fictional example designed for the sole purpose of demonstrating the usefulness of monitor-like ECLApplets. The SessionAssist! ECLApplet is property of IBM.

Navigating through an unfamiliar terminal session can be frustrating. A user may not know which commands he/she can execute or how to get help. But what if you could use Personal Communications or Host On-Demand with a help system that informed you about each screen as you navigated? New hires are just one group of people who could certainly use supplemental information for the first few times they use a terminal session.

SessionAssist! is just that - a help system that runs with either Personal Communications or Host On-Demand and displays "flashcards" of information for each screen. The example can serve as the foundation for a much more elaborate help system should you choose to develop one in greater detail, but for now, it is just an extensible proof of concept.

3.4.1 SessionAssist! in action

SessionAssist! is a monitor-like ECLApplet, meaning it watches the current session and makes adjustments when screens change. A database (referred to as the cardfile for this example) exists with information to compare to screens. When a user enters screen A, if a definition matching the screen exists in the cardfile, the help window will reflect any pertinent information about that screen.

SessionAssist! displays a rectangular window which contains two menus:

1. **File**
   - Exit
2. **Options**
   - Preferences (used to set interval in milliseconds to check session.)
   - SessionAssist! CardFile Designer (create new cards for cardfile.)

The remainder of the window is the location for any information. The examples below follow the screen layouts of `ralvms.raleigh.ibm.com`. When using the supplied cardfile database on the CD-ROM (cardfile.dat which should be placed in the Personal Communications or Host On-Demand main directory), a window like that shown below will appear when Personal Communications or Host On-Demand has attached to the host and displayed the login screen.

Figure 38. SessionAssist! with supplied cardfile showing predefined information about the login screen for ralvms (VM system)

As you can see, the window presents a set of messages to the user in an attempt to facilitate a better understanding of the screen that actually appears in either Host On-Demand or Personal Communications. Once a user logs into the host, the window above should reflect a different set of messages.

For example:

Figure 39. SessionAssist! changed to reflect a new set of help messages for a new screen

The above card would appear in SessionAssist!'s window if a user logged in correctly.

3.4.2 SessionAssist! under the hood

We have seen SessionAssist! at work, but we have not seen how it works yet. Remember: this is a monitor-like ECLApplet class and it checks the current session after each specified interval. But what does it check for exactly? Prior experience with HACL should have told you that the ECLApplet is searching for unique text within each screen. The cardfile, which is a flat file, defines the following attributes for each card, including the unique text:

- Title: text color
- Statements: text, color, and X,Y coordinates
- Background color
- Unique text which identifies the associated screen

After each interval, SessionAssist! walks through its cardfile and checks the strings of unique text against what appears in the presentation space. If the `SearchText()` function returns a valid linear position, the respective title, statements, and color will now be placed within SessionAssist's window, giving the appearance of displaying a new card. The searching is done within the `run()` function of the Thread subclass. The following is a snippet of this code:

```
 for (int i = 0; i < cardFile.size(); i++)
{
     if // Unique Text is found
       {
         //Only repaint if it is a new screen
         if (debug.topicIndex != i)
         {
            debug.topicIndex = i;
            debug.repaint();
         }
     }
}
try {
    Thread.sleep(runner.updateTime);
} catch (Exception e) {
```

Many of the coding techniques in the SessionAssist! program have already been discussed at length. If you are familiar with Chapter 2, “HACL for Java - applets and applications” on page 11, you should be able to design an ECLApplet that can do just about anything.

Note: ECLApplets inherit full security permissions. Normally, an applet has to request permission to write to files or communicate with a different host other than where it was downloaded from.

3.5 Summary

The SessionAssist! ECLApplet is just one example of how programs can be a great benefit to the user when they are designed to run concurrently with the emulator. ECLApplets can be written to function the same as any program since they have permission to act like an application and can reference all resources in the Java Development Kit. Entire applications can be written from the ECLApplet model.

Chapter 4. Host Access Beans for Java

With Java 1.1, Sun introduced JavaBeans: reusable pieces of code with methods and properties contained in visual objects. JavaBeans are supported by compilers and virtual machines that meet the Java 1.1 specification.

IBM made a solid effort to expand the Host Access Class Library into visual programming environments. As a result, IBM developed component software which allows developers to write applications to communicate with host systems. These beans, which are applications of the Host Access Class Library itself, significantly reduce development time by reducing the number of managed files/objects to only eight or nine, depending on the product you purchased.

In this chapter, we will discuss the Host Access Beans for Java. The next chaper will discuss Host Access Controls for ActiveX, the second of the two component software packages created from the Host Access Class Library.

Below is a list of the Host Access Beans for Java:

Table 2. Host Access Beans for Java list

JavaBean Icon	Name/Description
	**Session -** This non-visual bean provides methods and properties for setting up and establishing communications with the host system.
	**Screen -** This visual bean provides the graphical interface for displaying the host data from a Session bean.
	**Terminal -** This visual bean combines the Session and Screen beans to provide a composite bean that encompasses both communication with the host and the graphical interface for displaying the host data.
	**KeyPad -** This visual bean provides a simple grid of buttons which invokes various host functions.

JavaBean Icon	Name/Description
	**KeyRemap -** Using KeyRemap, keystrokes can be mapped to alternate characters or directly to host functions.
	**FileTransfer -** This visual bean provides a toolbar interface for transferring files to and from a host.
	**Macro -** This non-visual bean records and plays a single macro. Macro employs advanced screen recognition technology to reliably navigate host applications in any environment. Macro also provides the ability to prompt for user input and extract text from the screen during playback.
	**MacroManager -** This visual bean provides a toolbar interface for managing multiple macros. The MacroManager bean allows you to record, play, load, delete, and edit multiple macros.
	**ColorRemap -** This visual bean provides a simple interface for modifying the colors displayed by the Screen or Terminal beans.

The beauty of the Host Access Beans for Java is that a developer can access the HACL for Java methods and objects for more specific development purposes. Instead of just providing pieces of an emulator to piece together, Host Access Beans for Java allow a developer to implement such features as screen recognition, monitoring the presentation space, and any other abilities provided by HACL for Java.

At this level, Host Access Beans for Java are not that different from working with the standard HACL for Java classes. To demonstrate a popular customer request, we will provide an example called Double Time which creates two active sessions and facilitates communication between them.

Note: It will benefit you greatly if you understand the material contained in Chapter 2, “HACL for Java - applets and applications” on page 11. Host Access Beans for Java are built upon HACL for Java and much of the information in that chapter applies to this one.

4.1 Benefits and limitations

JavaBeans, in general, epitomize object-oriented programming. All the functionality one would need to communicate with a host is supplied in these components. A programmer moves further away from working with individual objects and develops a dependency to work with larger structures, such as a Terminal bean. But Host Access Beans for Java truly shine when applied in visual development environments, such as IBM's VisualAge or Sun's BeanBox. These environments allow you to set properties at design-time and significantly reduce the amount of code your program needs to function.

Host Access Beans for Java also fire events and those events can be tied to other JavaBeans. Tying Host Access Beans together is called "wiring". See 4.3.0.2, "Wiring Host Access Beans together" on page 71 for more information.

Note: Again, ensure that your Java Virtual Machine is compliant with the Java 1.1 specification.

4.2 Establish the environment

Even though JavaBeans are intended for use with visual Java environments, Host Access Beans for Java can be referenced in non-visual environments such as a standard text editor. The HACL files used in the first two Java-based chapters include the Host Access Beans subset. Refer to 2.2, "Establish the environment" on page 12 for information on how to establish your environment. Using the beans in a non-visual environment is no different from using non-bean classes.

In addition to the environments discussed in Chapter 2, "HACL for Java - applets and applications" on page 11, it is important to explain the BeanBox which was developed by Sun Microsystems and provided in the Beans Development Kit (BDK). BDK 1.0 is intended to support the JDK 1.1 specification.

4.2.1 Loading Host Access Beans for Java with Sun's BeanBox

The BeanBox is Sun's first attempt at providing a visual environment for programming with JavaBeans. The steps below explain how to load the Host Access Beans for Java into Sun's BeanBox.

1. Select **File.**
2. Click **LoadJar.**

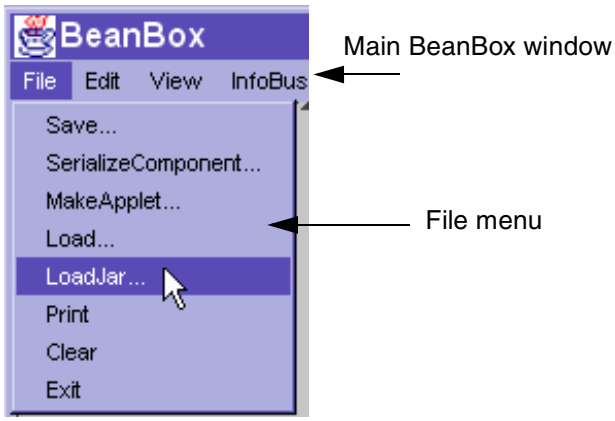


Figure 40. BeanBox file menu used to import the Host Access Beans for Java

3. A file dialog should now appear:
   - Locate habeans.jar if you installed Host On-Demand.
   - Locate pcseclj.jar if you installed Personal Communications.

   Click **Open**.

If the BeanBox did not have any problems with the files, the ToolBox window should now reflect the added Host Access Beans for Java. The figure below shows how the Host Access Beans for Java provided with Personal Communications and Host On-Demand appear when added to the BeanBox.

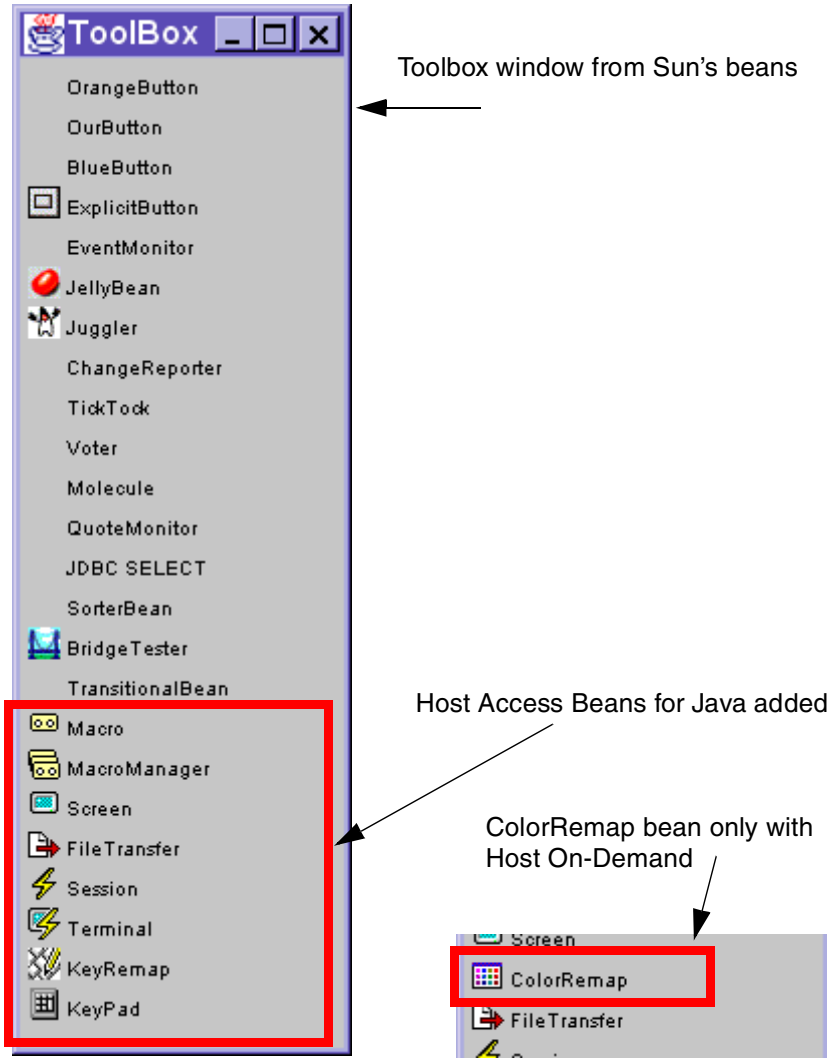


Figure 41. Host Access Beans loaded with Sun's BeanBox

4.2.2 Loading Host Access Beans for Java with VisualAge for Java

Section 2.2.3, "Loading the HACL for Java API with IBM's VisualAge for Java" on page 14 discusses how to add HACL for Java and goes a step further in explaining how to import the Host Access Beans for Java. Please refer to this section if you plan to use VisualAge and Host Access Beans for Java.

4.2.3 Loading Host Access Beans for Java with Visual Cafe

Symantec's Visual Cafe makes importing external JavaBeans a very simple process:

1. Click **File.**
2. Select **Add Component to Library.**
3. A File Dialog should appear. Enter the name of the file that you wish to use. For Personal Communications, use pcseclj.jar. For Host On-Demand, select habeans.jar located in the hostondemand\hod\lib\toolkit\jars directory.

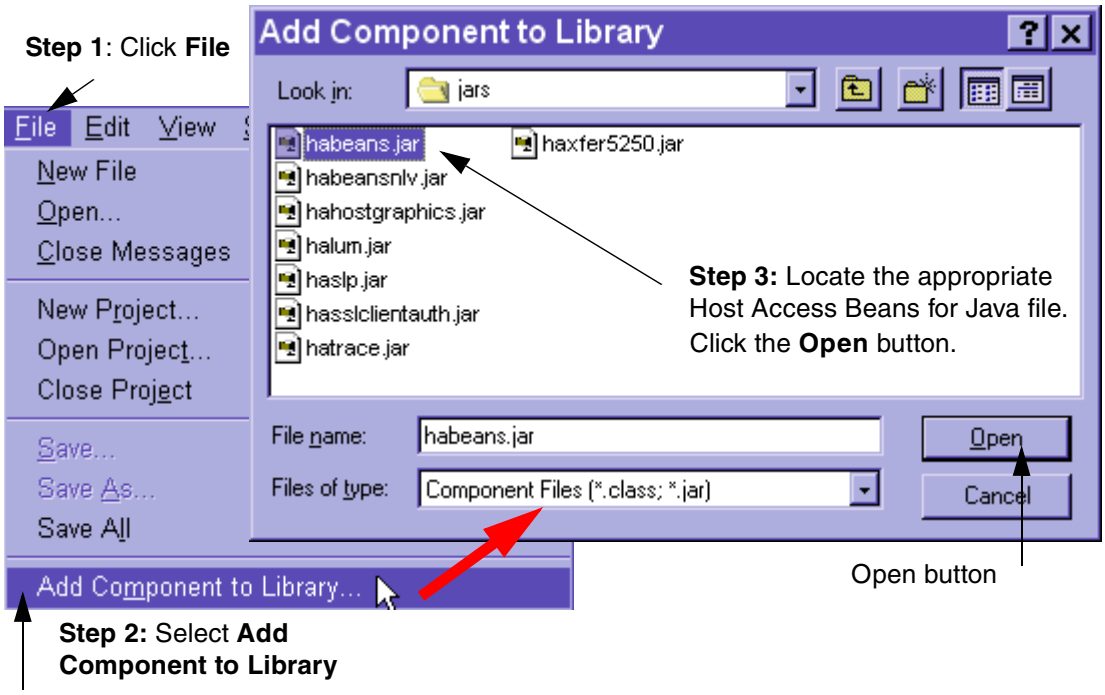


Figure 42. Adding Host Access Beans to Visual Cafe

4. Depending on which file you added, Visual Cafe will respond with one of two message boxes. If you added the Host Access Beans for Java provided with Host On-Demand, nine components should be added to the project.

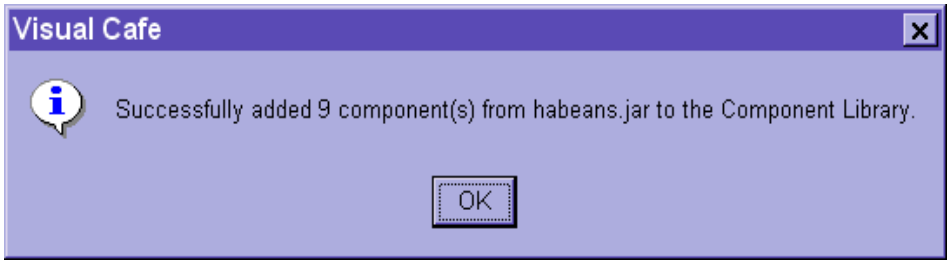


Figure 43. Visual Cafe successfully added the Host Access Beans for Java with Host On-Demand

Because Personal Communications does not contain the ColorRemap bean, only eight components are added from pcseclj.jar.

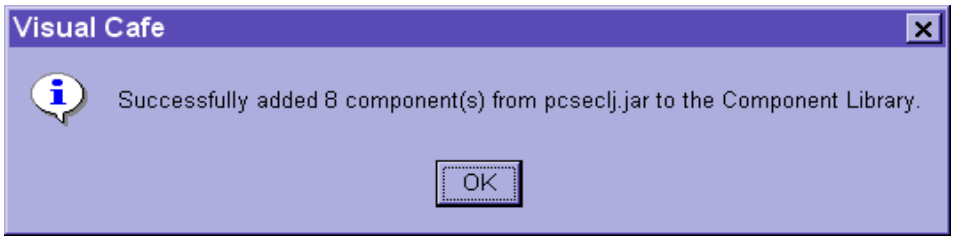


Figure 44. Visual Cafe successfully added the Host Access Beans for Java with Personal Communications

4.3 Getting started

We mentioned earlier that JavaBeans in a non-visual environment are handled in much the same way as non-bean objects. Remember the example provided in 2.6, “Example 1: JavaSample applet” on page 25? Recall that we used a Terminal bean to further demonstrate the HACL for Java functions. That Terminal object was created with the following line of code:

```
term = new Terminal(p);
```

`P` was a Properties object used to manually set several parameters such as host name.

But we want to focus on the *real* usability provided by Host Access Beans for Java in an IDE such as IBM’s VisualAge or Symantec’s Visual Cafe. Creating a Host Access Beans for Java-based program follows the same two-step process for any visual environment:

1. "Drop" your components into the workspace.
2. Edit the underlying code - add business logic and define events.

4.3.0.1 Customizing Host Access Beans for Java

We mentioned before that many properties can be set at design-time. Setting these properties at design-time is achieved when several Host Access Beans for Java provide a graphical interface *inside* the development environment to set each of the parameters. This feature is known as customizing a Host Access Bean for Java. For example, when one is working with the Terminal bean and selects **Customize** from any editing menu, the following window appears:

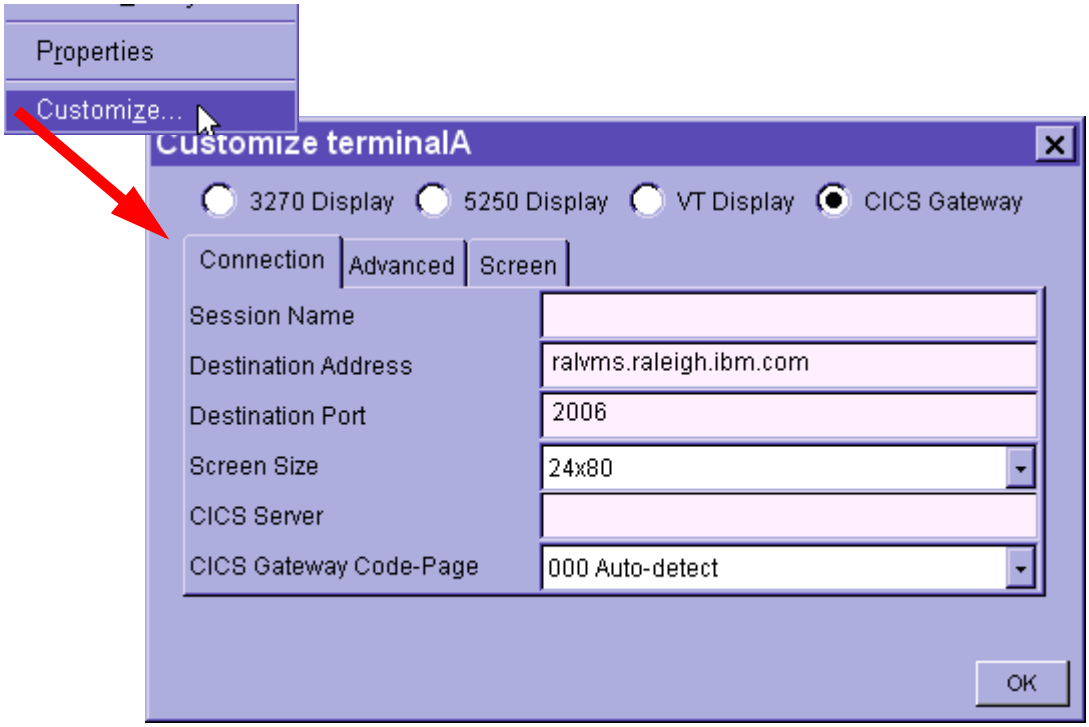


Figure 45. Customizing a Host Access Terminal bean - setting each property

In contrast, setting a property during the application's execution is known as setting the property at run-time; however, many of the properties are already defined because each Host Access Bean for Java uses default values for those not specified. You should get familiar with the properties available in

these windows. Doing so will help you tailor your code and components to best suit your needs.

Customize windows are available for the following beans:

- Terminal
- Screen
- FileTransfer
- Session

Tip: In addition to using these windows inside the development environment, they can be invoked directly as well. For example:

```
TerminalCustomizer tCust = new TerminalCustomizer();
tCcust.setObject((Object)myTerm);
```

These two lines of code create a TerminalCustomizer object. We would then need to add it to a container:

```
add(tCust);
```

There are customizers available for Terminal, Screen, FileTransfer, and Session beans.

Note: HACL objects and the Host Access Beans for Java are "invalidated" when certain session/terminal properties are changed. These objects and beans are then validated when the connection to a host is established. We encourage not changing properties inside the session/terminal objects/beans unless there is no communication with the host.

4.3.0.2 Wiring Host Access Beans together

Each bean sends and listens for specific events. Wiring two Host Access Beans together means assigning component A to listen for events from component B. In Java, component A is called a *listener*.

For example, the KeyPad bean generates a SendKeysEvent when one of its buttons is pushed. The Terminal bean needs to be notified of this event so it can send the key on to the host. The following line of code establishes the Terminal as a listener for SendKeysEvents:

```
myKeyPad.addSendKeysListener(myTerminal);
```

Conversely, to notify the KeyPad bean of any PropertyChange events, add:

```
myTerminal.addPropertyChangeListener(myKeyPadKeyPad);
```

The two previous lines would be placed within the initialization body of a Java program. Wiring each of the eight (nine with Host On-Demand) Host Access

Beans can be a lengthy chore, but you only have to wire the components you choose to use. Consult the Host Access Beans documentation provided in either of the two directories below for a full listing of the events fired and accepted by each component:

- \Personal Communications\doc\beans\beanReference.html
- \hostondemand\lib\en\doc\beans\beanReference.html

4.3.0.3 Incorporating HACL for Java

For simplicity, Host Access Beans for Java provide the basic functions needed to get your components running and interacting with each other. Fortunately, a developer can access any properties and functions of the current ECLSession just like they would with HACL for Java. There are only two steps required to access this level of functionality inside the Host Access Beans for Java:

1) Place the appropriate import statement at the beginning of your program so the compiler will recognize an ECLSession:

```
import com.ibm.eNetwork.ECL.*;
```

2) The Terminal and Session beans contain a function called `getECLSession()` which returns the ECLSession object:

```
ECLSession myECLSession = myTerminal.getECLSession();
ECLSession myECLSession = mySession.getECLSession();
```

Once you have a handle on the ECLSession object, you can do just about anything, including `GetPS()` to get a handle to a Presentation Space object.

4.4 Deployment and testing

To deploy a program written with the Host Access Beans, follow the same guidelines provided in 2.5, “Deployment and testing” on page 24. If you are developing an applet, be sure to reference the appropriate JAR and CAB files in your HTML tag and adhere to any other browser-specific security issues.

Fortunately, a Java program usually has an associated Java console/output window. Debug statements and use of the Terminal bean (if you are not already) can help determine if your program is making a successful connection to the host and acting as expected.

4.5 Example: Double Time

Emulators keep sessions separate from each other. What you do with Host A cannot and will not have any affect on Host B by default. But what if you

needed to pull information out of one session and use it with another? For example, a manager may receive a list of people as project contacts with Host A, but also needs to look up each person's phone number using Host B. Normally the manager would have two options:

- Write down each person's name and then look up their phone number.
- Remember the name, locate the respective phone number, and repeat.

Either way, the process is very inefficient because one has to navigate two sessions at different times.

Solution: Manage both sessions *at the same time.*

Double Time is a program written with the Host Access Beans that achieves a similar effect. After two sessions have been established, text is pulled out of Session A and written into Session B immediately. In fact, we have incorporated the Terminal beans to show the presentation space provided by each host in real time. Of course, the example does little more than this since we have tried to make it as generic a program as possible. Double Time was written using Visual Cafe 3.0. The source code has been included in the CD-ROM at the back of this book.

The scenario for Double Time is slightly different than the previously mentioned example. Double Time is intended to simulate an administrator logging into the accounts of the users he/she manages. Screen A provides the user's name and the administrator provides the administrative password which unlocks all accounts. Double Time takes the user name (which is selected in Terminal A) and administrative password and opens the user's account as Session B. Double Time presents this second session to the administrator for use.

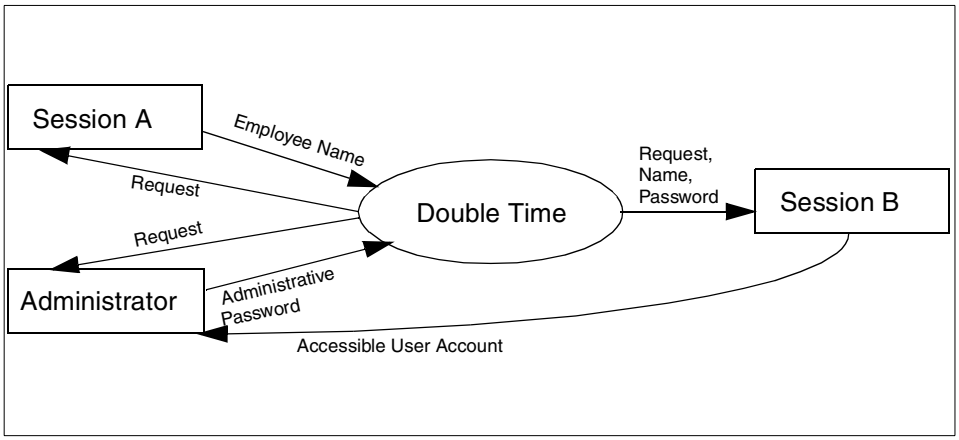


Figure 46. Object interaction diagram for double time

4.5.1 Double Time in action

When Double Time is started, two separate terminal screens appear, Screen A and Screen B, respectively. Below each terminal is a text field used to indicate to which respective host the Host Access Beans will connec. We have also demonstrated a wired KeyPad bean for each terminal session.

Remember: For Personal Communications, a .WS file is expected to be supplied rather than the host name because the session configuration is retrieved from this file.

After host names have been entered and an administrative password has been set with the lower text field, the Connect buttons can be pushed to execute the `startCommunications()` function. Below is a picture of what one half of Double Time looks like when the Terminal bean is connected to a host. Appendix C, “Complete screenshots for HACL examples” on page 131 provides a full screen shot of the program.

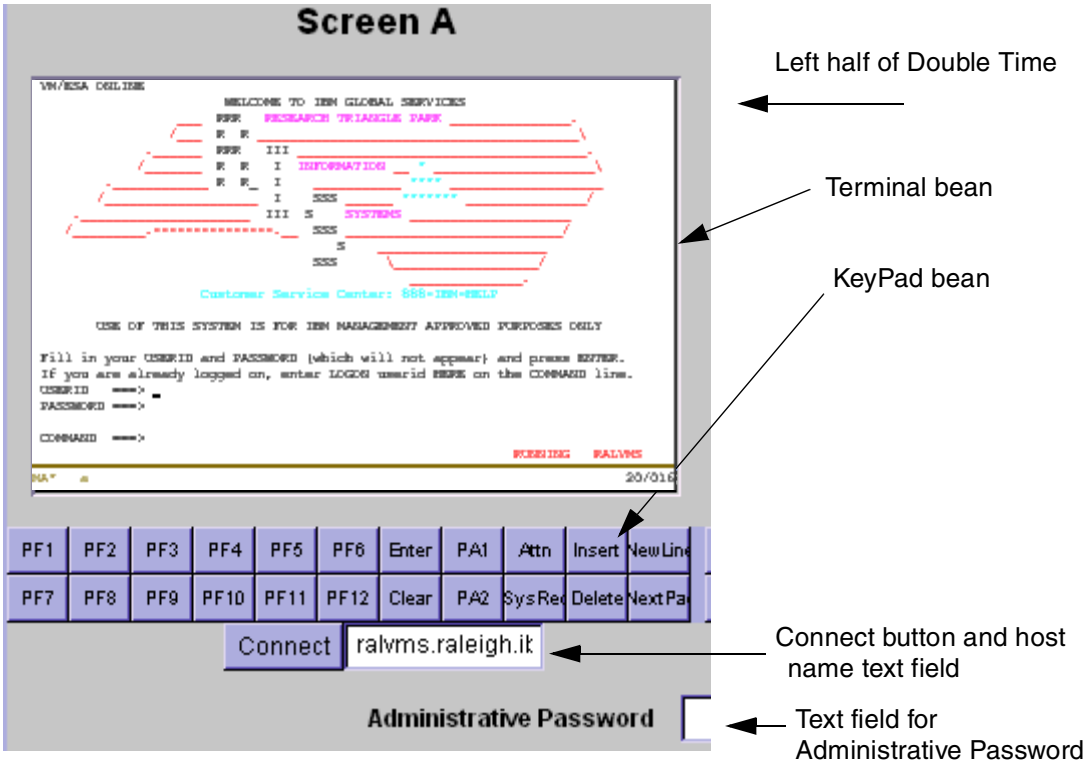


Figure 47. Left side of double time which demonstrates multiple session in a single program with Host Access Beans

Double Time calls the `setHost()` function from the Terminal bean and uses the value from the above text field as the parameter. Once the host is set, `startCommunication()` is executed and a connection is established. There are two things that Double Time must now accomplish:

- Pull text out of the selected presentation space in Screen A.
- Send text to the presentation space in Screen B.

A joint-session

Double Time is designed to pull a string of seven characters from Screen A starting wherever the cursor is located. Those seven characters, which should represent a user name, are entered into the user name prompt for Screen B.

When we look at the function list for a Terminal or Session bean, we notice that the `SendKeys()` function takes a SendKeysEvent object. Why is there not a `SendKeys()` function that takes a string like most other implementations of

HACL? The answer is simple: the available `SendKeys()` function is intended for wired Host Access Beans for Java only. Recall the previous example which explained how to wire a KeyPad bean to a Terminal bean. That is the ideal situation for the `SendKeys()` function. But notice that most of the functions you may already be familiar with in your previous HACL for Java programs are not available from the Host Access Beans for Java. Or so it would seem...

Solution: The alternative is a more non-bean approach for sending and receiving text. Double Time can use the internal HACL for Java workings to obtain a reference to the Presentation Space object. Once the Presentation Space object is available, Double Time executes the familiar `GetString()` function to grab text from Screen A. The Presentation Space object will also execute the `SendKeys()` function which accepts string, row, and column parameters.

As you can guess, all of the work is done within the function body for clicking the Log In button. This block of code has been provided below:

```
{
  ECLPS termA_PS = terminalA.getECLSession().GetPS();
  ECLPS termB_PS = terminalB.getECLSession().GetPS();
  int row_a;
  int col_a;
  char[] username = new char[8];

  try {
        // Step 1 - Pull Text out of rectangle
        col_a = termA_PS.GetCursorCol();
        row_a = termA_PS.GetCursorRow();
        int rc = termA_PS.GetString(username, username.length, row_a, col_a,
                                    username.length - 1);

         // Steps 2 & 3 - Input username from Screen A and
         // and adminstrative password from lower text field
         // into Screen B
         termB_PS.SendKeys(new String(username), 20 , 16);
         termB_PS.SendKeys(pass_tf.getText() + "[enter]", 21 , 16);
   } catch (Exception e) { e.printStackTrace(); }
}
```

Figure 48. Code executed when the Log In button is clicked

The result is seen immediately. The text sitting at Screen A's cursor and the administrative password are sent to Screen B at the coordinates for the login prompt.

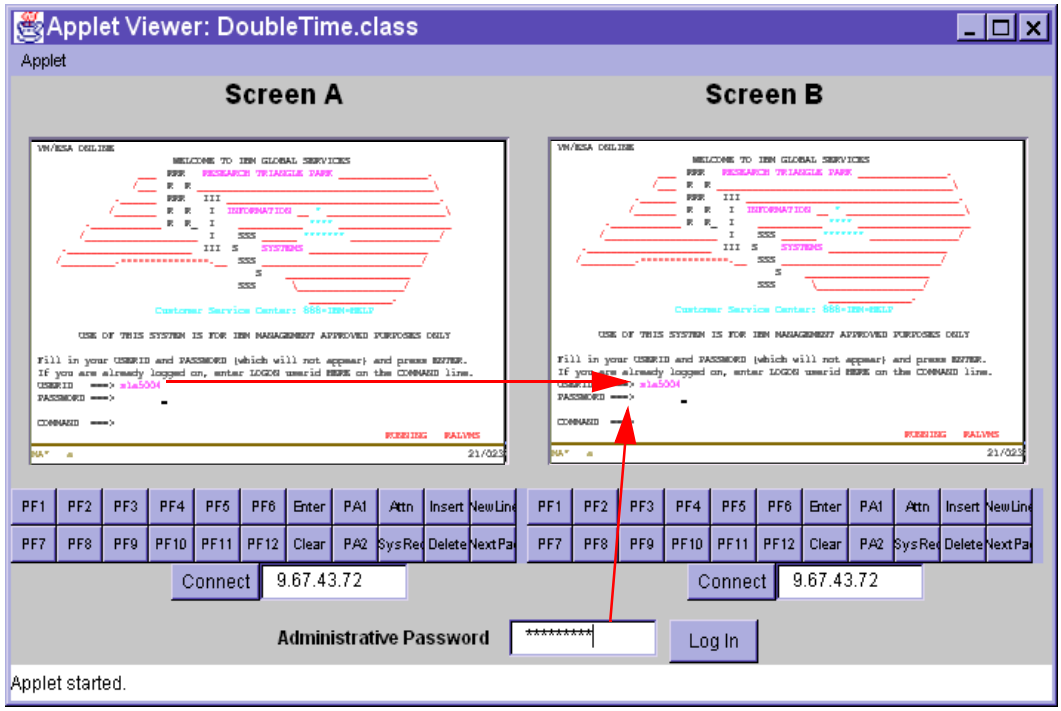


Figure 49. Double Time taking data from the user and Screen A to act upon Screen B

4.6 Summary

Host Access Beans for Java can provide the HACL for Java-style development if initially the beans do not fully meet your host access needs. Otherwise, Host Access Beans for Java should be used in the newer visual development environments such as VisualAge and Visual Cafe. Even more astonishing is how simple it is to create host-based programs that can do things that were not possible just a few years ago. The power to manipulate data between two sessions is just a beginning. Write a Host Access Beans for Java-based program that interacts with however many computers you want to interface with. How much easier can this get?

4.7 Additional references

The Host Access Beans documentation is provided in HTML format with Personal Communications and Host On-Demand. These documents provide excellent HTML-style reference for the various beans.

- For Host On-Demand the documentation is in:

  c:\hostondemand\lib\en\doc\beans\beanReference.html

- For Personal Communications the documentation is in:

  C:\Program Files\Personal
  Communications\doc\beans\beanReference.HTML

Another source of information is the documentation placed on the IBM Web site. Use the following link to supplement the information above:

`http://www-4.ibm.com/software/network/technology/hacl/`

Other good references:

- *Personal Communications Version 4.3 for Windows 95, 98, and NT,* SG24-4689
- *IBM SecureWay Host On-Demand: Enterprise Communications in the Era of Network Computing*, SG24-2149-00
- *eNetwork Personal Communications V4.3 for Windows 95, Windows 98, and Windows NT: Host Access Class Library*, SC31-8685

Chapter 5. Host Access Controls for ActiveX

ActiveX technology was founded upon Microsoft's COM Automation Object structure to provide developers with another set of object-oriented tools. ActiveX controls are tightly integrated into Microsoft Windows, Internet Explorer, and Microsoft Office in addition to many popular non-Microsoft programs.

What relevance does ActiveX technology have with respect to the Host Access Class Library? This book cannot stress enough how determined IBM is to provide developers with the tools to communicate with other hosts on a multitude of platforms. ActiveX is another application of HACL into the Win32 development environment, much like the Host Access Beans for Java. Sun's JavaBeans Bridge for ActiveX places a wrapper around existing Host Access Beans for Java so that they become ActiveX controls. In fact, developers who use the Host Access Controls for ActiveX are really implementing the Host Access Beans with subtle differences. This means that the eight standard components that a developer had access to in Chapter 4, "Host Access Beans for Java" on page 63 are equally available for ActiveX programmers. Again, those components/controls include:

- Session
- Screen
- Terminal
- KeyPad
- KeyRemap
- FileTransfer
- Macro
- MacroManager

An explanation of each control is provided in Table 2 on page 63.

Later in this chapter, we are going to provide an example of these ActiveX controls which caters to the user *and* Web developer. We will not only discuss how to use the functions provided in the Host Access Controls for ActiveX with scripting languages such as VBScript and JavaScript, but also discuss how to create a "funny-looking emulator." This ActiveX example shows how one can develop an emulator to look the way *they* want it to. Think of it as a Web-based emulator - Picasso style.

Refer back to Chapter 2, "HACL for Java - applets and applications" on page 11 and Chapter 3, "HACL for Java - ECLApplets" on page 51 if you are interested in Web-based programming with HACL in Java.

5.1 Benefits and limitations

The ActiveX controls provided with Host On-Demand 4.0 and Personal Communications 4.3 allow a developer to create an emulator using only the pieces they need. Perhaps a user only needs an emulator with the terminal screen and a keypad? Maybe they don't need a terminal screen at all! Maybe we could let new employees use a macro system to automate several of the tasks they will need to eventually learn. These are just a few scenarios where a highly-customizable emulator would be helpful. Sounds a lot like Host Access Beans for Java, right?

With respect to Web browsers, ActiveX controls only run with Internet Explorer. Netscape Navigator does not support these controls as of yet.

We mentioned that the Host Access Controls for ActiveX are essentially built from the Host Access Beans for Java. At this time, the Host Access Controls for ActiveX do not provide an entry point into the core HACL functions such as retrieving a handle to the presentation space or sending text with the `SendKeys(String s)` function as you might see in the Host Access Beans for Java example.

5.2 Establish the environment

If you installed Host On-Demand 4.0 for Win32 or Personal Communications 4.3, the ActiveX controls should have been automatically installed if the correct installation option were selected. If at any time Windows or the ActiveX Control Pad is unable to load an ActiveX control that was provided with these products, reinstall the appropriate program and the error should disappear.

5.3 Getting started

As we mentioned before, the example provided within this chapter caters to the Web developer, excluding Java programmers, of course. Many people know how to program in VBScript and JavaScript, but they do not have the tools to integrate host-based communication into their Web pages.

Incorporating the Host Access Controls for ActiveX into your Web page is a very simple process with Microsoft's ActiveX Control Pad. This utility inserts the necessary tag to load an ActiveX control into your Web page. You can also use the ActiveX Control Pad to write some or all of the VBScript code to "wire" the ActiveX objects together.

You can download Microsoft's ActiveX Control Pad from:

```
http://msdn.microsoft.com/workshop/misc/cpad/default.asp
```

5.3.1 Using the Control Pad to insert ActiveX controls into HTML

The Control Pad creates a new HTML file and inserts the appropriate tags needed to execute your ActiveX control. The following steps detail how to insert the Terminal ActiveX control into your newly created Web page. Inserting other components follows the same process:

1. Click **Edit.**
2. Select **Insert ActiveX Control.**
3. An Insert ActiveX Control Dialog should appear. Scroll down the list and select **IBM Host Access Terminal Control**. You could have selected any of the IBM Host Access Controls to insert into your Web page. Click the **OK** button to continue.

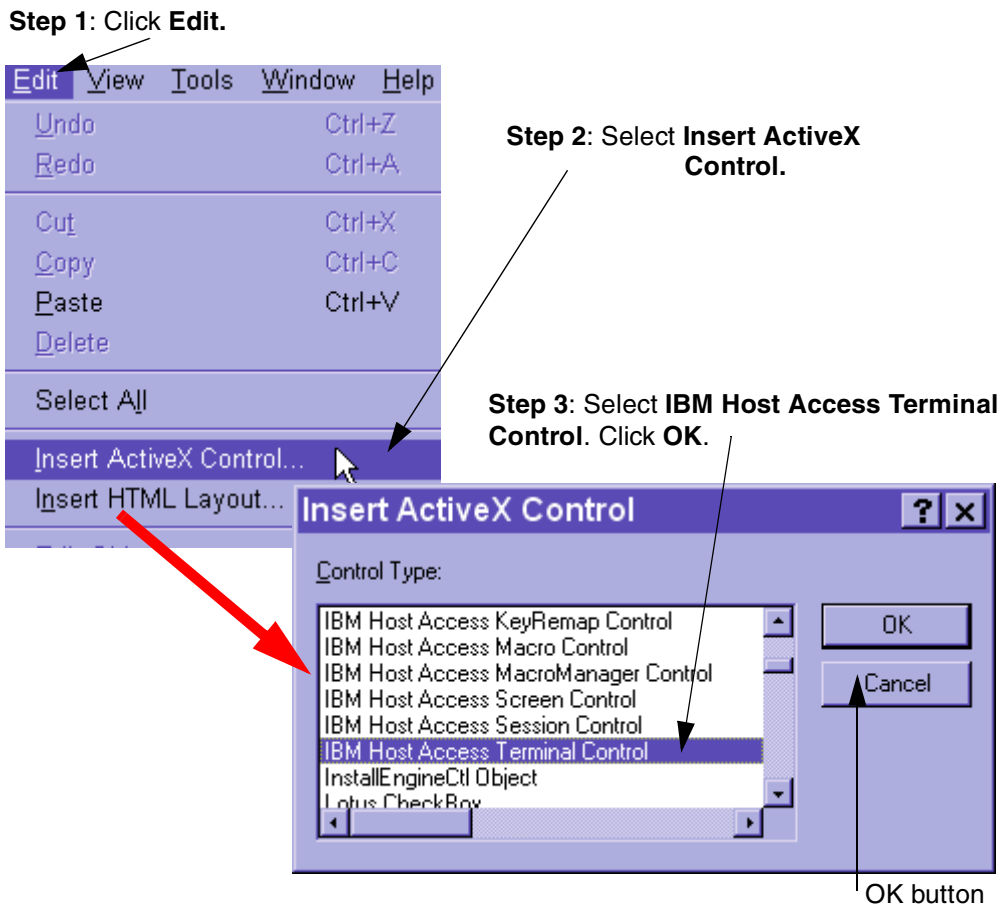


Figure 50. Selecting the IBM Host Access Terminal Control in Microsoft's ActiveX Control Pad

4. The Control Pad should display a Properties window. Scroll to the field labeled Host and type in a valid host name in the upper text field.
5. Click the **Apply** button.
6. Close the Properties window by clicking the **X** in the upper right-hand corner.

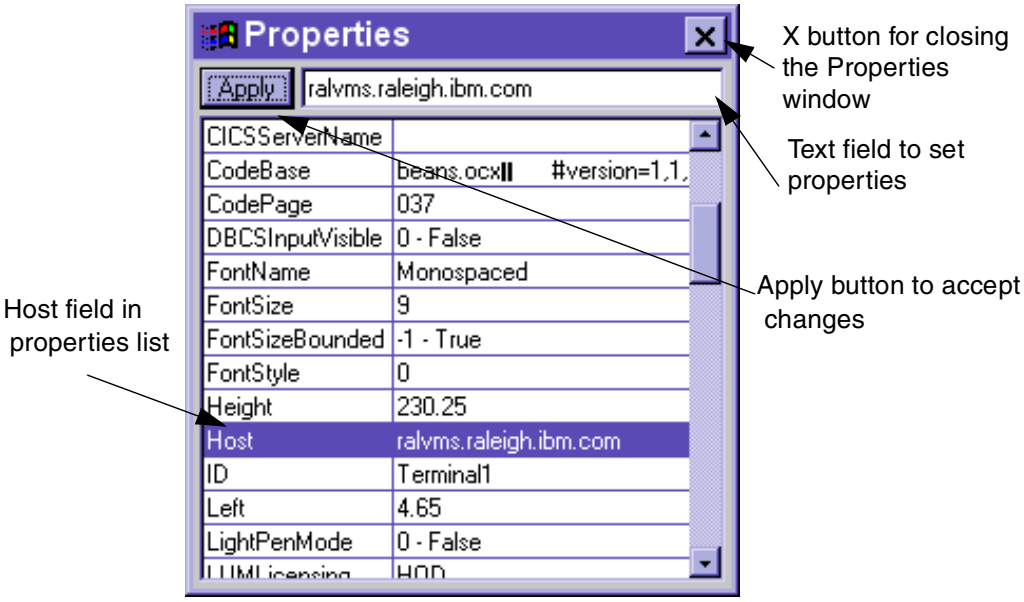


Figure 51. Properties window used to customize the ActiveX Control once it is inserted

Closing the Properties window should show another window with the Terminal control inside. Since communication with a host has not been initiated, it only appears as a black box with blue lines at the bottom. Close this window as well.

The real beauty of the Control Pad can be seen now. The only remaining window remaining should contain HTML text. The ActiveX Control Pad located the specific control, in this case the Terminal control, and created an HTML tag that will load the control locally when a Web browser reads the page. All of the control's parameters have been set and the unique ID for the terminal (42 characters long) has been included.

Figure 52. ActiveX Tag inserted into HMTL with Microsoft's ActiveX Control Pad utility

If you saved this file and loaded it into Internet Explorer, there would not be much more than a black square in the middle of the screen, much like one of the windows you closed inside the ActiveX Control Pad. Why? While an ActiveX control may have been loaded by the Web browser, we still have not told it to perform *any* functions, let alone initiate communications. Enter HTML scripting languages.

5.3.2 Controlling Host Access Controls for ActiveX with VBScript

This section is not intended to be a tutorial on VBScript. It briefly explains how to use VBScript to interact with the ActiveX control.

VBScript, which was also developed by Microsoft, is a Web-centric Visual Basic-style scripting language. Many of the same VB conventions, such as naming functions, are used in this language. Even as simplistic as it may be, the ActiveX Control Pad automatically generates much, if not all, of the code you need to interact with ActiveX controls, in this case, the Terminal control.

Internet Explorer executes the `_onLoad` and `_onUnload` functions for each different page it loads. In this example, we will use the `window_onLoad` function to call the `startCommunication()` function that is a part of the Terminal object.

Note: While this section details how to use the ActiveX Control Pad to insert VBScript, manual insertion of VBScript code is just as acceptable.

1. Select **Tools**.
2. Click **Script Wizard**.
3. The Script Wizard window should appear. Select the `onLoad` function from the window object in the Event list. In the adjacent Actions list, select the `startCommunication()` function that is a part of the `Terminal1` object.

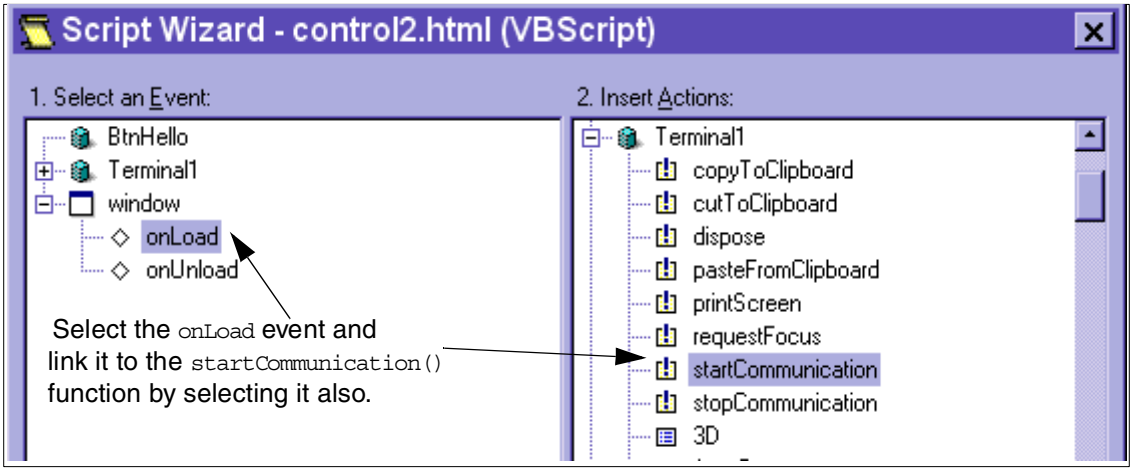


Figure 53. Selecting the onLoad event and linking it to the Terminal control's startCommunication() function

4. Click **Insert Action** to generate your VBScript code. Click **OK.**

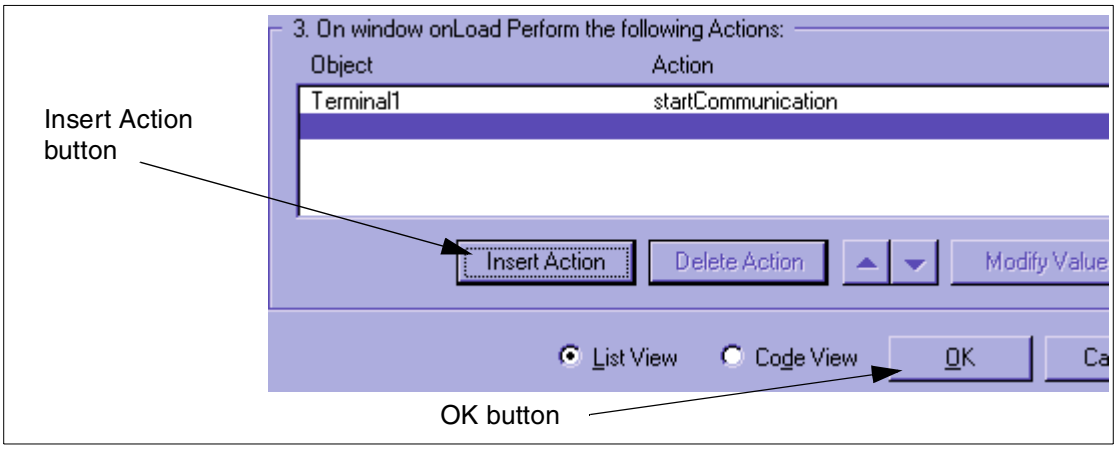


Figure 54. Finishing the Script Wizard dialog

If we save the HTML file now and load it with Internet Explorer, a Terminal screen should appear and be connected to the host you entered into the Host field earlier.

5.3.3 Controlling Host Access Controls for ActiveX with JavaScript

This section is not intended to be a tutorial on JavaScript. It briefly explains how to use JavaScript to interact with the ActiveX control.

A basic understanding of JavaScript is all one needs to interface with the ActiveX control. To use the ActiveX control, simply refer to the object's name and refer to the methods and properties like any other object. But where and what is the ActiveX control's name?

Remember that when one creates the HTML tag for an ActiveX control, they must include an ID parameter. The value of this parameter is the object's name within the Web browser. In the above example, the ID value was "Terminal1"; therefore, we can call any function or property of the terminal control by using the `Terminal1.`*`funtion()`* or `Terminal1.`*`property`* notation.

For purposes of example, add a button to the page that calls the `stopCommunication()` function of the Terminal object when it is clicked:

```
<FORM>
    <INPUT
     TYPE = "BUTTON"
     NAME = "DisconnectButton"
     VALUE = "Disconnect"
     onClick = "Terminal1.stopCommunication()">
</FORM>
```

As a result, a button that calls the `stopCommunication()` function when it is clicked is added to the Web page.

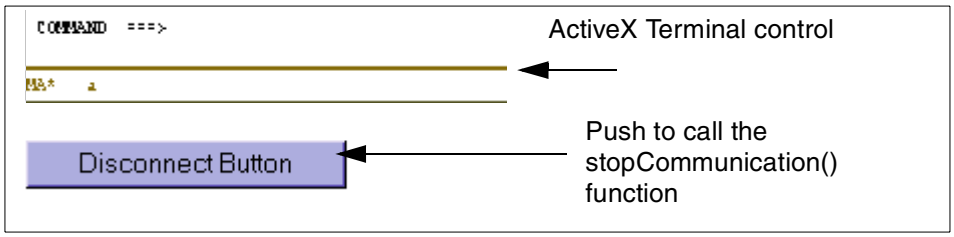


Figure 55. Disconnect button which calls the stopCommunication() function via JavaScript

5.4 Deployment over a network

Due to licensing issues, Personal Communications or Host On-Demand *must* be installed on the client system first. Then using your ActiveX-based Web pages over a network, such as the Internet, is simple. Ensure that your HTML file is accessible through a Web server and it should function as intended when a user downloads the page with Internet Explorer. There are no changes that need to be made to the file because Internet Explorer will read the Object tag and search for the controls on the local system. If they cannot be found, Internet Explorer will report that errors were found in the page.

5.5 Example: Em-You-Later

Both Personal Communications and Host On-Demand follow the same basic design for laying out the emulator's components such as the MacroManager and KeyPad. We would not suggest the design is a poor one; however, many users may want to customize the individual pieces of the emulator. Maybe user A wants the KeyPad above the Terminal and the MacroManager on the far left? Perhaps User B does not want to see the FileTransfer component at all.

When we sort through such opinions, we come to find out two things:

1) Not everybody is going to like how you visually designed your program, emulator or not.

2) It is probably best to just give users and developers pieces of the program so they can design it to their liking.

A program in which users can tailor the interface will greatly enhance the potential amount of productivity associated with that user. Enter our example, Em-You-Later.

Em-You-Later is a Web page that takes the visual Host Access Controls for ActiveX and presents them to the user. Of course, we decided to make an example more complex then just laying out five or so controls for you to interact with. By combining the power of JavaScript and VBScript, we developed a Web page that lets the user choose which components to incorporate into *their* customized emulator. Em-You-Later also uses scripting logic to set properties differently based upon user input.

Host Access Controls for ActiveX that we are going to use are:

- Terminal
- KeyPad
- MacroManager
- FileTransfer
- KeyRemap

Note: Session and Screen controls were not included because they are already integrated into the Terminal control. The Macro control was not added because of a lacking visual presence and because the MacroManager allows the user to create and execute macros. Nonetheless, these three controls are just as accessible as the five others.

5.5.1 Usage

A user executes Internet Explorer to launch Em-You-Later. Once the page is loaded, the user should see a table of ActiveX controls and a few associated properties. Figure 56 is a screenshot of Em-You-Later when it is first displayed.

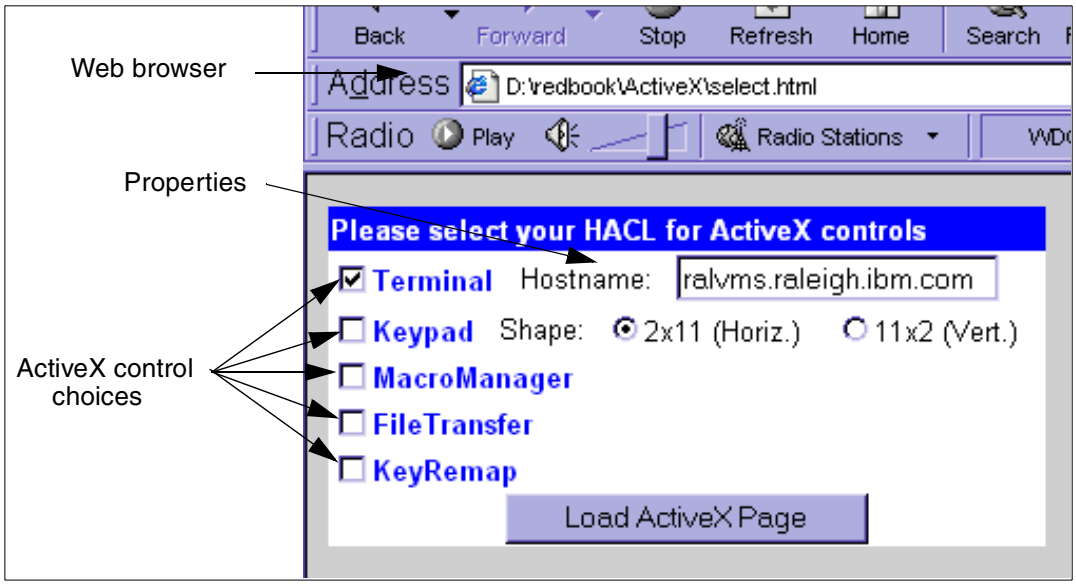


Figure 56. Em-You-Later immediately after execution

As you can see, each checkbox is associated with one of the previously listed ActiveX controls and we can prompt the user for any specific properties we may want/need them to define. The Terminal control is already selected because it is required for Em-You-Later to connect to a host. Un-checking the Terminal checkbox will result in a warning message.

Figure 57. Message box which appears if the user deselects Terminal from the list of ActiveX controls

Otherwise, the user can select the components he/she is interested in and set one of two properties to demonstrate being able to customize components even through a Web interface. The host name can be set inside the lone text field and the shape/positioning of the keypad, if it is selected, is also

configurable. Once the user has made a selection, they can click the **Load ActiveX Page** to have the program work its magic.

When the button is clicked, internal JavaScript routines generate a new HTML page. During this process, the components, such as checkboxes and radio buttons, share their values with the scripting language so that the page can be generated in accordance with the values the user submitted.

How do we do this? We use JavaScript to create a new variable, `fs`, and append strings that are characteristic of a Web page. After we have appended all the data needed for the new page, we write it out and close the current page. The open Web browser subsequently loads the new page that was just created. We have provided a piece of sample code below:

```
<script language=JavaScript>
{
     var fs = "";

     fs+='<HTML>\n'
     fs+='<HEAD>\n'
     fs+='<SCRI'
     fs+='PT LANGUAGE="VBScript">\n'

     ....

     fs+='</BODY>\n'
     fs+='</HTML>\n'
     document.write(fs);
     document.close();
}
</SCRIPT>
```

Figure 58. Sample code that generates a new HTML file in JavaScript

As you can see, `fs` is constantly being appended to. This (`fs`) is the resultant HTML file which loads into the Web browser when `document.close()` is executed. The HTML scripting tag:

```
<SCRIPT Language="VBScript">
```

has been split into two lines because Internet Explorer interprets the tag as being for the current page. When this happens, script errors result and Internet Explorer reports them to you. Breaking up the tag avoids this problem.

The blank section in the above code is where the bulk of the new page is constructed. Em-You-Later adds a component via a three-step process:

1. Check to see if the user selected the control.

2. Add functions for wiring.
3. Add the object reference tag.

5.5.2 Adding a control

If the value of a checkbox is checked (a value of true), then any functions for wiring controls will be immediately added to the new page. For example, if we wanted our new page to immediately connect the terminal to a host and be attached to any other controls, say a Keypad and MacroManager, the next few lines of our JavaScript would appear like this:

```
if (enableTerminal == true)
  {

    fs+='Sub window_onLoad()\n'
    fs+=' call Terminal1.startCommunication()\n'
    fs+='End sub\n'
    if (enableKeypad == true)
    {
      fs+='Sub KeyPad1_sendKeys(SendKeyEvent1)\n'
      fs+='call Terminal1.sendKeys(SendKeyEvent1)\n'
      fs+='end sub\n'
}
    if (enableMacroM == true)
    {
       fs+='Sub Terminal1_sendKeys(SendKeyEvent1)\n'
      fs+='call MacroManager1.sendKeys(SendKeyEvent1)\n'
      fs+='end sub\n'
    }
```

Figure 59. Code to wire components together for Em-You-Later

In essence, we are using JavaScripting techniques to lay out code that will be interpreted as VBScript. But now we must finish the process by actually defining the object in the new page. Using the techniques learned in 5.3.1, “Using the Control Pad to insert ActiveX controls into HTML” on page 81, we will manually insert the definition for the Terminal ActiveX control which was provided by Microsoft’s ActiveX Control Pad. To set any dynamic properties, simply break apart the parameter and insert a variable which reflects the correct value. See the highlighted lines of code below as an example.

```
function addTerminal()
{
  fs+='<OBJECT ID="Terminal1" WIDTH=400 HEIGHT=307\n'
  fs+='   CLASSID="CLSID:DC9B6B90-1A51-11D2-9AA3-08005AB9F957">\n'
   ....
  fs+='  <PARAM NAME="ServiceMgrHost" VALUE="">\n'
  fs+='  <PARAM NAME="Host" VALUE="'
  fs+=hostname
  fs+='">\n'
   ....
  fs+='  <PARAM NAME="LUMLicensing" VALUE="HOD">\n'
  fs+='  <PARAM NAME="SSLCertificateRemembered" VALUE="-1">\n'
  fs+='</OBJECT>\n'
}
```

Figure 60. JavaScript that defines an ActiveX control for a new HTML page

Of course, there is quite a bit of logic built into the page for enhanced usability. View the full HTML, select.html, on the included CD-ROM to see the file in its entirety.

5.5.3 The end result

Once you select your components and choose your assignable properties, your new Web page is generated, executed, and Em-You-Later displays its pieces. Bear in mind that this is just a simple example and other functionality can be built around this basic design. We could show you a screenshot of Em-You-Later with all of its components selected, but that would not be too different from using Host On-Demand or Personal Communications. Take a look at how the program looks in a few sample configurations.

With the Terminal, MacroManager, and FileTransfer controls, Em-You-Later looks like the figure below:

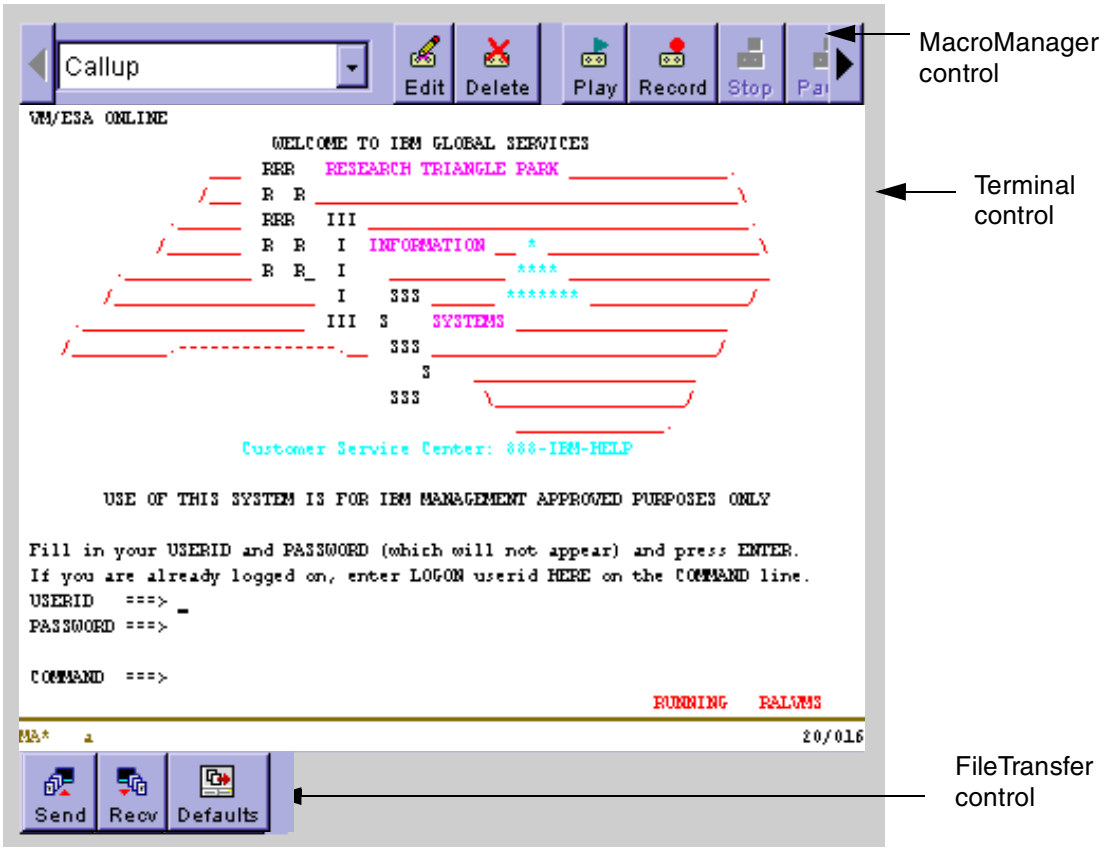


Figure 61. Em-You-Later with Terminal, MacroManager, and FileTransfer controls enabled

You may not need the functionality of FileTransfer or MacroManager controls. Perhaps just the KeyPad would suffice. Em-You-Later allows you to create a KeyPad control that is laid out either horizontally or vertically. How about completing your key-based emulator with the addition of the KeyRemap control? Select these components from the initial page and the result should match the following figure:

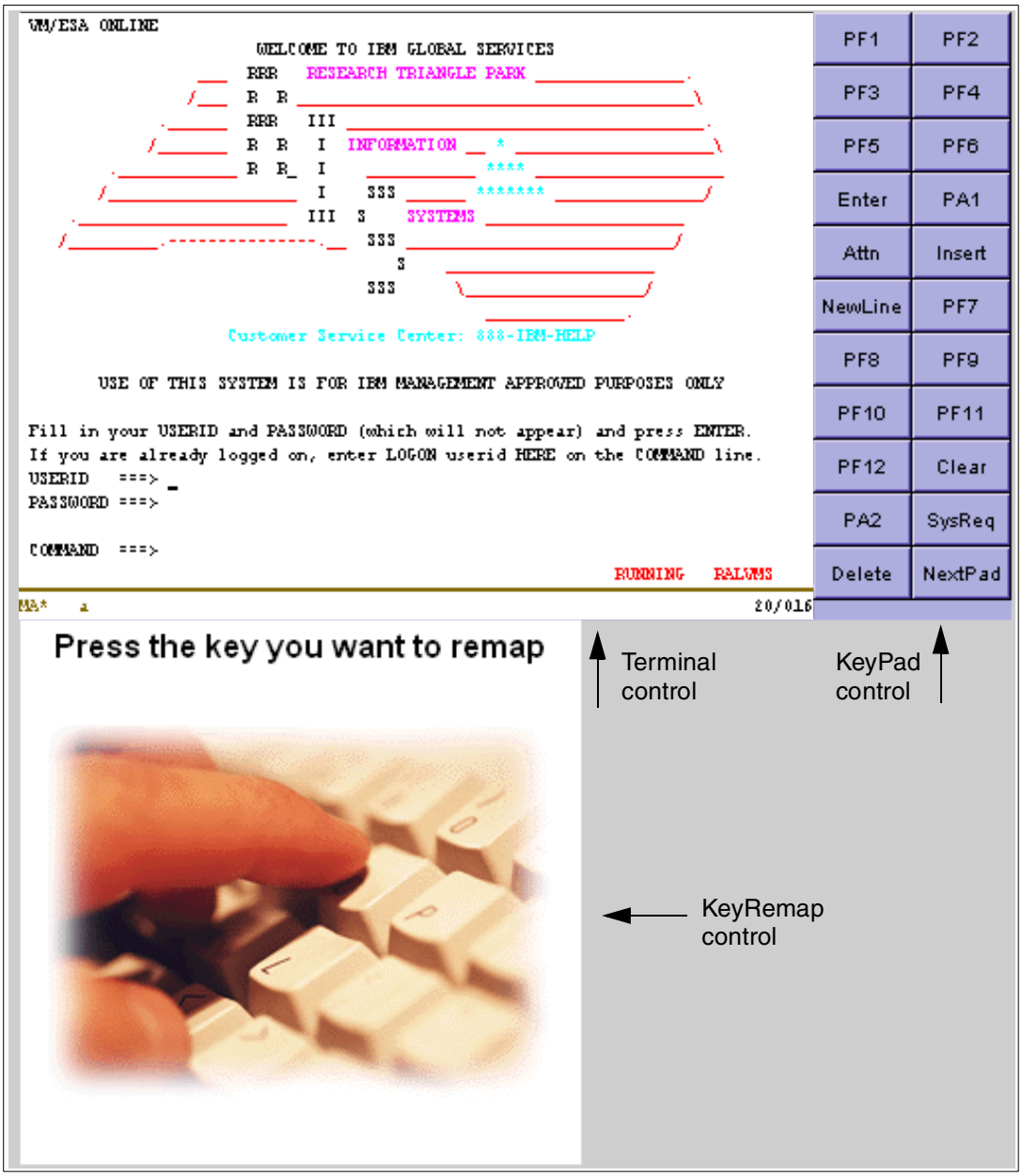


Figure 62. Em-You-Later with Terminal, KeyPad, and KeyRemap controls enabled

5.6 Summary

The Host Access Controls for ActiveX open up many new possibilities for host-based developers on the Win32 platform. While we discussed how to incorporate them into a Web browser, one could easily assimilate the controls into a word Pprocessor or accounting program. ActiveX also supports a wide range of Win32 development environments such as Visual Basic and Visual C++. If a Windows-specific approach is not what your solution needs, try one of the first three Java-based chapters.

5.7 Additional references

IBM suggests using the Host Access Beans documentation for the Host Access Controls for ActiveX because of their similarities. The Host Access Beans documentation is provided in HTML format with Personal Communications and Host On-Demand.

For Host On-Demand the documentation is in:

c:\Hostondemand\lib\en\doc\beans\beanReference.html

- For Personal Communications the documentation is in:

c:\Program Files\Personal Communications\doc\beans\beanReference.html

One can also view the following file with Host On-Demand:

c:\Hostondemand\lib\en\doc\beans\ActiveX.html

Chapter 6. HACL Automation Objects

Note: HACL Automation Objects are only available in Personal Communications.

The EHLLAPI language was established so that developers could write code to interface with emulators such as Personal Communications. In fact, many still do today. What if we could interface with an emulator by using another major application such as a word processor or spreadsheet software? What if we wanted to write a new application from scratch that could interface with the emulator?

Enter COM-based automation technology.

Developed by Microsoft and embedded in all Win32 platforms, *COM-based automation technology* allows a program to register its interfaces with the operating system. These interfaces, or *automation servers*, are used by *automation controllers*, such as Visual Basic, which allow programatic interactivity with the application. IBM supplies HACL Automation Objects as a means for Visual Basic programmers to use the knowledge they gained with EHLLAPI and develop applications without the hassles of a confusing single-entry-point interface.

Personal Communications contains ten automation servers called *Host Access Class Library Automation Objects*. They include:

- Connection List
- Connection Manager
- Field List
- Operator Information Area (OIA)
- Presentation Space
- Screen Description
- Screen Recognition
- Session
- Window Metrics
- File Transfer

This section will detail how to configure and write an application in Visual Basic 6.0 using the above HACL Automation objects. Similar techniques are applicable to other environments, such as Visual C++, that recognize COM Automation objects. For additional information, please refer to *Host Access Class Library*, SC31-8685, included with eNetwork Personal Communications V4.3 for Windows 95, Windows 98, and Windows NT.

6.1 Benefits and limitations

By using automation objects, a developer significantly reduces the amount of code needed for host-based communication. For example, there is no need to create or terminate the connection to a host because most, if not all, of the work is handled by Personal Communications's internal functions.

Automation objects themselves provide an interface for a variety of development environments so that *several* programming languages can be used to create host-based applications. Both Visual C++ and Visual Basic, for example, can use HACL Automation Objects.

Since programs, in essence, attach to Personal Communications with these objects, the emulator *must* be installed and configured for connection before the secondary application can work properly. The dependency on Personal Communications also suggests that your application should only be developed and executed on a Win32 platform such as Windows 95, 98, and NT.

6.2 Establish the environment

Setting up the environment in Visual Basic 6.0 is simple.

Execute Visual Basic 6.0 and select **Standard EXE** under the New tab in the New Project dialog. Click **Open**.

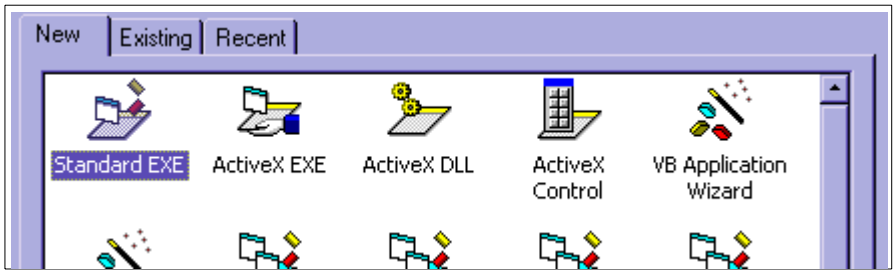


Figure 63. New project dialog with 'tandard EXE selected

At this point, a new project has been created with a blank form (window) positioned in the development space. By default, Visual Basic is not aware that Personal Communications has registered its automation servers with the operating system. For each new project, these objects will have to be specifically incorporated for use. Follow these simple instructions to load the appropriate automation controllers into your Visual Basic project:

1. Select **Project**.
2. Click **References**.
3. The References dialog should appear and display all of the available references that your projects may include.

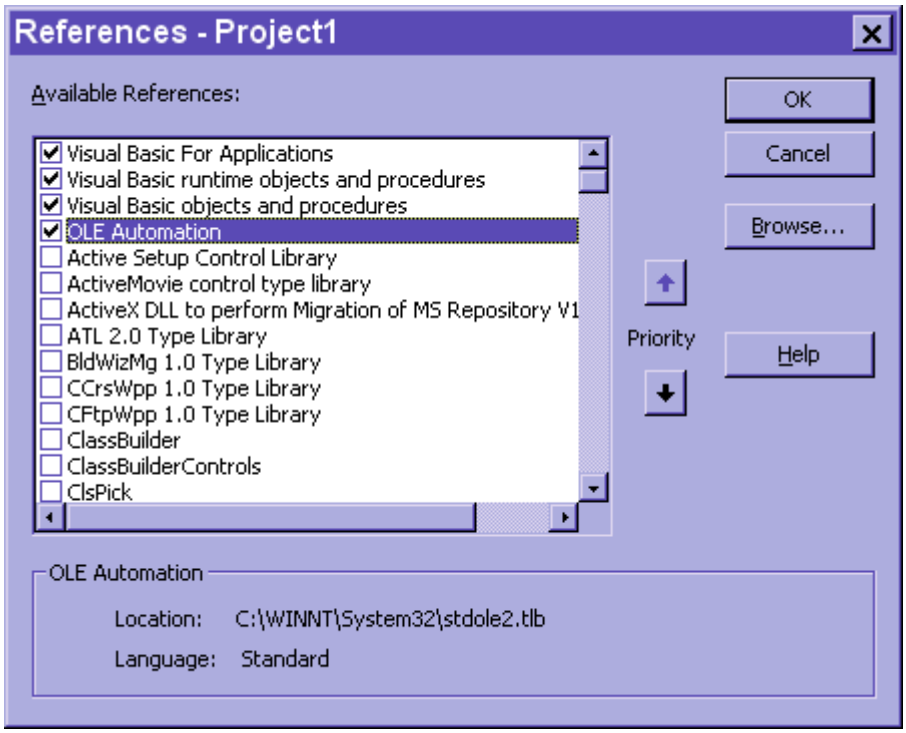


Figure 64. References dialog in Visual Basic 6.0

4. Check each box next to the following names:

   PCOMM autECLConnList Automation Object 1.0 Library
   PCOMM autECLConnMgr Automation Object 1.0 Library
   PCOMM autECLOIA Automation Object 1.0 Library
   PCOMM autECLPS Automation Object 1.0 Library
   PCOMM autECLScreenDesc Automation Object 1.0 Library
   PCOMM autECLScreenReco Automation Object 1.0 Library
   PCOMM autECLSession Automation Object 1.0 Library
   PCOMM autECLWinMetrics Automation Object 1.0 Library
   PCOMM autECLXfer Automation Object 1.0 Library

5. Click **OK**.

If the names above do not appear in the References dialog list or an error occurs when you click **OK**, reinstall your copy of Personal Communications. Uninstalling Personal Communications has been known to remove these automation objects and produce errors.

6.3 Getting started

HACL Automation Objects for Personal Communications provide similar functionality to the previously described HACL APIs; however, programs based on automation objects *must connect to an active Personal Communications program or start a configured session* to do any communication-based work. In other words, your program cannot be a stand-alone host-based application.

6.3.1 Making a module

To use the automation servers extended by Personal Communications, Visual Basic must create interface objects. These objects, if defined in the correct place, can be used at any location in your program and reduce the amount of code required by not having to reassign an object each time it is used. To accomplish this, use a Visual Basic module file.

Visual Basic uses module files to import functions, define global variables, and set global constants. Add a new module to your project by doing the following:

1. Make sure the Project Explorer is in view. Press Ctrl-R.

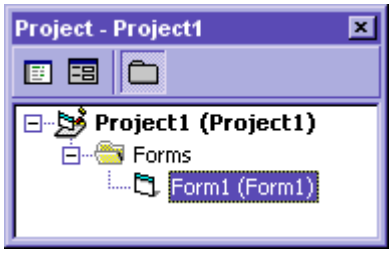


Figure 65. Project Explorer in Visual Basic 6.0

2. Right-click on the current form (Form1) and select **Add**. Click **Module**.

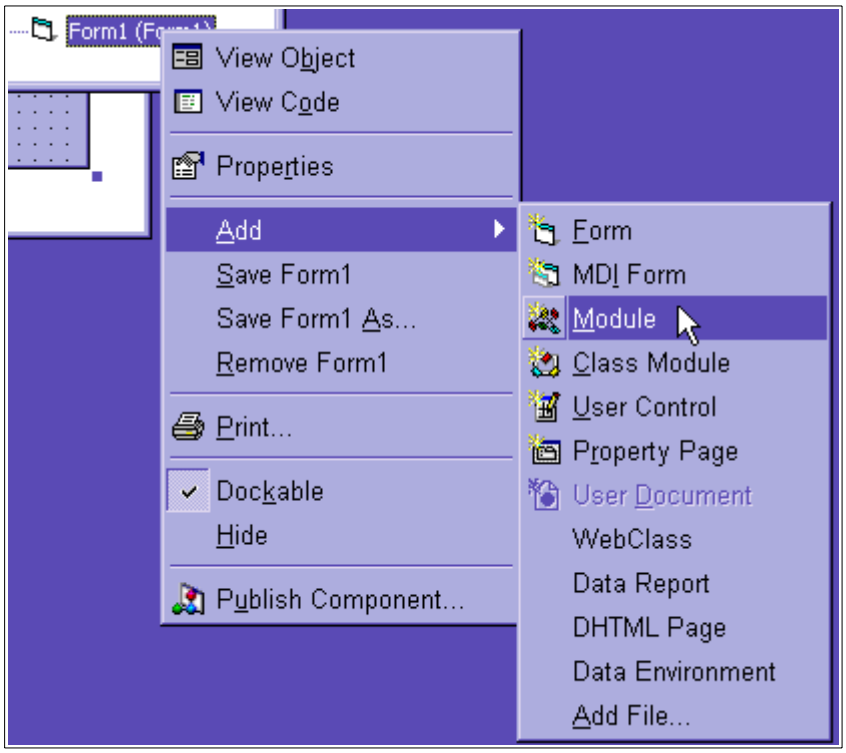


Figure 66. Adding a module file to the current project.

3. The Add Module dialog should now be showing. Select **Module** under the New tab and click **Open** to create and add the file to the project.

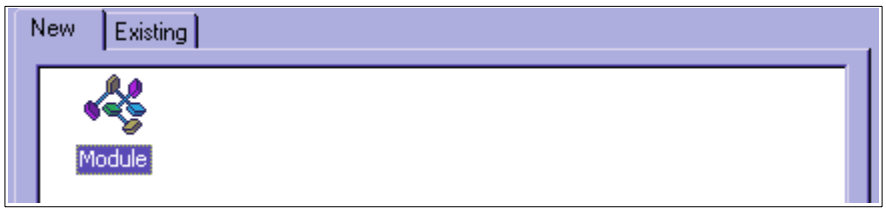


Figure 67. Add Module dialog with Module selected

Now that a module file has been successfully created, the project will need the definitions for the two *required* (for the purposes of this example) interface objects: AutPS and AutConnList. AutPS is the automation server included in Personal Communications that allows access to a selected session's presentation space. AutConnList provides a list of the current sessions being managed by Personal Communications.

4. If you have not already opened the window (step 3) double-click the module file listing which appeared in the Project Explorer to edit the contents. Since this is the first time the file has been edited, there will be no text within this window.

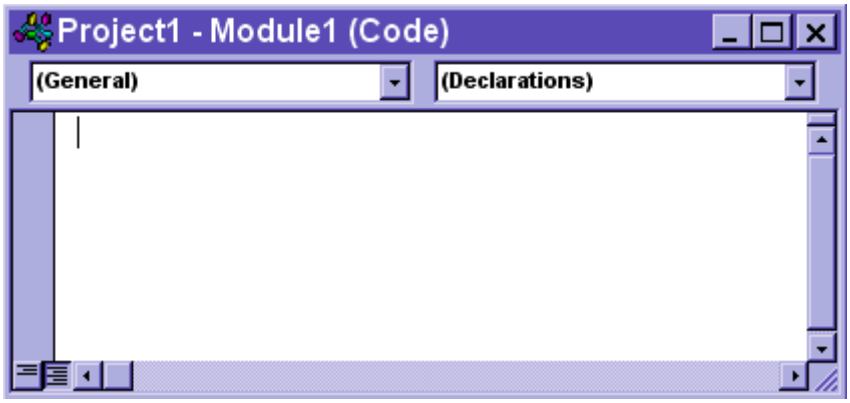


Figure 68. New module file being edited for the first time

5. The procedure to connect to a presentation space requires a Connection List object and a Presentation Space object. Place references to these objects by typing:

```
Global ps As AutPS
Global SessionList As AutConnList
```

   into your module file.

6. Double click on the form that was created when you first began this project. A window similar to that in the next figure should appear. The upper pull-down menus should read Form and Load.

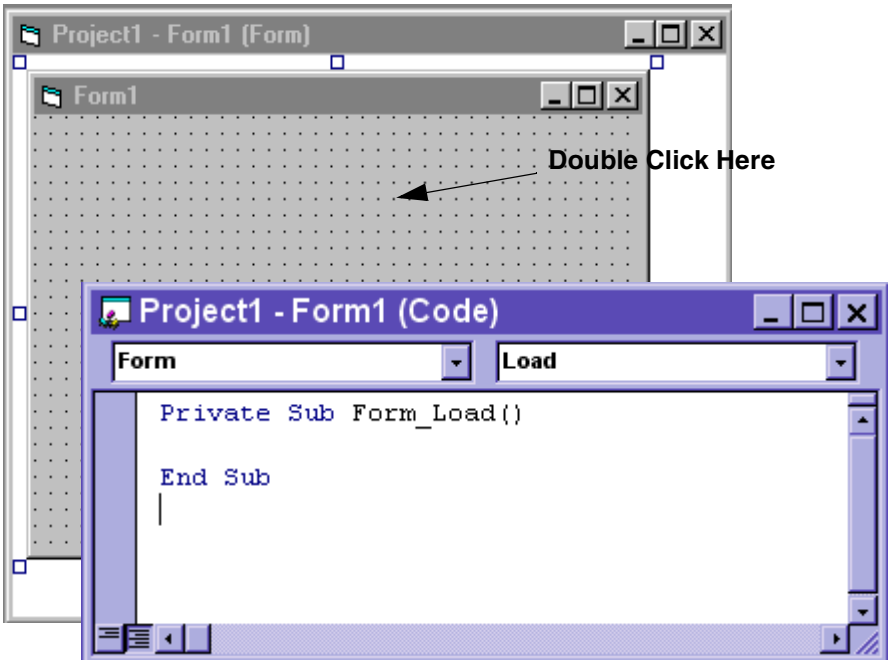


Figure 69. New form being edited for the first time

6.3.2 Creating the form

The next step is to begin designing your form (or window). We will not cover the details of how this is done in Visual Basic. The example we show later in 6.6, “Example: five key functions with HACL Automation Objects” on page 110 will give you an idea of what a form will look like when designed. In that example, buttons were added to the form, one for each function we will cover in the following sections.

The next sections will cover the basic programming techniques used in the later examples.

6.3.3 Creating the interface objects

The two references placed within the module file (step 2 in 6.3.1, “Making a module” on page 100) were just that - references. One must still tell Visual Basic that the references should point to or become interface objects. Since the automation servers (AutPS, AutConnList, etc.) were registered upon Personal Communications’s installation, you will need to call

`New(automation_server)` and Visual Basic will create an interface object linked to the associated automation server.

You do this with the following statements:

```
Set ps = New AutPS
Set SessionList = New AutConnList
```

Tip: Defining and creating an `AutSession` object would have created an interface object that represented an actual session and not just a presentation space.

This code can be typed in the `Form_Load` routine (shown in Figure 69 on page 103). Another option is to put these statements in the routine that connects to the session (our next step). In that case, the PCOMM session would have to be started before the connection was initiated, but not necessarily before the program was loaded.

Note: It is at the developer's discretion if any or all of this code should be executed at the same time and for which event. For example, it may be better to initiate communication with the host at startup, but not attach to Personal Communications until another event is triggered, such as the Get Text button in our later example, being pushed. The normal flow of the program should help determine when to execute the different functions and how. Feel free to use buttons, menus, and other useful objects that can trigger events.

The remaining functions in this chapter can be executed at any time given the following condition is met:

- The application is still connected to Personal Communications which has an active session open. Operating on a disconnected Personal Communications program can result in errors.

Note: If Personal Communications is not running, you will need to use a connection manager (autECLConnMgr) to execute PCOMM and subsequently perform your emulator-based commands.

6.3.4 Connecting to a presentation space

The final step in connecting to the presentation space is to get the list of sessions Personal Communications is managing and select one. In Visual Basic, `SessionList` is actually an array of sessions. The example below explains how to return the name of the first session in the array. The Presentation Space interface accepts this name, searches for a session with this handle, and selects it. The application can then attach itself to the selected session with the `SetConnectionByHandle()` function.

```
ps.SetConnectionByHandle(SessionList(1).Handle)
if (ps.Ready) Then
  'connected to Personal Communications and its session
else
'failed to connect to Personal Communications
 ps.StartCommunication
End if
```

A program should check the `ps.Ready` property to determine if communications with Personal Communications and a host were properly made.

Tip: The Connection List (SessionList) object includes a property which indicates the number of sessions it represents. If `SessionList.Count` equals 0, your program should use a Connection Manager. Otherwise, there should be no problem using `SetConnectionByHandle`.

This code can be placed anywhere within the project as long as it is executed *after* the interface objects have been created (see 6.3.3, “Creating the interface objects” on page 103).

6.3.5 Disconnecting from the presentation space

To disconnect, call the `StopCommunication()` function from the Presentation Space interface object:

```
ps.StopCommunication
```

To *reconnect* to Personal Communications, repeat the steps needed to create your interface objects, select an active session, and attach to Personal Communications. Note that `StartCommunication()` is executed asynchronously and the application may execute other instructions before the function has completed. If the `ps.Ready` property returns false, do one of two things:

- Wait and check the property x milliseconds later.
- Handle a failed connection attempt after n attempts.

Trying to access the properties of a Presentation Space object that has failed to connect will result in the program crashing. Please implement *some* program logic centered around the `ps.Ready` property. It will decrease any likelihood of your program encountering errors in its execution.

6.3.6 Searching for text

The PS object includes a `SearchText()` function which takes the following four arguments:

```
-String, The text for which you are searching for
-Integer, Identifies which direction to search (Optional)
       1 = Forward
       2 = Backward
-Integer, Row to begin searching (Optional)
-Integer, Column to begin searching (Optional)
```

For example, to search for "Hello" on row 3 at column 13 in a forward direction, one would enter:

```
if (ps.SearchText("Hello", 1, 3, 13) ) Then
    'found text
Else
    'didn't find text
End if
```

Note that `SearchText()` returns a boolean value. `TRUE` indicates the text was found and `FALSE` indicates that it was not.

6.3.7 Sending text

HACL Automation Objects provide a `SendKeys()` function with the PS object which accepts three parameters:

```
-String, The text with which you are sending
-Integer, Row to send text to (Optional)
-Integer, Column to send text to (Optional)
```

For example, if you needed to send the text "news" on row 10 at column 15, the `SendKeys` function would be called in this manner:

```
ps.SendKeys "news[enter]", 10, 13
```

The `[enter]` statement which appears in the above string may not be familiar to you. HACL supports key mnemonics - sending special keys using words familiar to the programming language. This feature makes for easier coding and readability. Refer to Appendix B, “SendKey() mnemonics” on page 127 for a complete list of key mnemonics supported by HACL.

6.3.8 Copying the presentation space to a string

Just like our previous examples, a program may need to pull text out of the presentation space. The application can accomplish this by executing the

`GetText()` function from the PS object which accepts the three parameters listed below:

```
-Integer, Row to read from (Optional)
-Integer, Column to read from (Optional)
-Integer, Number of characters to read in (Optional)
```

As an example, if the program needed to read the 25-character title of a new story starting with row 11 at column 13, one would execute:

```
psText = ps.GetText(11, 13, 25)
```

where `psText` was a string.

The HACL Automation Objects provide many more than five functions to the programmer. For a complete list of functions and how to use them, please refer to *eNetwork Personal Communications V4.3 for Windows 95, Windows 98, and Windows NT: Host Access Class Library*, SC31-8685.

6.4 Deployment

Deploying the application is easy considering Visual Studio provides a tool that does all of the work for you: the *Package & Deployment Wizard*. The following steps guide one through the wizard using mostly defaults. Feel free to experiment with the Wizard to customize the setup program more to your liking.

1. Launch the Package & Deployment Wizard from the Start/Programs menus.

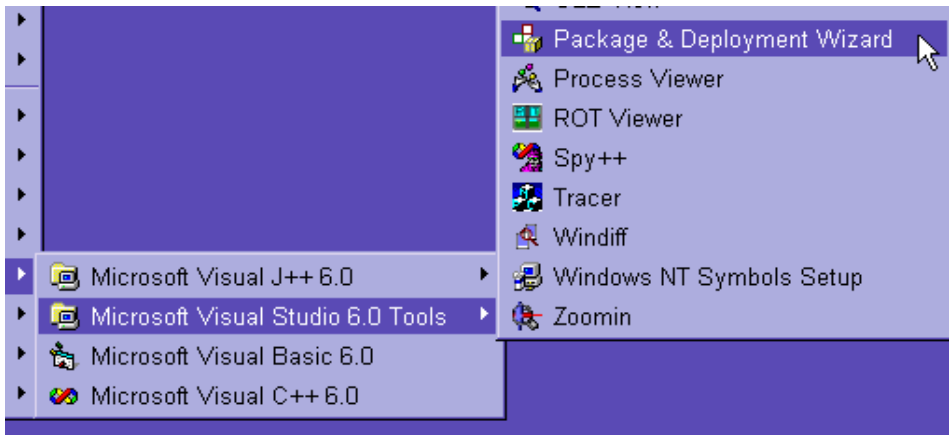


Figure 70. Selecting the Package & Deployment Wizard from Windows

2. Click the **Browse** button and an Open Project dialog will appear. Navigate to the directory where you saved your project and select the respective .vbp file.

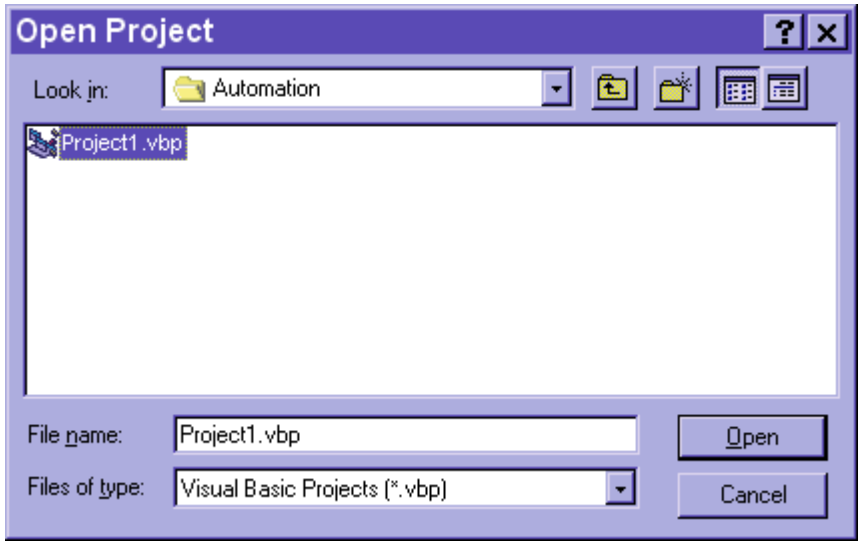


Figure 71. Open Project dialog when Browse'button is clicked in the Package & Deployment Wizard

3. Click the **Open** button and the full path of the project file will appear in the upper text field.
4. Next, you should notice three choices:
   - Package
   - Deploy
   - Manage Scripts

   Select the Package icon.

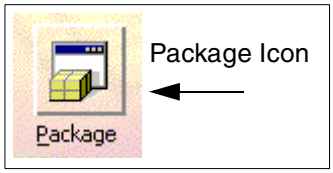


Figure 72. Package icon

5. A message box may appear and alert you that the wizard needs to find the executable (EXE) file. Select **Compile** to build it automatically.

6. The Wizard will then ask you what type of package to create. Select **Standard Setup Package** from the list and click **Next**. Unless you have a specific directory set aside for building setup programs, click **Next** again to accept the default directory.
7. A Missing Dialog Dependency dialog may appear and ask you to indicate which files have no dependencies. There is no need to add any dependencies because Personal Communications will have to be installed on the other computer, guaranteeing the automation objects will be installed as well. Click **OK** to continue.
8. Any files that the setup program will include in its installation will be displayed on the following screen. Uncheck any files that have the extension .tlb next to them. Again, these files will already have been installed on the target machine. Click **Next** to continue.
9. Choose how large you want your CAB file(s) to be. If you are not using any diskettes, the Single CAB selection should suffice. Again, click **Next** to continue.
10. The next window will ask you for an installation title. The text you enter into the field will be prominently displayed at the top of the setup program. Enter your title and click **Next** to move on.
11. The wizard will display the projected program icon and program folder. By default, the setup program will create a separate folder and icon under the Windows Start menu. If you want to change the names or add more icons for other files, this window will let you make those changes. Otherwise, click **Next** to continue.
12. The next two windows ask questions concerning the exact file placement and exclusive access to certain files. For the purpose of this project, click **Next** on each window to continue.
13. The final question from the Package & Deployment Wizard asks you to supply a name for the configuration you just supplied. Choose something specific or leave the default as it is. When the wizard is executed again, you will be able to load all of your packaging settings under this name. Click **Finish** to create your setup program.
14. If the program succeeded, a window should appear and notify you that the CAB file was placed inside a particular directory. This directory also contains setup.exe and setup.lst, two files needed to install the remainder of your program. Distribute these three files as you see fit.

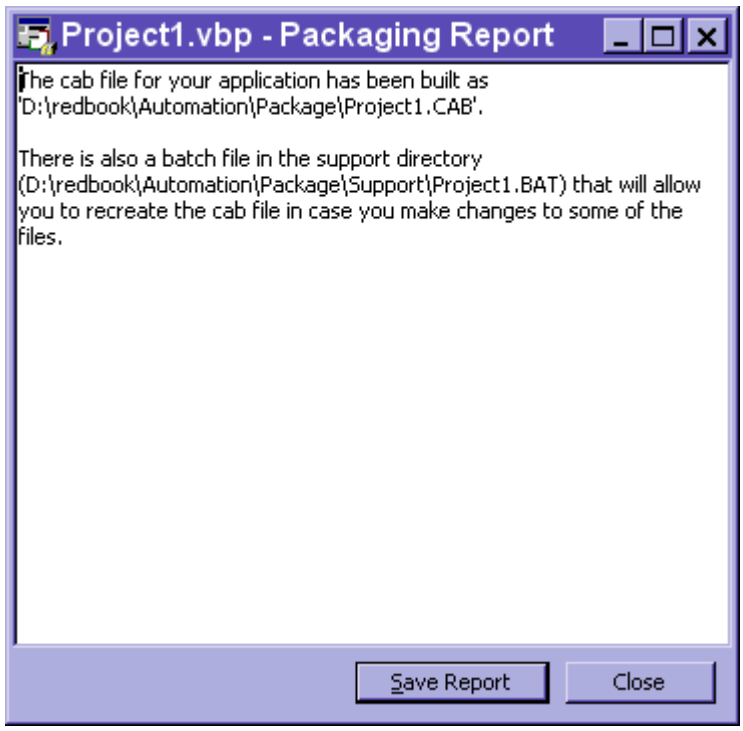


Figure 73. Completion window displayed by Packaging & Deployment Wizard

6.5 Testing

Now that your application is bundled into an installation program, copy the three files (CAB file, setup.exe, setup.lst) to a second computer. Run setup.exe and install the application. An icon should be placed in the program group made just for your program. If Personal Communications is running and attached to a host, the sample program should attach itself properly upon clicking the Connect button. If an error occurs, try re-installing Personal Communications. It is very likely the automation objects were uninstalled at some point.

6.6 Example: five key functions with HACL Automation Objects

The HACL Automation Objects example is written and compiled with Visual Basic 6.0. It takes the five previously discussed functions and executes them

with the appropriate user input. Below is a screen shot of the initial graphical user interface:

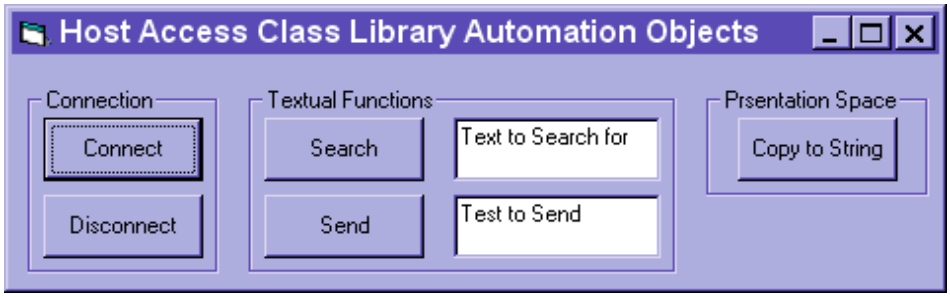


Figure 74. HACL Automation Objects example using Visual Basic 6.0

6.6.0.1 Connect button

Clicking the Connect button will execute the `ps.StartCommunication()` function. Similar to the last example, we did not implement a connection manager so PCOMM must be running. A message box (MsgBox) will reflect its success or lack thereof.

Figure 75. Message box displayed when StartCommunication() is successful

6.6.0.2 Disconnect button

Clicking on the Disconnect button will call the `ps.StopCommunication()` function. If the call is successful, Personal Communications will be disconnected from the host.

6.6.0.3 Search button

The Search button will search the entire presentation space for the string in the adjacent text box. Again, message boxes will be used to convey success or lack thereof.

Figure 76. Message Box displayed when SearchText() finds the string

6.6.0.4 Send button

Pushing the Send button will take the text in the adjacent text box and call the `ps.SendKeys()` function. The text will be sent to where the cursor is located in the presentation space.

6.6.0.5 Copy to String button

The fifth and final button copies the presentation space's text and displays it using a message box.

6.6.1 Source code

The example only passes the required parameters (if any) to each function to make the example work on a variety of systems. Feel free to add any optional parameters to test with your development environment and hosts. The example contains five buttons and two text boxes. Each component has its name and initial text/caption provided in the following table:

Table 3. Component name and initial text/caption for HACL Automation Objects example

Component Name	Initial Text/Caption
connect_Button	Connect
disconnect_Button	Disconnect
search_Button	Search
send_Button	Send
copy_Button	Copy to String
search_TB	Text to Search for
send_TB	Text to Send

6.6.1.1 Visual Basic form source code

```
Private Sub Connect_Click()
    Set ps = New AutPS
    Set SessionList = New AutConnList
    ps.SetConnectionByHandle (SessionList(1).Handle)
    If (ps.Ready) Then
        'connected to PCOMM and its session'
    Else
        'failed to connect to PCOMM'
         ps.StartCommunication
    End If
End Sub

Private Sub Copy_Click()
     If (ps.Ready) Then
        MsgBox ps.GetText
     End If
End Sub

Private Sub Disconnect_Click()
    If (ps.Ready) Then
        ps.StopCommunication
    End If
End Sub

Private Sub Search_Click()
    If (ps.Ready) Then
        If (ps.SearchText(search_TB.Text)) Then
            MsgBox "Text was found.", vbInformation
        Else
            MsgBox "Text was not found.", vbCritical
        End If
    End If
End Sub

Private Sub Send_Click()
    If (ps.Ready) Then
        ps.SendKeys (Send_TB.Text)
    End If
End Sub
```

Figure 77. Visual Basic source code for automation objects example - main file (.FRM)

6.6.1.2 Visual Basic module file

```
Global ps As AutPS
Global SessionList As AutConnList
```

Figure 78. Visual Basic source code for automation objects example - module file (.BAS)

The source code and all associated files are located on the CD-ROM included in the back of this redbook.

6.7 Example: Excel worksheet submission tool

Automation objects make bridging the gap between large-scale applications and Visual Basic very easy. Initially, when we were thinking of a scenario where a user needed to transmit data from an application into Personal Communications, we were stumped. But then we realized that many accountants still submit their work through a host system. Why not use the HACL Automation Objects to submit data from a Microsoft Excel spreadsheet to some central location?

The Excel worksheet submission tool takes important fields from a worksheet and sends their values to a host. We can accomplish this using the exact same techniques previously described because Excel uses Visual Basic for Applications (VBA) as its internal programming language. VBA is a scaled-down, yet very powerful, flavor of the Visual Basic programming language.

6.7.1 Usage

An Excel file (PCOMM.xls) has been created that contains accounting information and a Submit button. When the user loads the file (and allows macros to be enabled), a worksheet similar to Figure 79 will appear.

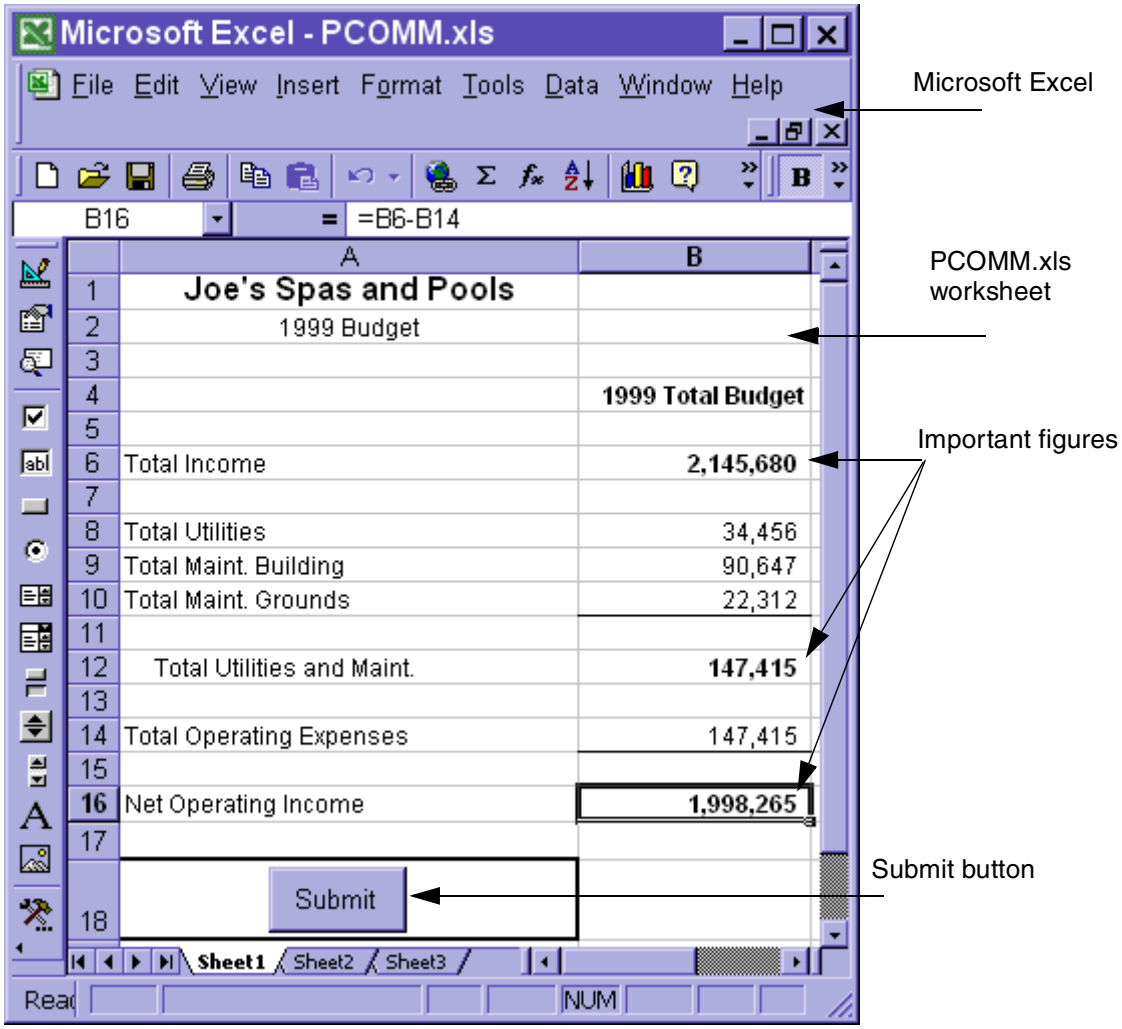


Figure 79. PCOMM.xls in Microsoft Excel

This is not the submission tool itself, but a worksheet which encompasses the VBA code and sample data for the proof of concept. Clicking **Submit** executes the real magic.

The sample worksheet details a general worksheet type. For our hypothetical situation, our accountant needs to submit the bold values in the supposed spreadsheet. These values represent:

- Total Income, (in position B6)
- Total Utilities and Maint., (in position B12*)*

- Net Operating Income, (in position B16)

When the user clicks the Submit button, Excel connects (for this example) to the active Personal Communications session and executes the necessary commands to essentially navigate through the host and populate the respective fields. Take a look at the VBA code that does all of the work. Remarkably, it is almost identical to the code we used earlier. Notice that the interface objects are defined local to the function instead of globally.

```
Private Sub CommandButton1_Click()
    Dim row1, row2, row3 As Integer
    Dim col1, col2, col3 As Integer

    Set ps = New AutPS
     Set SessionList = New AutConnList

    On Error GoTo ErrorHandler
    ps.SetConnectionByHandle (SessionList(1).Handle)
    If (ps.Ready) Then
        'Login and navigate to the correct screen
        '...

        'Send the Total Income Value
          With Worksheets("Sheet1").Cells(6, 2)
           ps.SendKeys .Value, row1, col1
          End With
        'Send the Total Utilities and Maint. Value
          With Worksheets("Sheet1").Cells(12, 2)
           ps.SendKeys .Value, row2, col2
        End With
        'Send the Net Operating Income
          With Worksheets("Sheet1").Cells(16, 2)
           ps.SendKeys .Value, row3, col3
          End With

         'Logoff
         '...

        MsgBox "The information has been submitted to ralvms.raleigh.ibm.com."
        Exit Sub
    Else
        MsgBox "HACL Automation objects could not attach themselves."
        Exit Sub
    End If

ErrorHandler:
    MsgBox "An error occurred within the program."
End Sub
```

Figure 80. VBA code to use the HACL Automation Objects with Microsoft Excel

The only visual confirmation Excel uses to indicate that the function has completed is the following:

Figure 81. Message box displayed when the HACL Automation Objects' function completes

6.7.2 Preparing Excel

You can re-create this example yourself (minus the code) by following just a few simple steps.

1. Select **Tools**.
2. Click **Macro** -----> **Visual Basic Editor...**

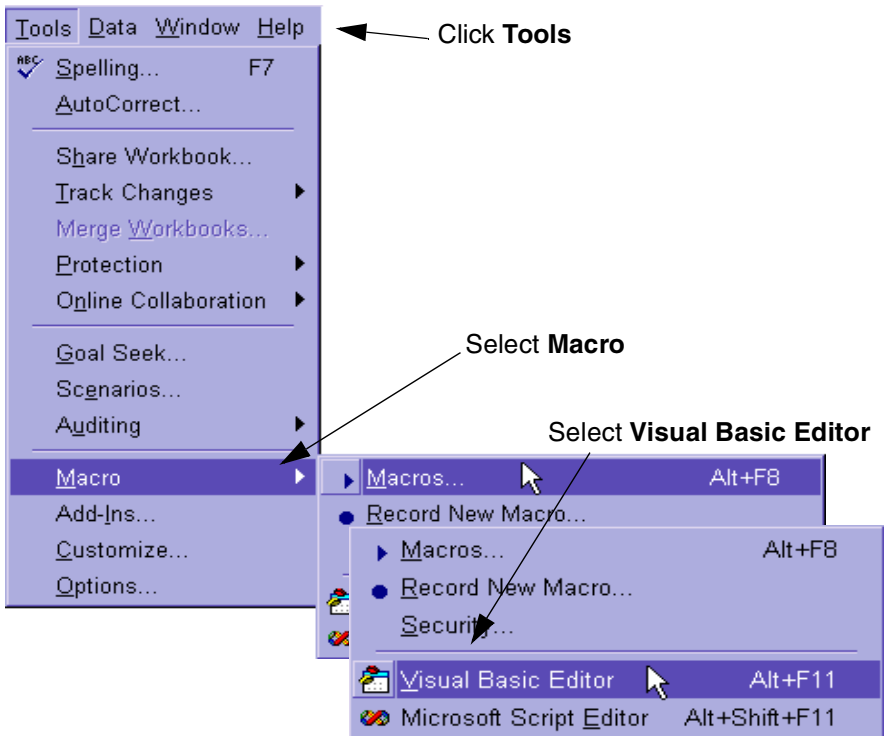


Figure 82. Selecting the Visual Basic Editor from Excel

Selecting the Visual Basic Editor will execute a separate program that is tied to Microsoft Excel. From this window:

3. Click **Tools** and select **References....**

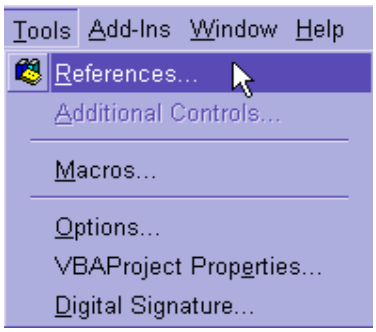


Figure 83. Selecting the References... option from the Visual Basic Editor

A familiar window (Figure 84) should now be seen. This is the same window you used earlier to insert the HACL Automation Objects into your Visual Basic project. Select each of the references that begin with "PCOMM aut" by placing a check in their corresponding checkbox.

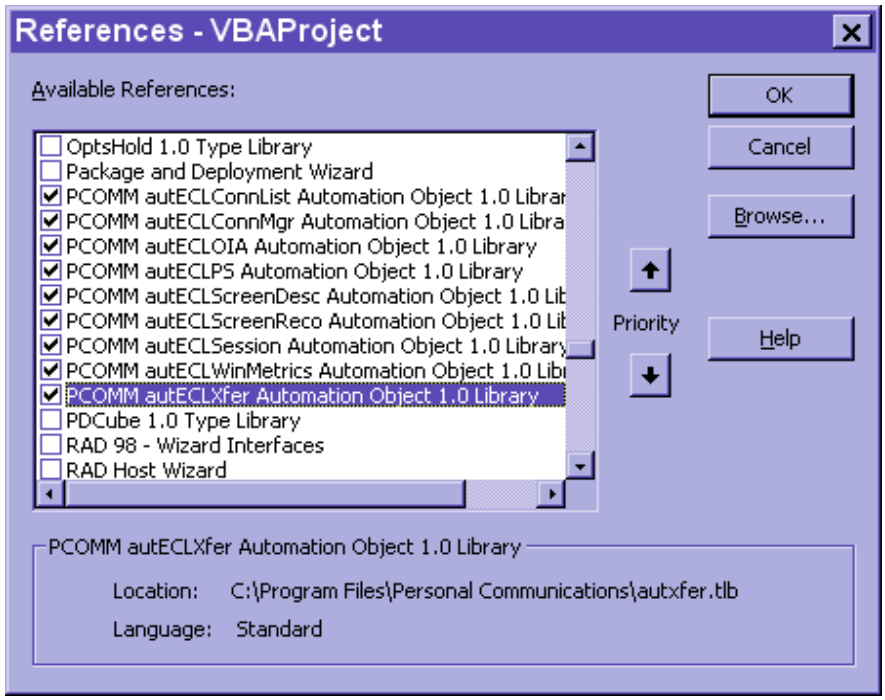


Figure 84. Inserting the HACL Automation Object into Excel

All that is left for you to do is add a button to the worksheet that executes the code required to connect to Personal Communications and interact with the host. Bring the Excel window back to the foreground and follow these few steps to add a button that will then let you develop code based upon these particular automation objects.

4. Click **View**.
5. Select **Toolbars ----> Control Toolbox** if it does not already have a check beside it.

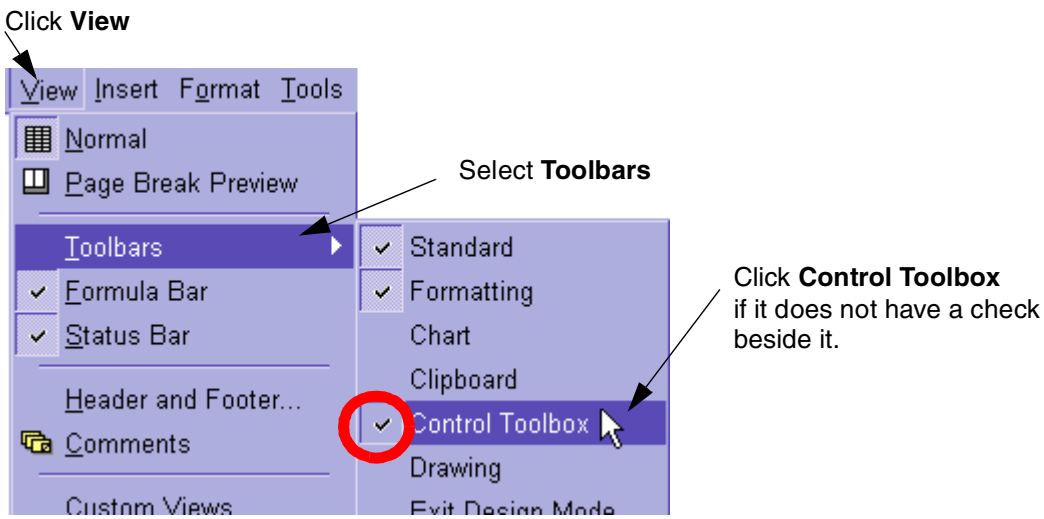


Figure 85. Adding the Control Toolbox in Excel

A toolbar like the one shown below will appear somewhere inside the Excel application, most likely at the top of the application. Select the icon that most resembles a standard button. The cursor should now change into crosshairs when placed over the worksheet. Click inside the worksheet to add a button.

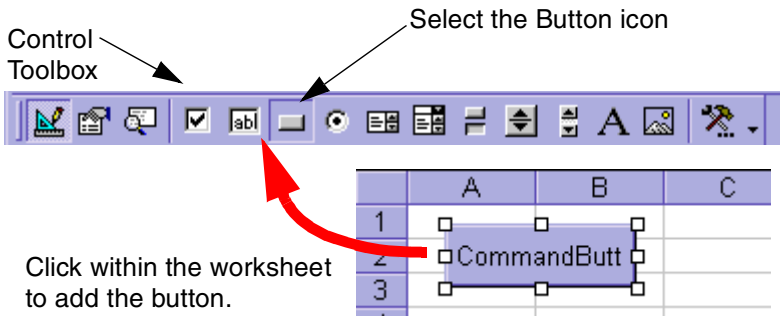


Figure 86. Adding a button to the existing worksheet with the control toolbar

6. Right-click on the button and select **View Code.** The Visual Basic Editor will be brought back to the forefront and will be ready for you to code any actions the button should take when it is clicked. This is where we inserted our code to connect to a host for the submission tool. You may want to do something similar.

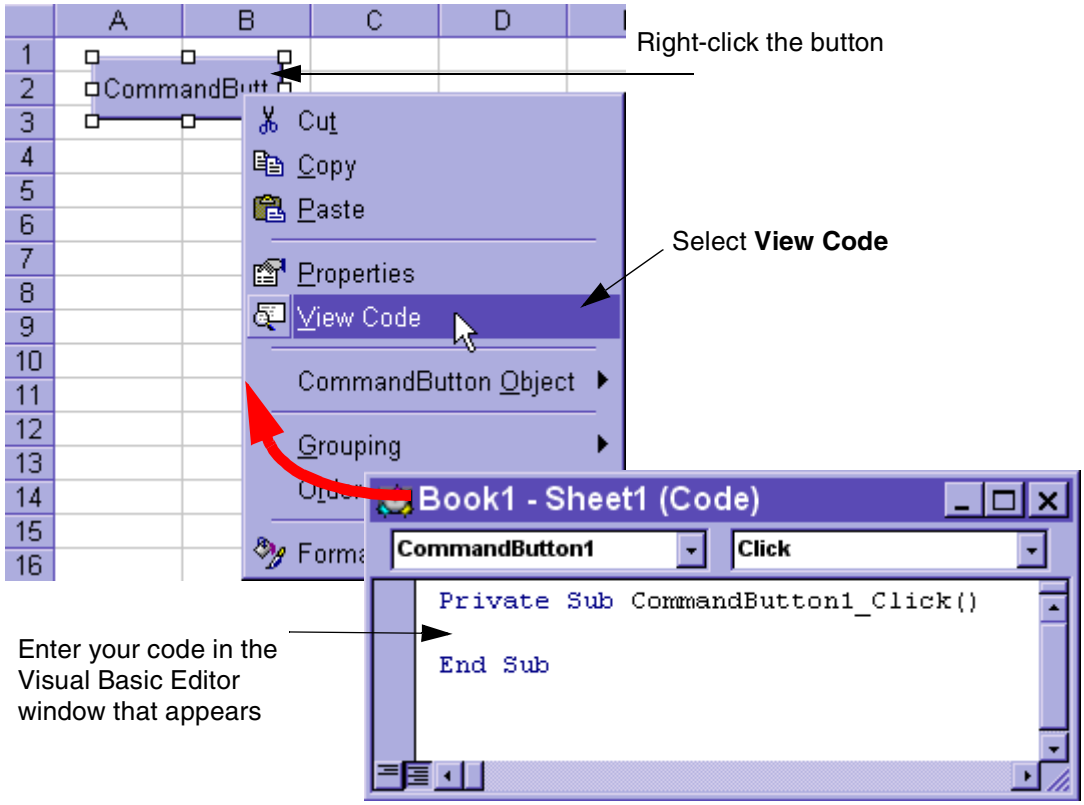


Figure 87. Editing the button's code

6.8 Summary

HACL Automation Objects provide a lightweight bridge between applications and the emulator. A developer who wants to transfer data between an application and the host only needs to spend a trivial amount of time implementing the HACL Automation Objects. There is no need to worry about communication issues such as 3270 datastream format because IBM has already done the work for you. Automation objects, in particular, make host-based communication even easier if your requirements include running Personal Communications as a primary or secondary application. Look at the worksheet submission tool itself. We were able to connect Microsoft Excel to a host in a matter of minutes. Imagine what you could do if the code was *tightly* integrated into your program.

6.9 Additional References

- *Personal Communications Version 4.3 for Windows 95, 98, and NT,* SG24-4689
- *IBM SecureWay Host On-Demand: Enterprise Communications in the Era of Network Computing*, SG24-2149-00
- *eNetwork Personal Communications V4.3 for Windows 95, Windows 98, and Windows NT: Host Access Class Library*, SC31-8685

Appendix A. Signing applets

Because of concerns over security, applets can only do so much by default. In a Web server environment, the user must grant the applet permission to proceed with its intent. Unfortunately, communicating with a host other than where the applet was downloaded from is one such action that must be approved. To ensure that applets operate securely and are trustworthy, both Netscape and Microsoft have implemented security managers that require each applet to be authenticated before it asks for permission. The following is a description of how to authenticate for both Internet Explorer, which uses CAB files, and Netscape Navigator, which uses JAR files.

A.1 How to sign CAB files for Internet Explorer

To create and sign a CAB file, download the Microsoft SDK for Java 3.2 from the following Web address:

```
http://www.microsoft.com/java/
```

Once installed, use the following commands to sign your class file(s) as one signed CAB file.

1) Create a test key named TestKey.pvk, incorporate the author's name, and create a test certificate as test.cer:

```
makecert  -sv TestKey.pvk -n "CN=Your Name" test.cer
```

2) Convert the test certificate into a public key:

```
cert2spc test.cer test.spc
```

3) Create the CAB file:

```
cabarc -p -c -s 6144 N test.cab (list class files)
```

4) Sign the CAB File with TestKey, public key:

```
signcode -j javasign.dll -jp low -spc test.spc -v TestKey.pvk -n
    "Program Name" -i http://yourwebaddress test.cab
```

A.2 How to sign JAR files for Netscape Navigator

To create and sign a JAR file, download the Netscape Signtool utility from the following Web address:

```
http://developer.netscape.com/software/signedobj/jarpack.html#signtool1.1
```

1) Open Netscape Navigator and click on the Security icon. Click **Passwords** and **Change Password.** Enter a password and confirm it to set a password on the certificate repository.

2) Create a test certificate and import it directly into the Netscape Certificate Repository. Substitute the path for one appropriate to your Navigator User directory. It should be similar to the one below:

```
signtool -G CertificateName -d "C:\Program Files\Netscape\Users\default"
```

3) Call `signtool` to sign and create your JAR file where:

```
A = the name of the certificate used in Step 2.
B = the name of the JAR file with the .jar extension
C = the name of the subdirectory which contains the class file(s) to be
      inserted into the JAR file

signtool -k A -d "C:\Program Files\Netscape\Users\default" -Z "B" C
```

Note: Don't forget to change the path to reflect the correct Netscape User directory.

A.3 Obtaining real certificates

A browser distinguishes between running an applet locally or across the network. In either case, the browser's internal security manager handles excessive access requests, such as writing to files, differently. Generating test certificates is sufficient to run your Java applets on a local system, but you will need to purchase real certificates when you are ready to deploy your program.

Presently, there are two Certificate Authorities (CAs) which will issue real certificates once they have verified your identity and you have paid the associated fee. We have listed those company names, their Web addresses, and the type of certificate a company would need to purchase from each site:

- VeriSign, Inc. at `http://www.verisign.com/`, *Commercial Class 3 Certificate*
- Thawte Consulting. `at http://www.thawte.com/`, *Developer Certificate*

Note: Because Internet Explorer uses Microsoft's Authenticode technology and Navigator incorporates Netscape's Object Signing technology, you will need to purchase a different certificate for each of these browsers. Fortunately, these certificates can be used for other purposes such as signing executable downloads and authenticating plug-ins.

Appendix B. SendKey() mnemonics

The following table lists each mnemonic keyword that the `SendKeys()` function will recognize for each session type. Special information is denoted by any adjacent superscripts.

To use these mnemonics, include each keyword as part of the `SendKeys()` string parameter, for example:

```
ECLPS.SendKeys("Hello[enter]")
```

would send the word "Hello" followed by the Enter key.

Table 4. SendKeys() mnemonics table

Function	Mnemonic Keyword	3270	5250	VT	CICS
Attention	[attn]	X	X		X
Alternate View	[altview]	X[3,4]	X[3,4]		X[3,4]
Backspace	[backspace]	X	X	X[1]	X
Backtab	[backtab]	X	X		X
Beginning of Field	[bof]	X	X		X
Clear	[clear]	X	X	X[1]	X
Cursor Down	[down]	X	X	X[1]	X
Cursor Left	[left]	X	X	X[1]	X
Cursor Right	[right]	X	X	X[1]	X
Cursor Select	[cursel]	X	X	X[1]	X
Cursor Up	[up]	X	X	X[1]	X
Delete Character	[delete]	X	X	X[1,2]	X
Display SO/SI	[dspsosi]	X[3,4]	X[3,4]		X[3,4]
DUP Field	[dup]	X	X		X
Enter	[enter]	X	X	X	X
End of Field	[eof]	X	X	X[1,2]	X
Erase EOF	[eraseeof]	X	X		X

Function	Mnemonic Keyword	3270	5250	VT	CICS
Erase Field	[erasefld]	X	X		X
Erase Input	[erinp]	X	X		X
Field Exit	[fldext]		X		
Field Mark	[fieldmark]	X	X		
Field Minus	[field-]		X		
Field Plus	[field+]		X		
F1	[pf1]	X	X	X	X
F2	[pf2]	X	X	X	X
F3	[pf3]	X	X	X	X
F4	[pf4]	X	X	X	X
F5	[pf5]	X	X	X	X
F6	[pf6]	X	X	X	X
F7	[pf7]	X	X	X	X
F8	[pf8]	X	X	X	X
F9	[pf9]	X	X	X	X
F10	[pf10]	X	X	X	X
F11	[pf11]	X	X	X	X
F12	[pf12]	X	X	X	X
F13	[pf13]	X	X	X	X
F14	[pf14]	X	X	X	X
F15	[pf15]	X	X	X	X
F16	[pf16]	X	X	X	X
F17	[pf17]	X	X	X	X
F18	[pf18]	X	X	X	X
F19	[pf19]	X	X	X	X
F20	[pf20]	X	X	X	X
F21	[pf21]	X	X		X

Function	Mnemonic Keyword	3270	5250	VT	CICS
F22	[pf22]	X	X		X
F23	[pf23]	X	X		X
F24	[pf24]	X	X		X
Help	[help]		X		
Home	[home]	X	X	X [1,2]	X
Insert	[insert]	X	X	X [1,2]	X
Keypad 0	[keypad0]			X	
Keypad 1	[keypad1]			X	
Keypad 2	[keypad2]			X	
Keypad 3	[keypad3]			X	
Keypad 4	[keypad4]			X	
Keypad 5	[keypad5]			X	
Keypad 6	[keypad6]			X	
Keypad 7	[keypad7]			X	
Keypad 8	[keypad8]			X	
Keypad 9	[keypad9]			X	
Keypad Dot	[keypad.]			X	
Keypad Enter	[keypadenter]			X	
Keypad Comma	[keypad,]			X	
Keypad Minus	[keypad-]			X	
New Line	[newline]	X	X		X
PA1	[pa1]	X	X		X
PA2	[pa2]	X	X		X
PA3	[pa3]	X	X		X
Page Up	[pageup]	X	X	X [1,2]	X
Page Down	[pagedown]	X	X	X [1,2]	X
Reset	[reset]	X	X	X	X

Function	Mnemonic Keyword	3270	5250	VT	CICS
System Request	[sysreq]	X	X		X
Tab Field	[tab]	X	X	X [1]	X
Test Request	[test]		X		

1 = VT supports this function, but the host application must act upon it.
2 = Only supported for VT200 mode.
3 = For DBCS sessions only.
4 = For Host On-Demand only.

Appendix C. Complete screenshots for HACL examples

The following pages are screenshots of each example provided in this document. These images were too large to display inside their respective chapters so we have dedicated this space for them. Below is a table of each screenshot's program name, page number, and the page number for each section detailed in the book.

Table 5. HACL example screenshot listing

Program Name	Screenshot Page Number	Discussion Page Number
TOTALS Applet	132	38
SessionAssist!	133	58
Double Time	134	72
Em-You-Later	135	87
Microsoft Excel Worksheet Submission Tool	136	114

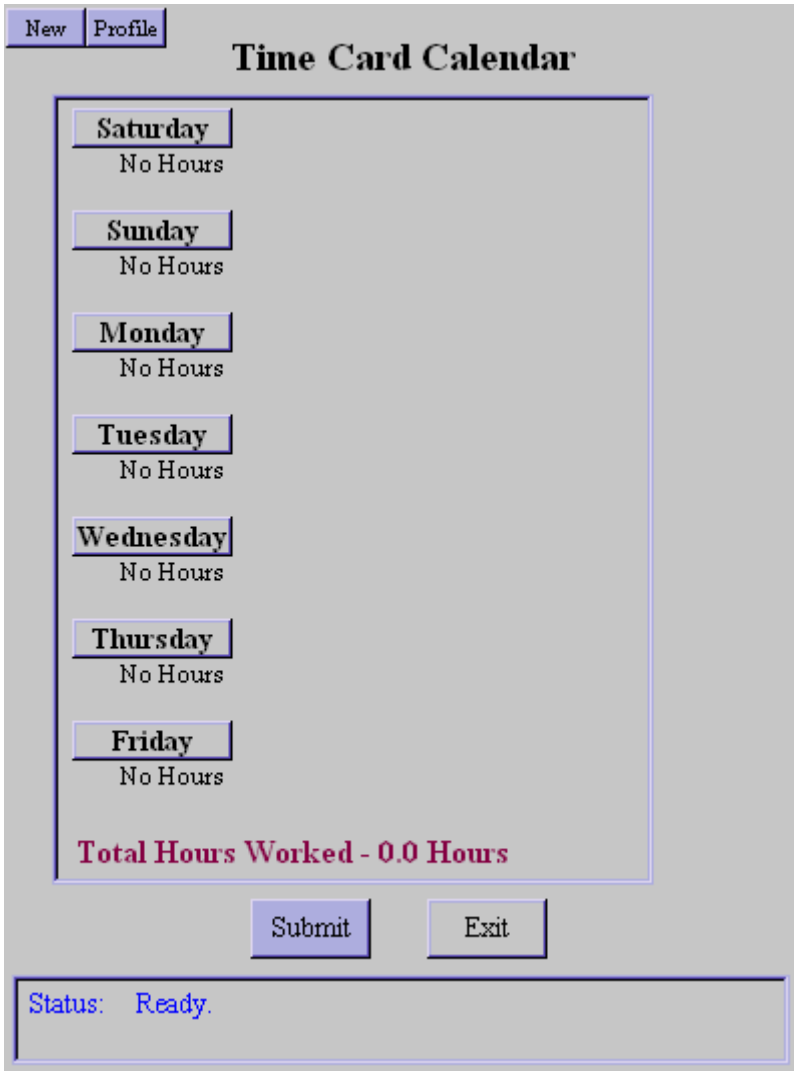


Figure 88. TOTALS applet - HACL for Java

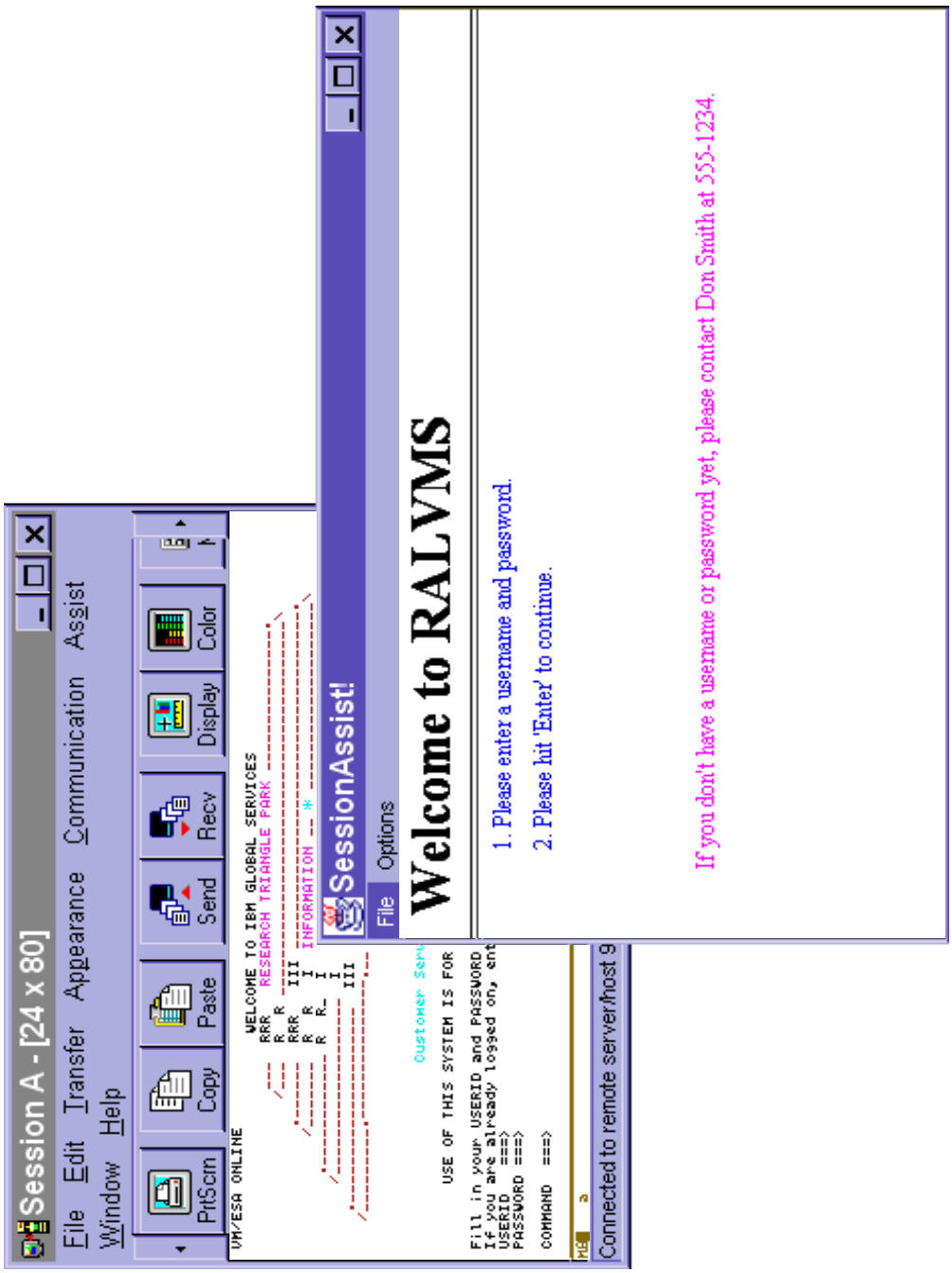


Figure 89. SessionAssist! - ECLApplets

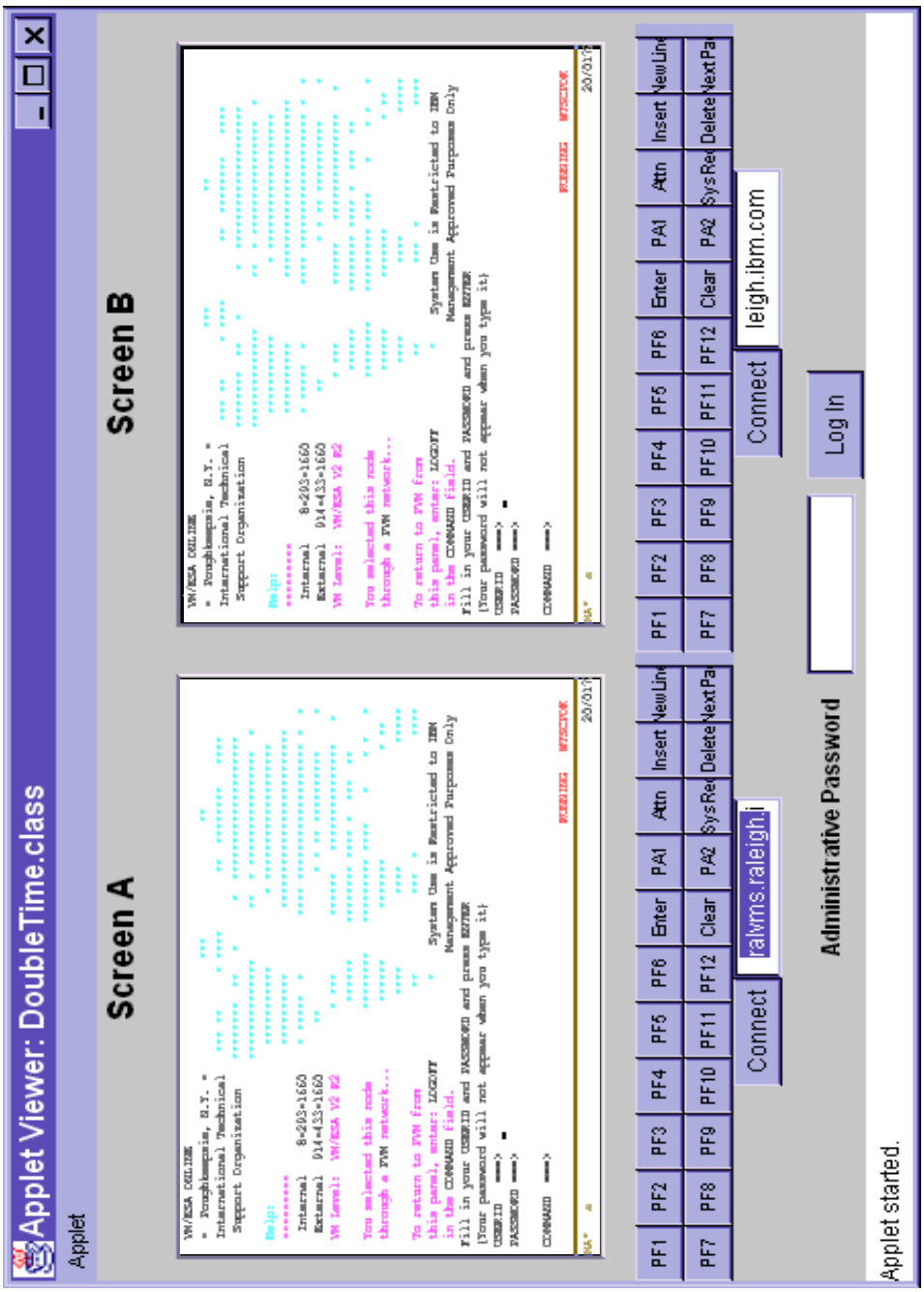


Figure 90. DoubleTime - Host Access Beans for Java

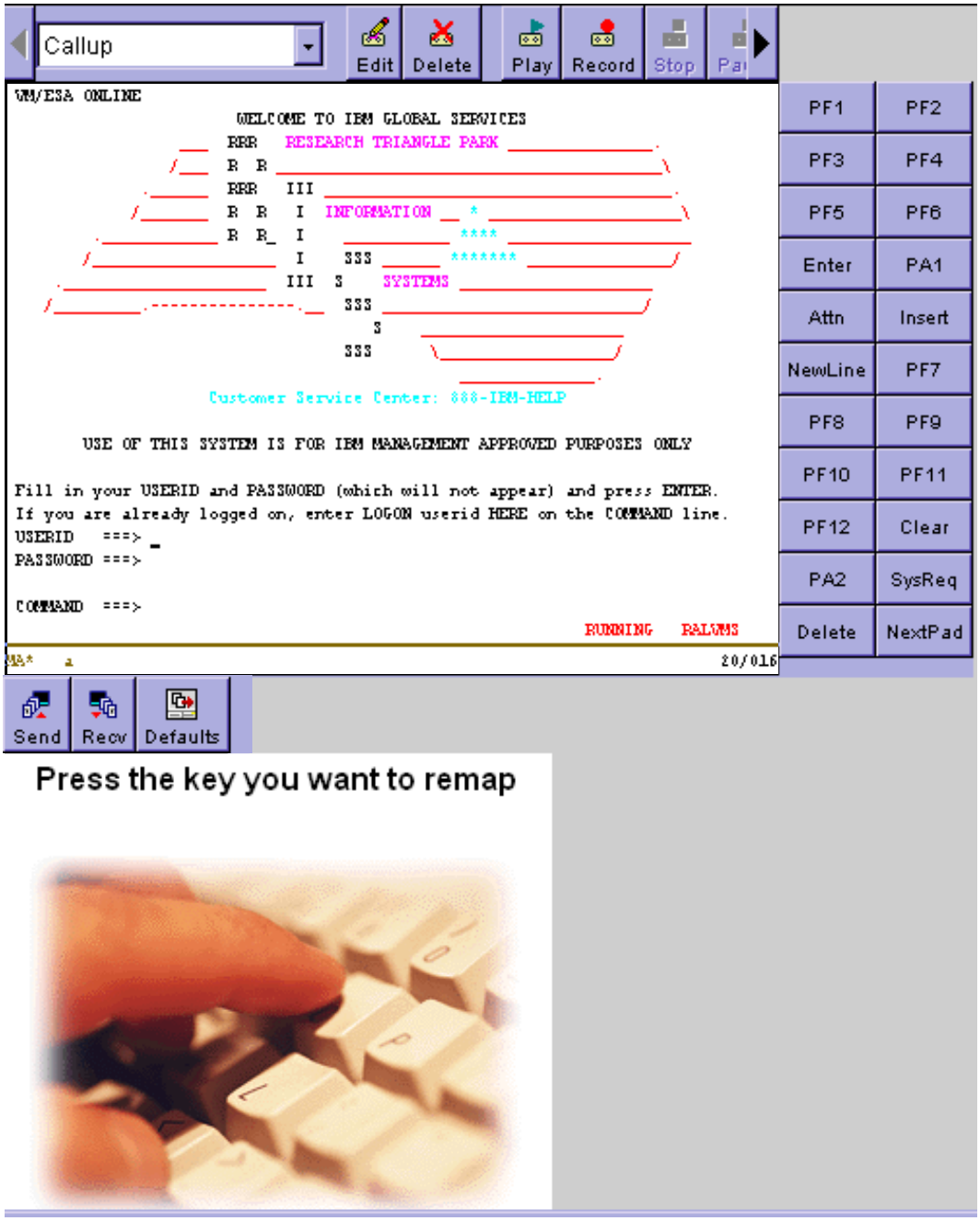


Figure 91. Em-You-Later with all components selected - Host Access Controls for ActiveX

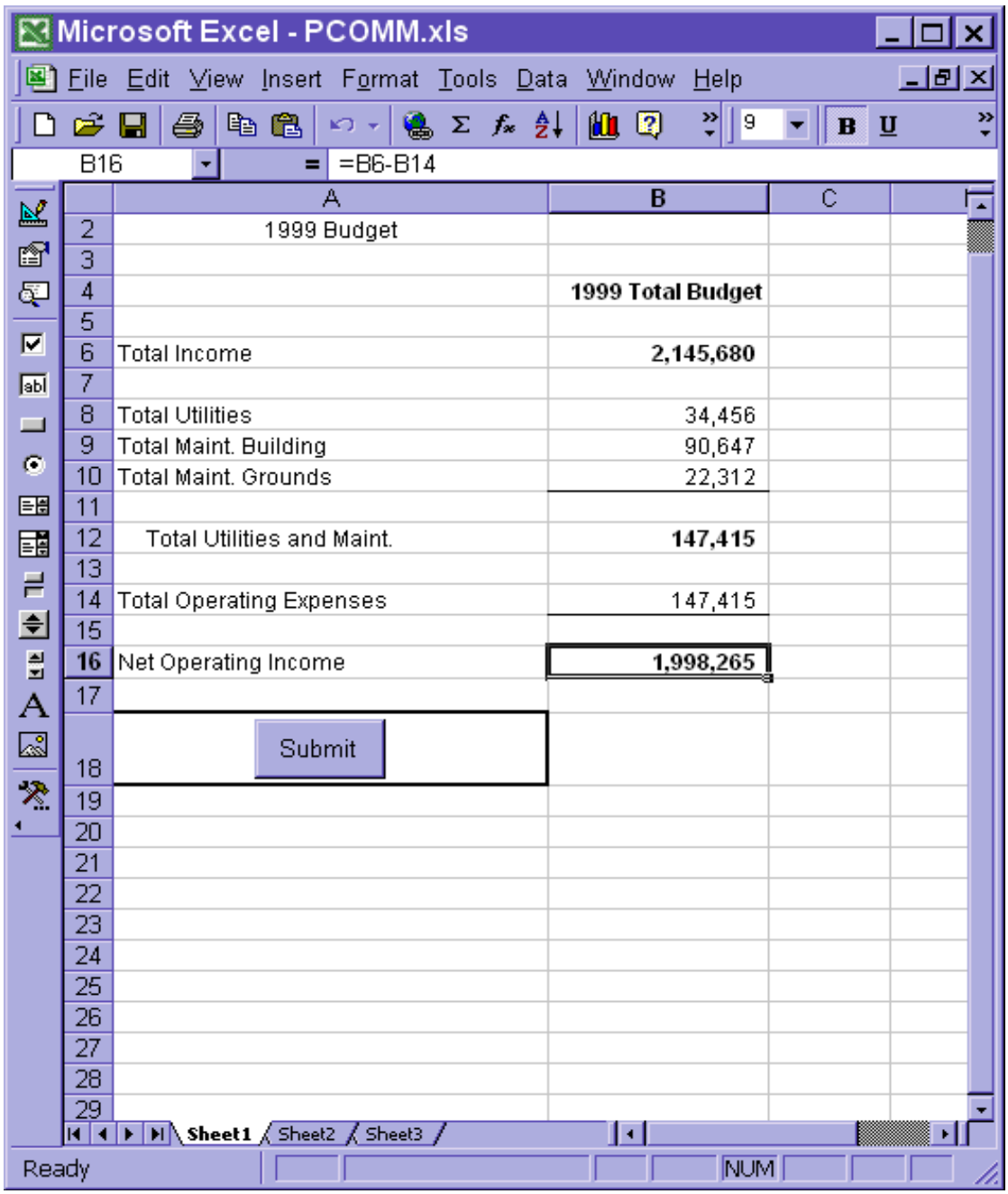


Figure 92. Excel Worksheet Submission Tool - HACL Automation objects

Appendix D. Using the additional material

This redbook also contains additional material in CD-ROM format. See the section below for instructions on using the material.

D.1 Using the CD-ROM

The CD-ROM that accompanies this redbook contains the following:

Files and Folders	*Description*
ActiveX	Host Access Controls for ActiveX coding examples
Automation	HACL Automation Objects coding examples
Host Access Beans	Host Access Beans for Java coding examples
Java	HACL for Java coding examples
Run Applets	ECLApplet coding examples
readme.txt	Information about the CD contents

D.1.1 System requirements for using the CD-ROM

The following system configuration is recommended for optimal use of the CD-ROM.

Hard disk space: 2 MB
Operating system: Windows NT, 95, or 98

D.1.2 How to use the CD-ROM

The CD-ROM contains programming examples talked about in this redbook. These examples can be browsed directly on the CD-ROM or can be copied from the CD-ROM to your workstation. These examples are the property of IBM and should be used only as reference material for coding your own programs.

D.2 Locating the additional material on the Internet

The CD-ROM associated with this redbook is also available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG245856

Alternatively, you can go to the IBM Redbooks Web site at:

http://www.redbooks.ibm.com/

Select the **Additional materials** and open the directory that corresponds with the redbook form number.

Appendix E. Special notices

This redbook is designed to help programmers design and implement Web-to-host solutions using Host Access APIs. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM SecureWay Host On-Demand Version 4.0 or IBM SecureWay Personal Communications V4.3, See the PUBLICATIONS section of the IBM Programming Announcement for these products 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, North Castle Drive, Armonk, NY 10504-1785.

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.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

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 ®	AS/400
CICS	eNetwork
VisualAge	OfficeVision
S/390	SecureWay

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries. (For a complete list of Intel trademarks see www.intel.com/tradmarx.htm)

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Appendix F. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

F.1 International Technical Support Organization publications

For information on ordering these ITSO publications see “How to get IBM Redbooks” on page 145.

- *Personal Communications Version 4.3 for Windows 95, 98, and NT,* SG24-4689
- *IBM SecureWay Host On-Demand: Enterprise Communications in the Era of Network Computing*, SG24-2149-00

F.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

F.3 Other publications

These publications are also relevant as further information sources:

- *eNetwork Personal Communications V4.3 for Windows 95, Windows 98, and Windows NT: Host Access Class Library*, SC31-8685

F.4 Referenced Web sites

The following Web sites are relevant for additional information:

- http://www-4.ibm.com/software/network/technology/hacl/
- http://java.sun.com
- http://www.javasoft.com/beans
- http://www.microsoft.com/java/
- http://developer.netscape.com/ software/signedobj/index.htm
- http://developer.netscape.com/docs/manuals/signedobj/capsapi.html
- http://msdn.microsoft.com/workshop/misc/cpad/default.asp
- http://www7.software.ibm.com/vad.nsf/

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download, or order hardcopy/CD-ROM redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl/

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

IBM Redbooks evaluation

Programming with the Host Access APIs
SG24-5856-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to `redbook@us.ibm.com`

Which of the following best describes you?
_ Customer _ Business Partner _ Solution Developer _ IBM employee
_ None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

What other redbooks would you like to see published?

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

SG24-5856-00
Printed in the U.S.A.

Programming with the Host Access APIs

SG24-5856-00