This chapter describes how you can use the Program-to-Program Communications (PPC) Toolbox to send and receive low-level message blocks between applications.

The PPC Toolbox can be used by different applications located on the same computer or across a network of Macintosh computers. The PPC Toolbox is available only in System 7 or later. To test for the existence of the PPC Toolbox, use the `Gestalt` function, described in Inside Macintosh: Operating System Utilities.

Read this chapter if you want your application to transmit and receive data from other applications that support the PPC Toolbox. Applications that utilize the PPC Toolbox must be open and connected to each other to exchange data. The PPC Toolbox allows you to send large amounts of data to other applications; it is typically useful for code that is not event-based. The PPC Toolbox is called by the Macintosh Operating System and can also be called by applications, device drivers, desk accessories, or other programs.

The PPC Toolbox provides a method of communication that is particularly useful for applications that are specifically designed to work together and are dependent on each other for information. For example, suppose one user organizes large amounts of data using a database application and another user filters and plots the same data using a plotting application. If both applications use the PPC Toolbox, these two applications can directly transmit data to each other when both applications are open and connected to each other.

You can also use the PPC Toolbox if your application communicates with other applications using high-level events or Apple events, and your application allows the user to choose another application to communicate with. You can use a PPC Toolbox routine that provides a standard user interface to display a dialog box that lists other applications that are available to exchange information. See “Browsing for Ports Using the Program Linking Dialog Box” beginning on page 11-22 for detailed information. See the chapter “Event Manager” in Inside Macintosh: Macintosh Toolbox Essentials for information on high-level events, and see earlier chapters in this book for information on Apple events.

The PPC Toolbox uses the AppleTalk Data Stream Protocol (ADSP) and the Name-Binding Protocol (NBP). For detailed information on ADSP and NBP, see Inside Macintosh: Networking.

Note
The sample applications “store data,” “display data,” “send and receive,” “make memo,” and “spell quick” used in this chapter are not actual products of Apple Computer, Inc. They are used for illustrative purposes only. ◆
About the PPC Toolbox

The PPC Toolbox provides you with the ability to
- exchange data with other open applications on the same computer or across a network of Macintosh computers
- browse through a listing of applications that are available to exchange data
- verify user identities for communication across a network

To utilize the PPC Toolbox to exchange data between open applications, each application involved must support the PPC Toolbox.

This chapter first defines the main elements of the PPC Toolbox and then discusses how to
- set up your application for communication
- use security features prior to establishing communication
- locate other applications that can exchange data
- initiate communication between applications
- accept or reject incoming communications requests
- transmit and receive data between applications
- terminate communication between applications

Ports, Sessions, and Message Blocks

To initiate communication between applications, you must first open a port. A port is a portal through which your application can exchange information with another application. A port is designated by a port name and a location name.

A port name is a unique identifier for a particular application on a computer. The port name contains a name string, a type string, and a script code for localization. The location name identifies the location of the computer on the network. The location name contains an object string, a type string, and a zone. An application can specify an alias location name by modifying its type string.

Your application can open as many ports as it requires as long as each port name is unique within a particular computer. See “Specifying Port Names and Location Names” beginning on page 11-17 for detailed information on port names and location names.
CHAPTER 11

Program-to-Program Communications Toolbox

Through its port, an open application can communicate with another open application during a **session**. One port can support any number of communication sessions. During a session, an application sends and receives data in the form of a **message block**. The PPC Toolbox treats each block of data as a byte stream and delivers it in the same sequence in which it was sent.

The words **port name**, **location name**, **session**, and **message block** are programmatic terms. You should not use them in the user interface of your application or in your user documentation. Instead, refer to a file that contains executable code as an **application program**. An application program that opens and uses PPC ports supports **program linking**. When you **link** two application programs together, you are forming a **program link**. A link allows two application programs to communicate with each other— you **unlink** two application programs when you break the link between them. You can compare the link between two application programs to the communication established using telephones. For example, a program link is similar to a telephone connection that enables various forms of communication such as human-to-human, modem-to-modem, and facsimile machine–to–facsimile machine.

Figure 11-1 shows a database application on one computer that has initiated a session with a spreadsheet application located on another computer on the network.

---

**Figure 11-1**  A PPC Toolbox session between two applications

---

**About the PPC Toolbox**

11-5
CHAPTER 11

Program-to-Program Communications Toolbox

The database application’s port name consists of “store data” (the name string) and “database” (the type string). Its location name consists of “Jane Doe’s Macintosh” (the object string), “PPCToolBox” (the type string), and “twilight” (the AppleTalk zone).

The spreadsheet application’s port name consists of “display data” (the name string) and “spreadsheet” (the type string). Its location name consists of “Joe Smith’s Macintosh” (the object string), “PPCToolBox” (the type string), and “loading” (the AppleTalk zone).

Setting Up Authenticated Sessions

Network communication must be active to initiate sessions with other computers across a network. The user must activate AppleTalk in the Chooser and enable program linking using the Sharing Setup control panel located in the Control Panels folder inside the System Folder. Figure 11-2 displays the icon for the Sharing Setup control panel.

Figure 11-2  The icon for the Sharing Setup control panel

![Sharing Setup Icon](image)

Figure 11-3 shows the Sharing Setup control panel.

Figure 11-3  The Sharing Setup control panel

![Sharing Setup Panel](image)
CHAPTER 11

Program-to-Program Communications Toolbox

To permit other computers to initiate sessions with the owner’s computer, the owner of the computer must click the Start button underneath Program Linking (Start toggles with Stop). The Sharing Setup control panel then indicates “Program linking is on. Click Stop to prevent other users from linking to your shared programs.” To prevent other computers from initiating sessions, an owner simply clicks Stop underneath Program Linking. The Sharing Setup control panel then indicates “Program linking is off. Click Start to allow other users to link to your shared programs.” Clicking the Start or Stop button also enables or disables the transmission of incoming Apple events across the network.

If a user clicks the Stop button while there are active incoming sessions (sessions initiated by other users), an alert box (shown in Figure 11-4) appears on the user’s screen.

Figure 11-4 The session termination alert box

If a user clicks OK, all active sessions initiated by other users are immediately terminated. Note that it is still possible for the owner of the computer to initiate sessions, even though other users may not initiate sessions with the owner’s computer.

The PPC Toolbox establishes the identity of users through the process of **authentication**. The authentication mechanism of the PPC Toolbox identifies each user through an assigned name and password. Each session initiated with a port that is located on a remote computer requires authentication (unless guest access is enabled) before a session is permitted. Sessions between applications located on the same computer never require authentication.

A computer’s owner can establish access for other users and guests by opening the Users & Groups control panel located in the Control Panels folder. The Users & Groups control panel allows an owner to specify the names and passwords of other users whose computers can initiating sessions with his or her ports across the network.

When the computer’s owner opens the Users & Groups control panel, the Guest icon appears. If the owner’s name is specified in the Sharing Setup control panel, an icon with the owner’s name also appears.
To specify a particular new user, the owner chooses New User from the File menu. The owner should type in the user’s name. When the owner opens a user icon in the Users & Groups control panel, the Finder displays the users and groups dialog box on the owner’s screen. Figure 11-5 shows the users and groups dialog box for a particular user.

To permit authenticated session requests, the owner can specify a password for each user. The owner allows other users to utilize the PPC Toolbox by clicking the checkbox under Program Linking. If the owner clicks the checkbox again, all active sessions initiated by this particular user are immediately terminated. The user termination alert box (shown in Figure 11-6) is displayed as a warning.
Program-to-Program Communications Toolbox

When the owner opens a Guest icon in the Users & Groups control panel, the Finder displays the guest dialog box on the owner’s screen. Authentication is not required if the owner permits guest access. Figure 11-7 shows the guest dialog box.

Figure 11-7  The guest dialog box

By clicking the checkbox under Program Linking, the owner permits guests to communicate using the PPC Toolbox or Apple events.

Consider this example of the authentication process: one user decides to make a dictionary service available to other users. A second user wishes to employ this service in a word-processing program. Assuming both programs support the PPC Toolbox, the word-processing program attempts to gain access to the dictionary service that is open on the first user’s computer by initiating a session. When the word-processing application requests a session, the PPC Toolbox attempts to authenticate the second user by requesting a user name and a password (unless guest access is enabled). If the authentication process verifies the user’s identity and the dictionary application accepts the request for a session, a session is established and the second user can access the dictionary’s data.
Figure 11-8 illustrates the authentication process that occurs when a user attempts to initiate a session.

**Using the PPC Toolbox**

This section describes how to

- use PPC Toolbox calling conventions
- open a port
- list all available port locations on the network
CHAPTER 11

Program-to-Program Communications Toolbox

- indicate that a port is available to accept session requests
- initiate a session
- accept and reject session requests
- read and write data during a session
- end a session after data is transmitted and received
- close a port when it is no longer needed to transmit or receive data
- invalidate users

To begin, you must determine whether the PPC Toolbox is available on the user’s computer system by using the Gestalt function with the selector gestaltPPCToolboxAttr. A noErr result code indicates that the PPC Toolbox is present.

The Gestalt function returns a combination of the following constants in the response parameter: gestaltPPCToolboxPresent, gestaltPPCSupportsRealTime, gestaltPPCSupportsOutGoing, and gestaltPPCSupportsIncoming.

The PPC Toolbox currently supports only sessions in real time. The Gestalt function returns gestaltPPCSupportsRealTime by default. If this bit is not set, you need to initialize the PPC Toolbox.

The Gestalt function returns gestaltPPCSupportsOutGoing to indicate support of outgoing sessions across a network of Macintosh computers. If this bit is not set, the user hasn’t enabled AppleTalk in the Chooser.

The Gestalt function returns gestaltPPCSupportsIncoming if the user has enabled program linking in the Sharing Setup control panel. If this bit is not set, the user either hasn’t enabled AppleTalk in the Chooser or hasn’t enabled program linking in the Sharing Setup control panel.

Use the PPCInit function to initialize the PPC Toolbox.

err := PPCInit;
CHAPTER 11

Program-to-Program Communications Toolbox

Listing 11-1 illustrates how you use the PPCInit function to initialize the PPC Toolbox.

**Listing 11-1** Initializing the PPC Toolbox using the PPCInit function

```pascal
FUNCTION MyPPCInit: OSErr;
VAR
    PPCAttributes: LongInt;
    err: OSErr;
BEGIN
    err := Gestalt(gestaltPPCToolboxAttr, PPCAttributes);
    IF err = noErr THEN  {PPC Toolbox is present}
        BEGIN
            IF BAND(PPCAttributes, gestaltPPCSupportsRealTime) = 0 THEN 
                BEGIN
                    MyPPCInit := PPCInit;  {initialize the PPC Toolbox}
                    {test the attributes for the PPC Toolbox}
                    err := Gestalt(gestaltPPCToolboxAttr, PPCAttributes);
                END;
            IF BAND(PPCAttributes, gestaltPPCSupportsOutGoing) <> 0 THEN 
                {ports can be opened to the outside world}
            ELSE 
                {it's likely that AppleTalk is disabled, so you }
                { may want to tell the user to activate AppleTalk }
                { from the Chooser}
            IF BAND(PPCAttributes, gestaltPPCSupportsIncoming) <> 0 THEN 
                {ports can be opened with location names that the }
                { outside world can see}
            ELSE 
                {it's likely that program linking is disabled, so }
                { you may want to tell the user to start program }
                { linking from the Sharing Setup control panel}
            END
        ELSE
        MyPPCInit := err;
    END;
```
Figure 11-9 illustrates a spreadsheet application (on the left) that has initiated a session with a database application (on the right) to exchange data using the PPC Toolbox. This figure includes an example of the sequence of PPC Toolbox routines executed by these applications. Detailed descriptions of the functions appear in the sections that follow.
To establish a session, each application must first open a port using the PPCOpen function. The spreadsheet application prepares to receive session requests by calling the PPCInform function.

Before initiating a session or opening a port, the database application can let the user browse through the list of available ports (using the PPCBrowser function). If the user decides to communicate with the spreadsheet application, the database application initiates a session with the spreadsheet application’s port using the StartSecureSession function. After the PPC Toolbox authenticates the user name and password of the initiating port, the spreadsheet application accepts the session request (using the PPCAccept function).

Once the session is established, the applications exchange information in the form of message blocks (using the PPCRead and PPCWrite functions). During a session, an application can both read from and write message blocks to another application. After the information exchange is done, each application ends the session (PPCEnd) and then closes its port (PPCClose) when it quits.

The PPCOpen function returns a port reference number. The port reference number is a reference number for the port through which you are requesting a session. The database application uses the port reference number in subsequent calls to the StartSecureSession and PPCClose functions. The StartSecureSession function returns a session reference number. The session reference number is used to identify the session during the exchange of data. It is used in subsequent calls to the PPCWrite, PPCRead, and PPCEnd functions.

The PPCOpen function returns a port reference number that the spreadsheet uses in subsequent calls to the PPCInform and PPCClose functions. The PPCInform function returns a session reference number that is used in subsequent calls to the PPCAccept, PPCRead, PPCWrite, and PPCEnd functions.

**PPC Toolbox Calling Conventions**

Most PPC Toolbox functions can execute synchronously (meaning that the application cannot continue until the function completes execution) or asynchronously (meaning that the application is free to perform other tasks while the function is executing). The PPC Toolbox functions that can only be executed synchronously include PPCInit, PPCBrowser, StartSecureSession, DeleteUserIdentity, and GetDefaultUser. All other PPC Toolbox functions can execute asynchronously or synchronously. Here’s an example:

FUNCTION PPCFunction (pb: PPCParamBlockPtr;
async: Boolean): OSErr;
The pb parameter should point to a PPC parameter block. Set the async parameter to TRUE if you want the function to execute asynchronously; set it to FALSE if you want the function to execute synchronously.

Note
The PPCInform, PPCRead, and PPCWrite functions should always be executed asynchronously, because they require interaction from the other application in the session before they complete execution.

The PPCParamBlockRec data type defines the PPC parameter block.

```plaintext
TYPE PPCParamBlockRec = RECORD
  CASE Integer OF
    0: (openParam: PPCOpenPBRec); {PPCOpen params}
    1: (informParam: PPCInformPBRec); {PPCInform params}
    2: (startParam: PPCStartPBRec); {PPCStart params}
    3: (acceptParam: PPCAcceptPBRec); {PPCAccept params}
    4: (rejectParam: PPCRejectPBRec); {PPCReject params}
    5: (writeParam: PPCWritePBRec); {PPCWrite params}
    6: (readParam: PPCReadPBRec); {PPCRead params}
    7: (endParam: PPCEndPBRec); {PPCEnd params}
    8: (closeParam: PPCClosePBRec); {PPCClose params}
    9: (listPortsParam: IPCListPortsPBRec); {IPCLListPorts } { params}
  END;
```

For an illustration of the fields of each individual parameter block (such as PPCInformPBRec or IPCListPortsPBRec), see Figure 11-18 on page 11-47.

Your application transfers ownership of the PPC parameter block (and any buffers or records pointed to by the PPC parameter block) to the PPC Toolbox until a PPC function completes execution. Once the function completes, ownership of the parameter block (and any buffers or records it points to) is transferred back to your application. If a PPC Toolbox function is executed asynchronously, your program cannot alter memory that might be used by the PPC Toolbox until that function completes.
A PPC Toolbox function that is executed asynchronously must specify NIL or the address of a completion routine in the ioCompletion field of the PPC parameter block. You should use the ioResult field to determine the actual result code when an asynchronously executed PPC Toolbox function completes.

If you specify NIL in the ioCompletion field, you should poll the ioResult field of the PPC parameter block after the function is called to determine whether the PPC function has completed the requested operation. You should poll the ioResult field within the event loop of your application. If the ioResult field contains a value other than 1, the function has completed execution. Note that you must not poll the ioResult field at interrupt time to determine whether the function has completed execution.

If you specify a completion routine in the ioCompletion field, it is called at interrupt time when the PPC Toolbox function completes execution.

▲ WARNING
Completion routines execute at the interrupt level and must preserve all registers other than A0, A1, and D0–D2. (Note that MPW C and MPW Pascal do this automatically.) Your completion routine must not make any calls to the Memory Manager directly or indirectly, and it can’t depend on the validity of handles to unlocked blocks. The PPC Toolbox preserves the application global register A5. ▲

You can write completion routines in C, Pascal, or assembly language. A completion routine declared in Pascal has this format:

```pascal
PROCEDURE MyCompletionRoutine (pb: PPCParamBlockPtr);
```

The pb parameter points to the PPC parameter block passed to the PPC Toolbox function. You may call another PPC Toolbox function from within a completion routine, but the function called must be executed asynchronously. It is recommended that you allocate parameter blocks of data type PPCParamBlockRec so that you may reuse the pb parameter to call another PPC Toolbox function from within a completion routine. For example, you should call either the PPCAccept function or the PPCReject function asynchronously from within a PPCInform completion routine to accept or reject the session request.

If your application is executing PPC Toolbox functions asynchronously, you may want to define your own record type to hold all data associated with a session. You can attach the data to the end of the parameter block. Here’s an example:
CHAPTER 11

Program-to-Program Communications Toolbox

TYPE
SessRecHndl = ^SessRecPtr;
SessRecPtr = ^SessRec;
SessRec =
RECORD
  pb: PPCParamBlockRec; {must be first }
  { item in record}
    thePPCPortRec: PPCPortRec;
    theLocationNameRec: LocationNameRec;
    theUserName: Str32;
END;

The additional data elements in your record can be accessed during execution of a completion routine by coercing the pb parameter to a pointer to your record type.

Specifying Port Names and Location Names

Before initiating a session, you must open a port to communicate with other programs. A port name and location name identify each port. An application can open as many ports as it requires as long as each port name is unique within a particular computer. You specify both the port name and the location name in the PPC parameter block.

Figure 11-10 illustrates a single Macintosh computer with two applications, and their corresponding port names and location names.

To open a port, you need to specify a port name. A port name consists of a name string, a type string, and a script code for localization. For example, you can designate “make memo” as the application’s name string, “word processor” as its type string, and “smRoman” as its script code.

A port name is defined by a PPC port record. The PPC port record contains a script code, name string, port kind selector, and type string. The script code is an integer script identifier used for localization. The name string consists of a 32-byte character string that designates the application name. You should keep both the script code and the name string in a resource. The port kind selector is an integer that selects the kind of type string. You should make it consistent internationally. The type string can be either a 32-byte character string or a 4-character creator and a 4-character file type. See the chapter “Finder Interface” of Inside Macintosh: Macintosh Toolbox Essentials for information on creators and file types. See Inside Macintosh: Text for information on script codes and localization.
The **PPCPortRec** data type defines the PPC port record.

```
TYPE PPCPortRec =
  RECORD
    nameScript: ScriptCode;  {script identifier}
    name: Str32; {port name in program }
      { linking dialog box}
    portKindSelector: PPCPortKinds; {general category of }
      { application}
    CASE PPCPortKinds OF
      ppcByString: (portTypeStr: Str32);
      ppcByCreatorAndType: (portCreator: OSTYPE;
        portType: OSTYPE);
    END;
```
CHAPTER 11

Program-to-Program Communications Toolbox

The location name identifies the location of the computer on the network. The PPC Toolbox provides the location name when the user starts up the computer. The location name is specified in the standard Name-Binding Protocol (NBP) form, \( \langle \text{object string} \rangle: \text{PPCToolBox} @\langle \text{AppleTalk zone} \rangle \). The object string is the name provided in the Sharing Setup control panel in the Control Panels folder. By default, the type string is “PPCToolBox”. The AppleTalk zone is the zone to which the particular Macintosh computer belongs. For example, “Jane Doe’s Macintosh:PPCToolBox@twilight” specifies the object string, type string, and AppleTalk zone for a particular computer.

The LocationNameRec data type defines the location name record. The locationKindSelector field can be set to ppcNoLocation, ppcNBPLocation, or ppcNBPTypeLocation.

```plaintext
TYPE LocationNameRec = RECORD
  locationKindSelector: PPCLocationKind; \{ which variant \}
  CASE PPCLocationKind OF
    { ppcNoLocation: storage not used by this value }
    ppcNBPLocation:
      (nbpEntity: EntityName); \{ NBP name entity \}
    ppcNBPTypeLocation:
      (nbpType: Str32); \{ just the NBP type \}
    \} \{ string for the \}
      \} \{ PPCOpen function \}
END;
```

The ppcNoLocation constant is used when the location received from or passed to a PPC Toolbox function is the location of the local machine.

The ppcNBPLocation constant is used when a full NBP entity name is received from or passed to a PPC Toolbox function.

**Note**

You should assign an NBP value directly—do not pack it using nbpSetEntity.

The ppcNBPTypeLocation constant is used only by the PPCOpen function when an alias location name is needed.

The NBP type to be used for the alias location name is passed in the location name record’s nbpType field. Alias location names allow you to filter the NBP objects (Macintosh computers) displayed by the program linking dialog box (shown in Figure 11-12 on page 11-22) using the PPCBrowser function. See “Browsing for Ports Using the Program Linking Dialog Box” beginning on page 11-22 for information on the PPCBrowser function.
An alias location name could be used to advertise a service (such as a dictionary service) that is available to any application located on the network. For example, “Joe Smith’s Macintosh: dictionary@ozone” specifies the object string, type string, and AppleTalk zone for a particular dictionary service.

To search for all dictionary services available within a zone, you use the PPCBrowser function and a filter. Figure 11-11 illustrates a Macintosh dictionary service application, its corresponding port name, and its alias location name.

---

### Opening a Port

To open a port and associate a name with it, use the PPCOpen function. Listing 11-2 illustrates how you use the PPCOpen function to open a port. In this listing, the name is “Inside Macintosh” and the port type string is “Example”. The location name is `<object string>:PPC Example@<AppleTalk zone>`. 

---

**Figure 11-11** The PPC Toolbox and a dictionary service application
Listing 11-2  Opening a PPC port

FUNCTION MyPPCOpen(VAR thePortRefNum: PPCPortRefNum;
         VAR nbpRegisteredFlag: Boolean): OSErr;

VAR
   thePPCOpenPBRec: PPCOpenPBRec;
   thePPCPortRec: PPCPortRec;
   theLocationNameRec: LocationNameRec;

BEGIN
   WITH thePPCPortRec DO
      BEGIN
         nameScript := smRoman; {Roman script}
         name := 'Inside Macintosh';
         portKindSelector := ppcByString;
         portTypeStr := 'Example';
      END;
   WITH theLocationNameRec DO
      BEGIN
         locationKindSelector := ppcNBPTypeLocation;
         nbpType := 'PPC Example';
      END;
   WITH thePPCOpenPBRec DO
      BEGIN
         serviceType := ppcServiceRealTime;
         resFlag := 0; {must be 0 for 7.0}
         portName := @thePPCPortRec;
         locationName := @theLocationNameRec;
         networkVisible := TRUE; {make this a visible }
                        { entity on the network}
      END;
   MyPPCOpen := PPCOpen(@thePPCOpenPBRec, FALSE);{synchronous}
   thePortRefNum := thePPCOpenPBRec.portRefNum;
   nbpRegisteredFlag := thePPCOpenPBRec.nbpRegistered;
END;
CHAPTER 11

Program-to-Program Communications Toolbox

The PPCOpen function opens a port with the port name and location name specified in the name and location fields of the parameter block. When the PPCOpen function completes execution, the portRefNum field returns the port reference number. You can use the port reference number in the PPCInform, PPCStart, StartSecureSession, and PPCClose functions to refer to the port you have opened.

Browsing for Ports Using the Program Linking Dialog Box

Before initiating a session, you can use either the PPCKVbrowser function or the IPCListPorts function to locate a port to communicate with.

Use the PPCBrowser function to display the program linking dialog box (shown in Figure 11-12) on the user’s screen.

**Note**

Because this function displays a dialog box on the user’s screen, you must not call the PPCBrowser function from an application that is running in the background.

**Figure 11-12** The program linking dialog box

In the program linking dialog box, the user selects the computer, zone, and application. The zone list is not displayed if there is no network connection. Figure 11-13 shows the dialog box without the zone list.
As shortcuts for the user, the program linking dialog box supports standard keyboard equivalents. Pressing Command-period or the Esc (Escape) key selects Cancel—pressing Enter or Return selects the OK button.

Each list is sorted in alphabetical order. As in the Chooser, the current list is indicated by a thick outline around its border. The program linking dialog box supports keyboard navigation and use of the arrow keys to select items from the current list. Pressing Tab or clicking the rectangle of another list switches the current list. Pressing Shift-Tab reverses the order in which the lists are selected. In addition, double-clicking an application name in the Programs list of the program linking dialog box is equivalent to clicking the OK button.

The PPCBrowser function allows users to browse for PPC ports.

```pascal
err := PPCBrowser (prompt, applListLabel, defaultSpecified, theLocation, thePortInfo, portFilter, theLocNBPType);
```

If the defaultSpecified parameter is TRUE, the PPCBrowser function tries to select the PPC port specified by the parameters theLocation and thePortInfo when the program linking dialog box first appears. If the default cannot be found, the PPCBrowser function selects the first PPC port in the list.
An application can open multiple ports as long as each port name is unique within a particular computer. Unique ports can have duplicate name fields but different types. For example, you can designate “make memo” as the application’s name string and “word processor” as its type string. You can also designate a separate port as “make memo” (the application’s name string) and “text only” (its type string).

In such a case, the PPCBrowser function does a secondary sort based on the port type. Ports with a type selector of ppcByCreatorAndType are displayed before ppcByString ports, and types are sorted alphabetically within each type selector.

The PPCBrowser function uses the IPCListPorts function to obtain the list of existing ports on a particular computer within a particular zone. The portFilter parameter of the PPCBrowser function allows you to filter the list of PPC ports before it displays them in the program linking dialog box. If you set the portFilter parameter to NIL, the PPCBrowser function displays the names of all the existing PPC ports returned by the IPCListPorts function. If you do not set the portFilter parameter to NIL, you must set it to a pointer to a port filter function that you create.

Listing 11-3 illustrates how you use a sample port filter function. In this listing, the MyBrowserPortFilter function returns TRUE for ports with the port type string “Example”.

Listing 11-3  Using a port filter function

FUNCTION MyBrowserPortFilter(theLocationNameRec: LocationNameRec;
                                  thePortInfoRec: PortInfoRec)
     : Boolean;
BEGIN
  IF thePortInfoRec.name.portKindSelector = ppcByString THEN
    IF thePortInfoRec.name.portTypeStr = 'Example' THEN
      MyBrowserPortFilter := TRUE
    ELSE
      MyBrowserPortFilter := FALSE
    END;
  ELSE
    MyBrowserPortFilter := FALSE;
  END;
CHAPTER 11

Program-to-Program Communications Toolbox

The PPCBrowser function calls your filter function once for each port on the selected computer. Your function should return TRUE for each port you want to display in the program linking dialog box, and FALSE for each port that you do not want to display. Do not modify the data in the filter function parameters theLocationNameRec and thePortInfoRec.

The PPCBrowser function returns the selected port name in the parameter thePortInfo. The IPCListPorts function returns the port names in the area of memory pointed to by the bufferPtr field of the IPCListPorts parameter block. Both functions specify each port name in a port information record.

```pascal
TYPE PortInfoRec =
  RECORD
    filler1: SignedByte; {space holder}
    authRequired: Boolean; {authentication required}
    name: PPCPortRec; {port name}
  END;
```

If the authRequired field returns TRUE, the port requires authentication before a session can begin. You should use the StartSecureSession function to initiate a session with this port. If this field returns FALSE, you can use either the PPCStart function or the StartSecureSession function to initiate a session. See “Initiating a PPC Session” beginning on page 11-29 for detailed information. The name field of the port information record specifies an available port name.

Listing 11-4 illustrates how you use the PPCBrowser function to display the program linking dialog box in order to obtain the location and name of a port chosen by the user. In this listing, the PPCBrowser function builds lists of zones (shown in the AppleTalk Zones list of the program linking dialog box), objects (shown in the Macintoshes list), and ports (shown in the Programs list). In this example, the PPCBrowser function next tries to default to object “Moof™” in the “Twilight” zone. If it matches the object and zone, it also tries to default to the port “Inside Macintosh” with the port type “Example”. 
CHAPTER 11

Program-to-Program Communications Toolbox

Note that the data in the records LocationNameRec and PortInfoRec is used to match the names in the program linking dialog box. The data has nothing to do with the NBP type used by NBPLookup or the filtered PPC ports that show up in the program linking dialog box. The NBPLookup function uses the NBP type supplied in the LocNBPType. The PPC port names are filtered using the MyBrowserPortFilter function shown in Listing 11-3 on page 11-24.

Listing 11-4  Browsing through dictionary service ports

FUNCTION MyPPCBrowser(VAR theLocationNameRec: LocationNameRec; VAR thePortInfoRec: PortInfoRec): OSErr;

VAR
prompt: Str255;
applListLabel: Str255;
defaultSpecified: Boolean;
theLocNBPType: Str32;
BEGIN
prompt := 'Choose an example to link to:';
applListLabel := 'Examples';
defaultSpecified := TRUE;
WITH theLocationNameRec DO
BEGIN
locationKindSelector := ppcNBPLocation;
WITH nbpEntity DO
BEGIN
objStr := 'Moof™';
{typeStr is ignored}
zoneStr := 'Twilight';
END;
END;
END;
WITH thePortInfoRec.name DO
BEGIN
{nameScript and name should be resources to allow easy localization}
nameScript := smRoman; {Roman script}
name := 'Inside Macintosh';
{the port type should always be hard-coded to allow the application to find ports of a particular type even after the name is localized}
portKindSelector := ppcByString;
portTypeStr := 'Example';
END;
Program-to-Program Communications Toolbox

```
{ when building the list of objects (Macintoshes), }
{ show only those with the NBP type "PPC Example" }
theLocNBPType := 'PPC Example'; {match this NBP type }
MyPPCBrowser := PPCBrowser(prompt, applListLabel, 
defaultSpecified, 
theLocationNameRec, 
thePortInfoRec, 
@MyBrowserPortFilter, 
theLocNBPType);

END;
```

Obtaining a List of Available Ports

To generate a list of ports without displaying dialog boxes, you can use the IPCListPorts function. The IPCListPorts function allows you to obtain a list of ports on a particular computer within a particular zone. To obtain a list of ports, several steps are required. First, use the GetZoneList function to obtain a list of zones. Next, you must use the PLookupName function to obtain a list of computers with ports. After establishing the zone and the computer, you can use the IPCListPorts function to obtain the list of available ports. See Inside Macintosh: Networking for information on the GetZoneList and PLookupName functions.

Listing 11-5 illustrates how you use the IPCListPorts function to obtain a list of ports on a particular computer. This function returns a list of port information records in the buffer pointed to by the parameter thePortInfoBufferPtr. The actual number of port information records is returned in the parameter theActualCount.
Listing 11-5  Using the IPCListPorts function to obtain a list of ports

FUNCTION MyIPCListPorts
  (theStartIndex: Integer;
   theRequestCount: Integer; VAR theActualCount: Integer;
   theObjStr: Str32; theZoneStr: Str32;
   thePortInfoBufferPtr: PortInfoArrayPtr): OSErr;

VAR
  theIPCListPortsPBRec:  IPCListPortsPBRec;
  thePPCPortRec:  PPCPortRec;
  theLocationNameRec: LocationNameRec;
BEGIN
  {list all PPC ports at the specified location}
  WITH thePPCPortRec DO
    BEGIN
      nameScript := smRoman;
      name := '=';  (*match all names*)
      portKindSelector := ppcByString;
      portTypeStr := '=';  (*match all types*)
    END;
  WITH theLocationNameRec DO
    BEGIN
      locationKindSelector := ppcNBPLocation;
      WITH nbpEntity DO
        BEGIN
          {set NBP object from the list returned by NBPLookup}
          objStr := theObjStr;
          {set NBP type, in this example to "PPC Example"; if you }
          {don't supply your own NBP type, use "PPCToolBox"}
          typeStr := 'PPC Example';
          {set NBP zone from the list returned by GetZoneList}
          zoneStr := theZoneStr;
        END;
      END;
    END;
  WITH theIPCListPortsPBRec DO
    BEGIN
      startIndex := theStartIndex;
      requestCount := theRequestCount;
      portName := @thePPCPortRec;
      locationName := @theLocationNameRec;
      bufferPtr := thePortInfoBufferPtr;
    END;
The `IPCListPorts` function returns information about ports that are on the computer specified in the `locationName` field of the list ports parameter block. If you set the `locationName` field to `NIL` or if you set the `locationKindSelector` field in the location name record to `ppcNoLocation`, the `IPCListPorts` function returns only the port names for the local computer.

The `bufferPtr` field points to an area of memory that contains the requested port names. You are responsible for allocating enough memory to hold the requested port names. The buffer length must be equal to

\[ \text{sizeof(PortInfoRec)} \times \text{requestCount} \]

### Preparing for a Session

To communicate, you can open a port for your application and make it available to receive session requests, to initiate sessions, or both. Applications that are able to receive session requests can choose to accept or reject incoming session requests.

Before an application can accept and establish a session with another application, the PPC Toolbox authenticates the initiating user (unless guest access is enabled or the applications are located on the same computer). Once a session begins, the two applications can exchange data with each other.

### Initiating a PPC Session

Once you have established the name and the location of the port that you want to communicate with, you can initiate a session. You can use either the `StartSecureSession` function or the `PPCStart` function to initiate a session.

The `StartSecureSession` function displays several dialog boxes to identify each user who requests a session. You may prefer to use the `PPCStart` function for low-level code such as that used for drivers, which typically do not provide a user interface. You may also prefer to use `PPCStart` when the application you are initiating a session with does not require authentication. The `IPCListPorts` and `PPC Browser` functions return information about whether a particular port requires authentication.

**Note**

Do not call the `StartSecureSession` function from an application that is running in the background, because the function displays several dialog boxes on the user’s screen. ♦
The StartSecureSession function provides authentication services to identify each user who requests a session. This function combines the processes of prompting for user name and password and initiating a session into one synchronous procedure call. If authentication fails, the PPC Toolbox rejects the incoming session request.

```plaintext
err := StartSecureSession (pb, userName, useDefault, allowGuest, 
guestSelected, prompt);
```

Set the useDefault parameter to TRUE if you want the StartSecureSession function to use the default user identity (described later in this section). If the default user identity cannot be authenticated, the StartSecureSession function displays a dialog box to allow a user to log on. Figure 11-14 shows the user identity dialog box.

![User Identity Dialog Box](image)

**Figure 11-14** The user identity dialog box

The prompt parameter of the StartSecureSession function allows you to specify a line of text that the dialog box can display. The allowGuest parameter specifies whether to enable the Guest radio button. If a port requires authentication, you should set this parameter to FALSE.

The userName parameter specifies the name of the user who is attempting to initiate a session. If the user name is not specified, the user identity dialog box appears on the user’s screen with the owner name provided from the Sharing Setup control panel.
If the user enters an invalid password, the `StartSecureSession` function displays the dialog box shown in Figure 11-15.

![Figure 11-15 The incorrect password dialog box](image)

After the user clicks OK, the user identity dialog box reappears in the foreground so that the user can enter the password again.

If the user’s name is invalid, the `StartSecureSession` function displays the dialog box shown in Figure 11-16.

![Figure 11-16 The invalid user name dialog box](image)

After the user clicks OK, the user identity dialog box reappears so that the user can enter a new user name.

The `StartSecureSession` function remains in this loop until a secure session is initiated or the user clicks Cancel in the user identity dialog box. If a secure session is initiated, `StartSecureSession` returns the user reference number in the corresponding field in the PPCStart parameter block. The user reference number represents the user name and password. A user reference number of 0 indicates that a session has been initiated with guest access. See “Setting Up Authenticated Sessions” beginning on page 11-6 for more information.
Before your application quits, you need to invalidate all user reference numbers obtained
with the StartSecureSession function except for the default user reference number
and the guest reference number (0). See “Invalidating Users” on page 11-44 for detailed
information.

Listing 11-6 illustrates how to use the StartSecureSession function to establish an
authenticated session. This listing shows only one session, although your application
may conduct multiple sessions at one time.

**Listing 11-6** Using the StartSecureSession function to establish a session

```pascal
FUNCTION MyStartSecureSession(thePortInfoPtr: PortInfoPtr;
    theLocationNamePtr: LocationNamePtr;
    thePortRefNum: PPCPortRefNum;
    VAR theSessRefNum: PPCSessRefNum;
    VAR theUserRefNum: LongInt;
    VAR theRejectInfo: LongInt;
    VAR userName: Str32;
    VAR guestSelected: Boolean): OSErr;

VAR
    thePPCStartPBRec: PPCStartPBRec;
    useDefault: Boolean;
    allowGuest: Boolean;
    err: OSErr;
BEGIN
    WITH thePPCStartPBRec DO
    BEGIN
        ioCompletion := NIL;
        portRefNum := thePortRefNum; {from the PPCOpen function}
        serviceType := ppcServiceRealTime;
        resFlag := 0;
        portName := @thePortInfoPtr^.name; {from the PPCBrowser}
        locationName := theLocationNamePtr; {from the PPCBrowser}
        userData := 0; {application-specific data that the }
            { PPCInform function sees}
    END;
    useDefault := TRUE;
    allowGuest := NOT thePortInfoPtr^.authRequired;
    err := StartSecureSession(@thePPCStartPBRec, userName,
        useDefault, allowGuest,
        guestSelected, stringPtr(NIL)^);
```
Chapter 11

Program-to-Program Communications Toolbox

IF err = noErr THEN
  BEGIN
    theSessRefNum := thePPCStartPBRec.sessRefNum;
    theUserRefNum := thePPCStartPBRec.userRefNum;
  END
ELSE
  IF err = userRejectErr THEN
    {return rejectInfo from the PPCReject function}
    theRejectInfo := thePPCStartPBRec.rejectInfo;
    MyStartSecureSession := err;
  END;
END;

For low-level code such as that used for drivers (which typically do not provide a user interface), you can use the PPCStart function instead of the StartSecureSession function to initiate a session. You can also use the IPCListPorts function (instead of displaying the program linking dialog box) to obtain a list of ports.

If the authRequired field of the port information record contains FALSE, the port allows guest access. If the authRequired field of the port information record contains TRUE, use the PPCStart function and the user reference number obtained previously from the StartSecureSession function to reestablish an authenticated session.

You can also attempt to log on as the default user using the GetDefaultValue function to obtain the default user reference number and the default user name. The default user name is established after the owner starts up the computer.

err := GetDefaultUser (userRef, userName);

The userRef parameter is a reference number that represents the user name and password of the default user. The userName parameter contains the owner name that is specified in the Sharing Setup control panel.

The GetDefaultUser function returns an error when the default user identity does not exist (no name is specified in the Sharing Setup control panel) or the user is not currently logged on.

Listing 11-7 illustrates how you use the PPCStart function to initiate a session. The PPCStart function uses the port information record and the location name record to attempt to open a session with the selected PPC port.
Listing 11-7  Initiating a session using the PPCStart function

FUNCTION MyPPCStart(thePortInfoPtr: PortInfoPtr;
theLocationNamePtr: LocationNamePtr;
thePortRefNum: PPCPortRefNum;
VAR theSessRefNum: PPCSessRefNum;
VAR theUserRefNum: LongInt;
VAR theRejectInfo: LongInt): OSErr;

VAR
  thePPCStartPBRec: PPCStartPBRec;
  userName: Str32;
  err: OSErr;
BEGIN
  WITH thePPCStartPBRec DO
    BEGIN
      ioCompletion := NIL;
      portRefNum := thePortRefNum; {from the PPCOpen function}
      serviceType := ppcServiceRealTime;
      resFlag := 0;
      portName := @thePortInfoPtr^.name; {destination port}
      locationName := theLocationNamePtr; {destination location}
      userData := 0; {application-specific data for PPCInform}
    END;
    err := GetDefaultUser(thePPCStartPBRec.userRefNum, userName);
    IF err <> noErr THEN
      thePPCStartPBRec.userRefNum := 0;
    IF thePortInfoPtr^.authRequired AND
      (thePPCStartPBRec.userRefNum = 0) THEN
      {port selected doesn't allow guests & you don't have a }
      { default user ref number so you can't log on to this port}
      err := authFailErr
    ELSE {attempt to log on}
      err := PPCStart( @thePPCStartPBRec, FALSE);
    IF err = noErr THEN
      BEGIN
        theSessRefNum := thePPCStartPBRec.sessRefNum;
        theUserRefNum := thePPCStartPBRec.userRefNum;
      END
    ELSE IF err = userRejectErr THEN
      {return rejectInfo from the PPCReject function}
      theRejectInfo := thePPCStartPBRec.rejectInfo;
      MyPPCStart := err;
    END;
END;
CHAPTER 11

Program-to-Program Communications Toolbox

The port to which you wish to connect must have an outstanding PPCInform function to successfully start a session. You cannot initiate a session with a port that is not able to receive session requests.

If the port is open, has an outstanding PPCInform function posted, and accepts your session request, the PPCStart function returns a noErr result code and a valid session reference number. This session reference number is used to identify the session during the exchange of data.

Receiving Session Requests

Your application can open as many ports as it requires as long as each port name is unique within a particular computer. A single port can support a number of communication sessions. To allow a port to receive session requests, use the PPCInform function. (Note that you must open a port to obtain a port reference number before calling the PPCInform function.) A port may have any number of outstanding PPCInform requests.

Listing 11-8 illustrates how you use the PPCInform function to allow a port to receive session requests. In this listing, the parameter thePPCParamBlockPtr points to a PPC parameter block record allocated by the application. The portRefNum, autoAccept, portName, locationName, userName, and ioCompletion parameters of the PPC parameter block record must be supplied. If you want to automatically accept all incoming session requests, you can set the autoAccept field in the PPCInform parameter block.
CHAPTER 11

Program-to-Program Communications Toolbox

Listing 11-8 Using the PPCInform function to enable a port to receive sessions

FUNCTION MyPPCInform(thePPCParamBlockPtr: PPCParamBlockPtr;
thePPCPortPtr: PPCPortPtr;
theLocationNamePtr: LocationNamePtr;
theUserNamePtr: stringPtr;
thePortRefNum: PPCPortRefNum): OSErr;

BEGIN
WITH thePPCParamBlockPtr^.informParam DO
BEGIN
ioCompletion := @MyInformCompProc;
portRefNum := thePortRefNum; {from the PPCOpen function}
autoAccept := FALSE; {the completion routine }
{ handles accepting or }
{ rejecting requests}
portName := thePPCPortPtr;
locationName := theLocationNamePtr;
userName := theUserNamePtr;
END;
MyPPCInform := PPCInform(PPCInformPBPtr(thePPCParamBlockPtr),
TRUE); {asynchronous}
END;

A PPC parameter block record is used instead of a PPCInform parameter block record so that the same parameter block can be reused to make other PPC Toolbox calls from the PPCInform completion routine. The parameter block and the records it points to cannot be deallocated until all calls that use the parameter block and records have completed.

You should make the call to PPCInform asynchronously. For each function that you use asynchronously, you should provide a completion routine. The completion routine gets called at interrupt time when the PPCInform function completes.
CHAPTER 11

Program-to-Program Communications Toolbox

Listing 11-9 illustrates a completion routine for a PPCInform function. You can use the data passed into your PPCInform completion routine (user name, user data, port name, and location name) to determine whether to accept or reject the session request.

Listing 11-9  Completion routine for a PPCInform function

PROCEDURE MyInformCompProc(pb: PPCParamBlockPtr);
BEGIN
  IF pb^.informParam.ioResult = noErr THEN
  BEGIN
    { decide if this session should be accepted or rejected by }
    { looking at data supplied by the session requester }
    IF pb^.informParam.userData <> -1 THEN
      DoPPCAccept(pb)
    ELSE
      DoPPCReject(pb);
  END
  ELSE
  { use a global to tell the application that }
  { PPCParamBlockRec and the records it points to }
  { can be deallocated }
  gPBInUse := FALSE;
END;

When the PPCInform function completes, the MyInformCompProc procedure determines whether to accept or reject the incoming session request. It does this by calling PPCAccept or PPCReject, as described in the next section.

Accepting or Rejecting Session Requests

Use the PPCAccept function or the PPCReject function to accept or reject an incoming session request.

▲ WARNING

If the PPCInform function (with the autoAccept parameter set to FALSE) returns a noErr result code, you must call either the PPCAccept function or the PPCReject function. The computer trying to initiate a session (using the StartSecureSession function or the PPCStart function) waits (hangs) until the session attempt is either accepted or rejected, or until an error occurs. ▲
CHAPTER 11

Program-to-Program Communications Toolbox

Listing 11-10 illustrates how you use the PPCAccept function to accept a session request. This listing reuses the parameter block used in the PPCInform function, so the sessRefNum field already contains the session reference number needed by the PPCAccept function.

Listing 11-10  Accepting a session request using the PPCAccept function

PROCEDURE DoPPCAccept(pb: PPCParamBlockPtr);
VAR
   err: OSErr;
BEGIN {accept the session}
   pb^.acceptParam.ioCompletion := @MyAcceptCompProc;
   {the sessRefNum field is set by the PPCInform function}
   err := PPCAccept(@pb^.acceptParam, TRUE); {asynchronous}
END;

For each function that you use asynchronously, you should provide a completion routine. Listing 11-11 illustrates a completion routine for a PPCAccept function. This procedure gets called at interrupt time when the PPCAccept function completes. If there are no errors, it sets the global variable gSessionOpen to TRUE. The global variable gPBInUse is set to FALSE to inform the application that the parameter block and the records it points to are no longer in use.

You can use the session reference number in subsequent PPCWrite, PPCRead, and PPCEnd functions once a session is accepted.

Listing 11-11  Completion routine for a PPCAccept function

PROCEDURE MyAcceptCompProc(pb: PPCParamBlockPtr);
BEGIN
   IF pb^.acceptParam.ioResult = noErr THEN
      {accept completed so the session is completely open}
      gSessionOpen := TRUE;
      {use a global to tell the application that PPCParamBlockRec }
      { and the records it points to can be deallocated}
      gPBInUse := FALSE;
   END;

Use the PPCReject function to reject an incoming session request. Listing 11-12 illustrates how you use the PPCReject function to reject a session request.
CHAPTER 11

Program-to-Program Communications Toolbox

This listing reuses the parameter block used in the PPCInform function, so the sessRefNum field already contains the session reference number needed by the PPCReject function.

Listing 11-12  Rejecting a session request using the PPCReject function

PROCEDURE DoPPCReject(pb: PPCParamBlockPtr);
VAR
   err: OSErr;
BEGIN {reject the session}
   WITH pb^.rejectParam DO
   BEGIN
      ioCompletion := @MyRejectCompProc;
      {the sessRefNum field is set by the PPCInform function}
      rejectInfo := -1;
      END;
      err := PPCReject(@pb^.rejectParam, TRUE); {asynchronous}
   END;

Listing 11-13 illustrates a completion routine for a PPCReject function. This procedure is called at interrupt time when the PPCReject function completes. In this example, the global variable gPBInUse is set to FALSE to inform the application that the parameter block and the records it points to are no longer in use.

Listing 11-13  Completion routine for a PPCReject function

PROCEDURE MyRejectCompProc(pb: PPCParamBlockPtr);
BEGIN
   {use a global to tell the application that PPCParamBlockRec }
   { and the records it points to can be deallocated}
   gPBInUse := FALSE;
END;

Exchanging Data During a PPC Session

After a session begins, each application can send data to and receive data from the other using a sequence of message blocks. The PPC Toolbox treats each message block as a byte stream and does not interpret the contents of the message block. The size of a message block can be between 0 and \((2^{32}-1)\) bytes. The PPC Toolbox treats the buffer size as an unsigned long integer.
The PPC Toolbox delivers the message blocks in the same sequence as they are sent and without duplicates. In Figure 11-17, an application transmits message blocks during a session.

For each message block, you specify a block creator, block type, and user data. The first `PPCWrite` function that you use to create a new message block sets the attributes for the block. The `PPCRead` function returns the block creator, block type, and user data attributes for the current message block when the call completes.

Although the PPC Toolbox does not interpret these attributes, they can give the receiving application information about how to process the contents of the message block. For example, a database application may specify, in the block creator field, a counter to indicate the block number (block number 20 of 30 total blocks). This application could also specify a code, such as ‘DREC’, in the block type field to indicate that the information it contains is a database record. In addition, this application could specify, in the user data field, the length of the message block.

**Reading Data From an Application**

An application can both read from and write data to another application during a session. Use the `PPCRead` function during a session to read incoming blocks of data from another application.

Once a session is initiated, you should have a `PPCRead` function pending. You can issue a `PPCRead` function from inside a completion routine. This provides you with immediate notification if an error condition arises or the session closes.
CHAPTER 11

Program-to-Program Communications Toolbox

The blockCreator, blockType, and userData fields are returned for the block you are reading. (These fields are set by the PPCWrite function.) To determine whether there is additional data to be read, check the more field. The value FALSE indicates the end of a message block.

Listing 11-14 illustrates how you use the PPCread function to read data during a session.

Listing 11-14 Using the PPCRead function to read data during a session

FUNCTION MyPPCRead(thePPCReadPBPtr: PPCReadPBPtr; theSessRefNum: PPCSessRefNum; theBufferLength: Size; theBufferPtr: Ptr): OSErr;
BEGIN
  WITH thePPCReadPBPtr^ DO
  BEGIN
    ioCompletion := NIL;
    sessRefNum := theSessRefNum; {from PPCStart or PPCInform}
    bufferLength := theBufferLength;
    bufferPtr := theBufferPtr;
  END;
  MyPPCRead := PPCRead(thePPCReadPBPtr, TRUE); {asynchronous}
END;

You should make any calls to PPCRead asynchronously. You can provide a completion routine that will be called when the PPCRead function has completed, or you can poll the ioResult field of the PPC parameter block to determine whether the PPCRead function has completed. A PPCRead completion routine can issue another asynchronous PPC Toolbox call or set global variables. If another PPC Toolbox call is made from a completion routine, then the PPCRead function must use a record of data type PPCParamBlockRec instead of type PPCReadPBRec.

Listing 11-15 illustrates a function that can be used to poll the ioResult field of a record of data type PPCReadPBRec. The function returns TRUE when the PPCRead function associated with PPCReadPBRec has completed.

Listing 11-15 Polling the ioResult field to determine if a PPCRead function has completed

FUNCTION MyReadComplete(thePPCReadPBPtr: PPCReadPBPtr; VAR err: OSErr): Boolean;
BEGIN
  err := thePPCReadPBPtr^.ioResult;
  MyReadComplete := err <> 1;
END;

Using the PPC Toolbox
CHAPTER 11

Program-to-Program Communications Toolbox

Sending Data to an Application

Use the PPCWrite function to send a message block during a session specified by the session reference number.

You should call the PPCWrite function asynchronously. You can provide a completion routine that will be called when the PPCWrite function has completed, or you can poll the ioResult field of the PPC parameter block to determine whether the PPCWrite function has completed. A PPCWrite completion routine can issue another PPC Toolbox call or set global variables. If another PPC Toolbox call is made from a completion routine, then the PPCWrite function must use a record of data type PPCParamBlockRec instead of type PPCWritePBRec. Note that message blocks are sent in the order in which they are written.

Listing 11-16 illustrates how you use the PPCWrite function to write data during a session.

Listing 11-16 Using the PPCWrite function to write data during a session

FUNCTION MyPPCWrite(thePPCWritePBPtr: PPCWritePBPtr;
  theSessRefNum: PPCSessRefNum;
  theBufferLength: Size;
  theBufferPtr: Ptr): OSErr;
BEGIN
  WITH thePPCWritePBPtr^ DO
  BEGIN
    ioCompletion := NIL;
    sessRefNum := theSessRefNum; {from PPCStart or PPCInform}
    bufferLength := theBufferLength;
    bufferPtr := theBufferPtr;
    more := FALSE; {no more data to read}
    userData := 0; {application-specific data}
    blockCreator := '????'; {application-specific data}
    blockType := '????'; {application-specific data}
  END;
  MyPPCWrite := PPCWrite(thePPCWritePBPtr, TRUE); {asynchronous}
END;

The first PPCWrite function that you use to create a new message block sets the block creator, block type, and user data attributes for the block. These attributes are returned to the application when it reads from the message block. Set the more field to FALSE to indicate the end of the message block or set this field to TRUE if you want to append additional data to a message block.
Listing 11-17 illustrates a function that can be used to poll the `ioResult` field of a record of data type `PPCWritePBRec`. The function returns `TRUE` when the `PPCWrite` function associated with `PPCWritePBRec` has completed.

```
FUNCTION MyWriteComplete(thePPCWritePBPtr: PPCWritePBPtr; VAR err: OSErr): Boolean;
BEGIN
  err := thePPCWritePBPtr^.ioResult;
  MyWriteComplete := err <> 1;
END;
```

**Ending a Session and Closing a Port**

After data is written and read in, use the `PPCEnd` function to end the session (identified by the session reference number). You may receive an error if you use the `PPCEnd` function to end a session that has already been terminated.

Listing 11-18 illustrates how you use the `PPCEnd` function to end a session.

```
FUNCTION MyPPCEnd(theSessRefNum: PPCSessRefNum): OSErr;
VAR
  thePPCEndPBRec: PPCEndPBRec;
BEGIN
  thePPCEndPBRec.sessRefNum := theSessRefNum;
  MyPPCEnd := PPCEnd(@thePPCEndPBRec, FALSE); {synchronous}
END;
```

The `PPCEnd` function causes all calls to the `PPCRead` and `PPCWrite` functions to complete (with a `sessClosedErr` result code) and invalidates the session reference number. The `PPCEnd` function also releases any PPC Toolbox resources so that they can be reused.

Use the `PPCClose` function to close the port specified by the port reference number. When you close a port, all sessions associated with a port are ended. Any active asynchronous calls associated with a session then call their completion routines (if they have one).
CHAPTER 11

Program-to-Program Communications Toolbox

Listing 11-19 illustrates how you use the PPCClose function to close a port.

Listing 11-19  Closing a PPC port using the PPCClose function

FUNCTION MyPPCClose(thePortRefNum: PPCPortRefNum): OSErr;
VAR
  theClosePBRec: PPCClosePBRec;
BEGIN
  theClosePBRec.portRefNum := thePortRefNum;  {from PPCOpen}
  MyPPCClose := PPCClose(@theClosePBRec, FALSE);  {synchronous}
END;

In this example, the call to PPCClose is made synchronously.

Invalidating Users

It is your responsibility to invalidate all user reference numbers obtained with the StartSecureSession function before your application quits. However, while your application remains open, you may want to keep track of a user reference number to start a session with a port, end it, and then later start another session with the same port. Use the DeleteUserIdentity function to invalidate the user reference number for a particular user.

err := DeleteUserIdentity (userRef);

The DeleteUserIdentity function removes a user by invalidating the specified user reference number. Note that you cannot invalidate the guest reference number (0) and, in most cases, you should not dispose of the default user reference number.
Listing 11-20 illustrates how you use the `DeleteUserIdentity` function to invalidate a user reference number obtained from a `StartSecureSession` function. The sample code does not invalidate the user reference number if it is either the default user reference number or the guest reference number (0).

```
FUNCTION MyDeleteNewUserRefNum(newUserRef: LongInt): OSErr;
VAR
  err: OSErr;
  defUserRef: LongInt;
  defUserName: Str32;
BEGIN
  IF newUserRef <> 0 THEN
    BEGIN {user reference number passed was not the guest}
      err := GetDefaultUser(defUserRef, defUserName);
      IF err = noErr THEN
        BEGIN {there is a default user}
          IF newUserRef <> defUserRef THEN
            {new user ref number isn't the default user ref num, } 
            { so ok to delete}
            err := DeleteUserIdentity(newUserRef);
          END
        ELSE {there is no default, so delete new user ref num}
          err := DeleteUserIdentity(newUserRef);
          MyDeleteNewUserRefNum := err;
        END
      ELSE {there is no default, so delete new user ref num}
        err := DeleteUserIdentity(newUserRef);
        MyDeleteNewUserRefNum := err;
      END
    END
  ELSE {user reference number passed was the guest}
    MyDeleteNewUserRefNum := noErr;
  END;
```
PPC Toolbox Reference

This section describes the data structures and routines that are specific to the PPC Toolbox. The section “PPC Toolbox Routines” beginning on page 11-51 describes PPC Toolbox routines. “Application-Defined Routines” beginning on page 11-78 describes completion routines and port filter functions.

Data Structures

This section describes the PPC parameter block, PPC port record, location name record, and port information record.

The PPC Toolbox Parameter Block

PPC Toolbox functions require a pointer to a PPC parameter block. You must fill out any fields of the parameter block that the specific PPC Toolbox function requires.

```
TYPE PPCParamBlockRec = RECORD
  CASE Integer OF
    0: (openParam: PPCOpenPBRec);  {PPCOpen params}
    1: (informParam: PPCInformPBRec);  {PPCInform params}
    2: (startParam: PPCStartPBRec);  {PPCStart params}
    3: (acceptParam: PPCAcceptPBRec);  {PPCAccept params}
    4: (rejectParam: PPCRejectPBRec);  {PPCReject params}
    5: (writeParam: PPCWritePBRec);  {PPCWrite params}
    6: (readParam: PPCReadPBRec);  {PPCRead params}
    7: (endParam: PPCEndPBRec);  {PPCEnd params}
    8: (closeParam: PPCClosePBRec);  {PPCClose params}
    9: (listPortsParam: IPCListPortsPBRec);  {IPCListPorts}
      { params}
  END;
```

Figure 11-18 on the next page shows the PPC Toolbox parameter blocks. Note that the reserved fields are not included in the illustration. The qLink, csCode, intUse, intUsePtr, and reserved fields are used internally by the PPC Toolbox. Your application should not rely on the PPC Toolbox to preserve these fields across calls.

Your application transfers ownership of the PPC Toolbox parameter block (and any buffers or records pointed to by the PPC Toolbox parameter block) to the PPC Toolbox until a PPC function is complete. Once the function completes, ownership of the parameter block (and any buffers or records it points to) is transferred back to your application. If a PPC Toolbox function is executed asynchronously, your program cannot alter memory that might be used by the PPC Toolbox until that function completes.
### Figure 11-18  The PPC Toolbox parameter blocks

<table>
<thead>
<tr>
<th>Offset</th>
<th>PPC Toolbox Reference</th>
<th>Other</th>
<th>portRefNum</th>
<th>portRefNum</th>
<th>portRefNum</th>
<th>portRefNum</th>
<th>portRefNum</th>
<th>filler1</th>
<th>filler1</th>
<th>filler1</th>
<th>filler1</th>
<th>filler1</th>
<th>filler1</th>
<th>filler1</th>
<th>filler1</th>
<th>filler1</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td></td>
<td></td>
<td>portRefNum</td>
<td>portRefNum</td>
<td>portRefNum</td>
<td>portRefNum</td>
<td>portRefNum</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
<td>filler1</td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
<td>serviceType</td>
<td>serviceType</td>
<td>serviceType</td>
<td>serviceType</td>
<td>serviceType</td>
<td>filler2</td>
<td>bufferLength</td>
<td>bufferLength</td>
<td>filler2</td>
<td>bufferLength</td>
<td>bufferLength</td>
<td>filler2</td>
<td>filler2</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td></td>
<td>resFlag</td>
<td>resFlag</td>
<td>resFlag</td>
<td>resFlag</td>
<td>resFlag</td>
<td>filler3</td>
<td>filler3</td>
<td>filler3</td>
<td>filler3</td>
<td>filler3</td>
<td>filler4</td>
<td>filler4</td>
<td>filler4</td>
<td>filler4</td>
</tr>
<tr>
<td>46</td>
<td></td>
<td></td>
<td>portName</td>
<td>portName</td>
<td>portName</td>
<td>portName</td>
<td>portName</td>
<td>filler4</td>
<td>bufferPtr</td>
<td>bufferPtr</td>
<td>bufferPtr</td>
<td>bufferPtr</td>
<td>bufferPtr</td>
<td>bufferPtr</td>
<td>bufferPtr</td>
<td>bufferPtr</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td>locationName</td>
<td>locationName</td>
<td>locationName</td>
<td>locationName</td>
<td>locationName</td>
<td>filler4</td>
<td>filler4</td>
<td>filler4</td>
<td>filler4</td>
<td>filler4</td>
<td>locationName</td>
<td>locationName</td>
<td>locationName</td>
<td>locationName</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td>locationName</td>
<td>locationName</td>
<td>locationName</td>
<td>locationName</td>
<td>locationName</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
<td>networkVisible</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rejectInfo</td>
<td>rejectInfo</td>
<td>rejectInfo</td>
<td>rejectInfo</td>
<td>rejectInfo</td>
<td>rejectInfo</td>
<td>rejectInfo</td>
<td>rejectInfo</td>
<td>rejectInfo</td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A PPC Toolbox function that is executed asynchronously must specify NIL or the address of a completion routine in the ioCompletion field of the PPC parameter block. The ioResult field should be used to determine the actual result code when an asynchronously executed PPC Toolbox function completes. If you specify a completion routine in the ioCompletion field, it is called at interrupt time when the PPC Toolbox function completes execution. See page 11-78 for the routine declaration for a completion routine.

The PPC Port Record

A PPC port name is defined by a PPC port record. The PPCPortRec data type defines the PPC port record.

```pascal
TYPE PPCPortRec = RECORD
  nameScript: ScriptCode;  {script identifier}
  name: Str32;  {port name in program }
    { linking dialog box}
  portKindSelector: PPCPortKinds;  {general category of }
    { application}
  CASE PPCPortKinds OF
    ppcByString: (portTypeStr: Str32);
    ppcByCreatorAndType:
      (portCreator: OSType;
        portType: OSType);
END;
```

Field descriptions

- **nameScript**: An integer script code.
- **name**: A string that designates the application name.
- **portKindSelector**: An integer that selects the kind of type string (either ppcByString or ppcByCreatorAndType).
- **portTypeStr**: If the portKindSelector field specifies ppcByString, the portTypeStr field contains a 32-byte character string.
- **portCreator**: If the portKindSelector field specifies ppcByCreatorAndType, the portCreator field contains a 4-character creator code.
- **portType**: If the portKindSelector field specifies ppcByCreatorAndType, the portType field contains a 4-character type code.

To open a port, you need to specify a port name. As previously described, a port name consists of a script code, a name string, and a type string. For example, you can designate “smRoman” as the script code, “make memo” as the application’s name string, and “word processor” as its type string.
CHAPTER 11

Program-to-Program Communications Toolbox

The Location Name Record

A location name identifies the location of a computer on the network. A location
name is specified in the standard Name-Binding Protocol (NBP) form,
<object string>:PPCToolBox @<AppleTalk zone>. The object string is the name provided in
the Sharing Setup control panel in the Control Panels folder. By default, the type string is
"PPCToolBox". The AppleTalk zone is the zone to which the particular Macintosh
computer belongs. For example, “Jane Doe's Macintosh:PPCToolBox@twilight” specifies
the object string, type string, and AppleTalk zone for a particular computer.

The LocationNameRec data type defines the location name record. The
locationKindSelector field can be set to ppcNoLocation, ppcNBPLocation, or
ppcNBPTypeLocation.

TYPE LocationNameRec =
RECORD
    locationKindSelector: PPCLocationKind; {which variant}
    CASE PPCLocationKind OF
        {ppcNoLocation: storage not used by this value}
        ppcNBPLocation:
            (nbpEntity: EntityName); {NBP name entity}
        ppcNBPTypeLocation:
            (nbpType: Str32); {just the NBP type }
                { string for the } 
                { PPCOpen function}
    END;

Field descriptions

locationKindSelector
An integer that determines how the location is specified.
You can use either of the constants ppcNBPLocation
or ppcNBPTypeLocation. (The PPC Toolbox uses the constant
ppcNoLocation when the location received from or passed to a
PPC Toolbox function is the location of the local machine.)
nbpEntity
If the locationKindSelector field specifies ppcNBPLocation,
the nbpEntity field specifies a full NBP entity name.
nbpType
If the locationKindSelector field specifies
ppcNBPTypeLocation, the nbpType field specifies an alias
location name. This location kind is used only by the PPCOpen
function when an alias location name is needed.

Note
You should assign an NBP value directly—do not pack it using
nbpSetEntity. ◆
The Port Information Record

A port information record identifies whether a particular port requires authentication and specifies the port's port name. Both the PPCBrowser and IPCListPorts functions return information about ports using port information records. In addition, if you provide a port filter function, the PPC Toolbox provides information to your function about the current port in a port information record. The PortInfoRec data type defines a port information record.

```pascal
TYPE PortInfoRec =
  RECORD
    filler1: SignedByte;    {space holder}
    authRequired: Boolean; {authentication required}
    name: PPCPortRec;      {port name}
  END;
```

Field descriptions

- **filler1**: Reserved.
- **authRequired**: Specifies whether the port requires authentication. This field is `TRUE` if the port requires authentication before a session can begin. Otherwise, this field is `FALSE`.
- **name**: Specifies an available port name.

For information on the PPCBrowser and IPCListPorts functions, see page 11-52 and page 11-54, respectively. For information on port filter functions, see page 11-78.

PPC Toolbox Routines

This section describes the routines for

- initializing the PPC Toolbox
- displaying the program linking dialog box
- listing available ports
- opening and closing a port
- starting and ending a session
- accepting and rejecting a session
- reading and writing data
- obtaining the default user reference number and name
- invalidating a user reference number

Result codes appear after each function where applicable.
Initializing the PPC Toolbox

You use the PPCInit function to initialize the PPC Toolbox.

PPCInit

Use the PPCInit function to initialize the PPC Toolbox.

FUNCTION PPCInit: OSErr;

DESCRIPTION

After initialization, most PPC Toolbox routines can execute either synchronously or asynchronously.

Note that a noGlobalsErr result code indicates that the PPC Toolbox is not loaded properly.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the PPCInit function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPCBrowser</td>
<td>$0000</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are

<table>
<thead>
<tr>
<th>Registers on entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0     Selector code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Registers on exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0     Result code</td>
</tr>
</tbody>
</table>

RESULT CODES

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>noGlobalsErr</td>
<td>-904</td>
<td>System unable to allocate memory, critical error</td>
</tr>
</tbody>
</table>

Using the Program Linking Dialog Box

You can use either the PPCBrowser function or the IPCListPorts function to locate a port to communicate with. Use the PPCBrowser function to display the program linking dialog box. For the description of IPCListPorts, see page 11-54.
Use the **PPCBrowser** function to display the program linking dialog box, which allows a user to select a port to communicate with.

```pascal
FUNCTION PPCBrowser (prompt: Str255; applListLabel: Str255;
defaultSpecified: Boolean;
VAR theLocation: LocationNameRec;
VAR thePortInfo: PortInfoRec;
portFilter: PPCFilterProcPtr;
theLocNBPTpye: Str32): OSErr;
```

**prompt**  
A line of text that the **PPCBrowser** function displays as a prompt in the program linking dialog box. If you specify **NIL** or an empty string is passed, the default prompt “Choose a program to link to:” is used.

**applListLabel**  
The title of the list of PPC ports. If you specify **NIL** or an empty string is passed, the default title “Programs” is used.

**defaultSpecified**  
A value that determines which port is initially selected in the program linking dialog box. If you specify **TRUE**, you must provide information in the parameters **theLocation** and **thePortInfo**. In this case, the **PPCBrowser** function tries to select the PPC port specified by the parameters **theLocation** and **thePortInfo** when the program linking dialog box first appears. If you specify **FALSE**, the **PPCBrowser** function selects the first port in the list and you can leave the location name record and the port information record (in the parameters **theLocation** and **thePortInfo**) uninitialized.

**theLocation**  
The port location. For information on this data structure, see “The Location Name Record” on page 11-49.

**thePortInfo**  
The port name. For information on this data structure, see “The Port Information Record” on page 11-50.

**portFilter**  
Determines how the list of PPC ports is filtered. If this parameter is **NIL**, the names of all existing PPC ports are displayed. If this parameter isn’t **NIL**, it must be a pointer to a port filter function.

**theLocNBPTpye**  
The NBP type passed to **NBPLookup** to generate the list of computers. If you specify **NIL** or an empty string is passed, the default, “PPCToolBox”, is used.
DESCRIPTION

The `PPCBrowser` function builds the list of ports and then displays the program linking dialog box.

If you set the `defaultSpecified` parameter to `TRUE`, the `PPCBrowser` function tries to select the PPC port specified by the parameters `theLocation` and `thePortInfo` when the program linking dialog box first appears. The `locationKindSelector` field in the location name record must be set to the `ppcNoLocation` constant (which specifies the local computer) or the `ppcNBPLocation` constant (which specifies the NBP object and NBP zone). The `ppcNBPTypeLocation` constant is not supported for matching. When matching the location, only the object string and the zone string of the entity name are used—the type string is ignored. When matching the port, the entire PPC port record (script, name, and port type) is used in the port information record. The `authRequired` field of the port information record is ignored.

The parameter `theLocNBPType` of the `PPCBrowser` function specifies the NBP type passed to `NBPLookup` to generate the list of computers. If you specify `NIL` or an empty string is passed, the default, “PPCToolBox”, is used. Note that the current computer is always included in the list of computers (even if a location with the specified type does not exist for it). If the parameter `theLocNBPType` contains either of the NBP wildcard characters (= or ≈), the `PPCBrowser` function returns a `paramErr` result code.

If the `PPCBrowser` function returns `noErr`, the parameters `theLocation` and `thePortInfo` specify the port chosen by the user. If the `PPCBrowser` function returns a `userCanceledErr` result code, the user clicked the Cancel button, and no port was selected. If the function returns a `memFullErr` result code, there was not enough memory to load the `PPCBrowser` package, and the dialog box did not appear.

**Note**

You must not call the `PPCBrowser` function from an application that is running in the background, since this function displays a dialog box on the user’s screen.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>noErr</code></td>
<td>0 No error</td>
</tr>
<tr>
<td><code>paramErr</code></td>
<td>-50 Illegal parameter</td>
</tr>
<tr>
<td><code>memFullErr</code></td>
<td>-108 Not enough memory to load PPCBrowser package</td>
</tr>
<tr>
<td><code>userCanceledErr</code></td>
<td>-128 User decided not to conduct a session</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of the `PPCBrowser` function, see Listing 11-4 on page 11-26. For an example of the program linking dialog box, see Figure 11-12 on page 11-22. For information on port filter functions, see page 11-78.
Obtaining a List of Ports

Use the IPCListPorts function to generate a list of existing ports without displaying a dialog box. The IPCListPortsPBRec data type defines the parameter block used by the IPCListPorts function.

**IPCLlistPorts**

Use the IPCListPorts function to generate a list of existing ports without displaying a dialog box.

```
FUNCTION IPCListPorts (pb: IPCListPortsPBPtr;
                      async: Boolean): OSErr;
```

- **pb** A pointer to an IPCListPorts parameter block.
- **async** A value that specifies whether the function is to be executed asynchronously (TRUE) or synchronously (FALSE).

### Parameter block

- **ioCompletion** PPCCompProcPtr Address of a completion routine
- **ioResult** OSErr Result code
- **startIndex** Integer Index to the port entry list
- **requestCount** Integer Number of port names requested
- **actualCount** Integer Number of port names returned
- **portName** PPCPortPtr Pointer to a PPCPortRec
- **locationName** LocationNamePtr Pointer to a LocationNameRec
- **bufferPtr** PortInfoArrayPtr Pointer to an array of PortInfoRec

### DESCRIPTION

If your application calls the IPCListPorts function asynchronously, you must specify in the ioCompletion field either the address of a completion routine or NIL. If you set ioCompletion to NIL, you should poll the ioResult field of the PPC parameter block (from your application’s main event loop) to determine whether the PPC Toolbox has completed the requested operation. A value in the ioResult field other than 1 indicates that the call is complete. Note that it is unsafe to poll the ioResult field at interrupt time since the PPC Toolbox may be in the process of completing a call. See “PPC Toolbox Calling Conventions” beginning on page 11-14 for detailed information.

If you call the IPCListPorts function asynchronously, you must not change any of the fields in the parameter block until the call completes. The port name, location name, and buffer pointed to by IPCListPortsPBRec are owned by the PPC Toolbox until the call completes. These objects must not be deallocated or moved in memory while the call is in progress.
CHAPTER 11

Program-to-Program Communications Toolbox

The `startIndex` field specifies the index to the list of ports on the remote machine from which the PPC Toolbox begins to get the list. In most cases, you'll want to start at the beginning, so set the `startIndex` field to 0. The `requestCount` field specifies the maximum number of port information records that can fit into your buffer.

The `actualCount` field returns the actual number of entries returned. Your program can use the `IPCListPorts` function repeatedly to obtain the entire list of ports. Ports that are not visible to the network are not included in the ports listing on a remote machine. (If you specify `FALSE` for the `networkVisible` field in the `PPCOpen` function, the port is not included in the listing of available ports across a network.)

The `portName` field must contain a pointer to a PPC port record that specifies which PPC ports to list. You can specify particular values in the PPC port record or you can use an equal sign (=) in the name or the `portTypeStr` fields as a wildcard to match all port names or port types.

The `locationName` field should contain a pointer to a location name record that designates the computer that contains the PPC ports you want returned. If the `locationKindSelector` field in the location name record is `ppcNoLocation` or if the `locationName` pointer is `NIL`, then the location is the local machine. If the `locationKindSelector` field in the location name record is `ppcNBPLocation`, then the location is a remote machine designated by the location name record's `nbpEntity` field.

The `IPCListPorts` function returns an array (list) of port information records in the area of memory pointed to by `bufferPtr`. Make sure that the buffer pointed to by the `bufferPtr` field is at least `sizeof(PortInfoRec) * requestCount`.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `IPCListPorts` function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPC</td>
<td>$000A</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are

**Registers on entry**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>Pointer to a parameter block</td>
</tr>
<tr>
<td>D0</td>
<td>Selector code</td>
</tr>
</tbody>
</table>

**Registers on exit**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>Result code</td>
</tr>
</tbody>
</table>
### RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>notInitErr</td>
<td>-900 PPC Toolbox has not been initialized yet</td>
</tr>
<tr>
<td>nameTypeErr</td>
<td>-902 Invalid or inappropriate locationKindSelector in location name</td>
</tr>
<tr>
<td>noGlobalsErr</td>
<td>-904 System unable to allocate memory, critical error</td>
</tr>
<tr>
<td>localOnlyErr</td>
<td>-905 Network activity is currently disabled</td>
</tr>
<tr>
<td>sessTableErr</td>
<td>-907 PPC Toolbox is unable to create a session</td>
</tr>
<tr>
<td>noResponseErr</td>
<td>-915 Unable to contact application</td>
</tr>
<tr>
<td>badPortNameErr</td>
<td>-919 PPC port record is invalid</td>
</tr>
<tr>
<td>networkErr</td>
<td>-925 An error has occurred in the network</td>
</tr>
<tr>
<td>badLocNameErr</td>
<td>-931 Location name is invalid</td>
</tr>
</tbody>
</table>

### SEE ALSO

For an example of the use of the `IPCLeakPorts` function, see Listing 11-5 on page 11-28.

### Opening and Closing a Port

You open a port using the `PPCOpen` function and close a port using the `PPCClose` function.

### PPCOpen

You open a port using the `PPCOpen` function.

**FUNCTION**

```
FUNCTION PPCOpen (pb: PPCOpenPBPtr; async: Boolean): OSErr;
```

- **pb** A pointer to a `PPCOpen` parameter block.
- **async** A value that specifies whether the function is to be executed asynchronously (`TRUE`) or synchronously (`FALSE`).

#### Parameter block

- `ioCompletion` PPCCompProcPtr Address of a completion routine
- `ioResult` OSErr Result code
- `portRefNum` PPCPortRefNum Port reference number of port opened
- `serviceType` PPCServiceType Service type requested—must be `ppcServiceRealTime`
- `resFlag` SignedByte Reserved field—must be 0
- `portName` PPCPortPtr Pointer to a `PPCPortRec`
- `locationName` LocationNamePtr Pointer to a `LocationNameRec`
- `networkVisible` Boolean Make this port network visible
- `nbpRegistered` Boolean Port location was registered on the network
DESCRIPTION

If your application calls the `PPCOpen` function asynchronously, you must specify in the `ioCompletion` field either the address of a completion routine or `NIL`. If you set `ioCompletion` to `NIL`, you should poll the `ioResult` field of the PPC parameter block (from your application’s main event loop) to determine whether the PPC Toolbox has completed the requested operation. A value in the `ioResult` field other than 1 indicates that the call is complete. Note that it is unsafe to poll the `ioResult` field at interrupt time since the PPC Toolbox may be in the process of completing a call. See “PPC Toolbox Calling Conventions” beginning on page 11-14 for detailed information.

If you call the `PPCOpen` function asynchronously, you must not change any of the fields in the parameter block until the call completes. The port name and location name pointed to by the `PPCOpen` parameter block record are owned by the PPC Toolbox until the call completes. These objects must not be deallocated or moved in memory while the call is in progress.

The `portRefNum` field returns the PPC port identifier. Use this port reference number to initiate a session for this particular port. Set the `serviceType` field to indicate that this port accepts sessions in real time. For System 7, this field must always be set to the `ppcServiceRealTime` constant. You must set the `resFlag` field to 0.

The `portName` field must contain a pointer to a PPC port record that specifies the name of the PPC port to be opened.

The `locationName` field should contain a pointer to a location name record that designates the location of the PPC port to be opened. If the `locationName` pointer is `NIL`, then the default name PPC Toolbox is used. If a location name record is used, then the `locationKindSelector` field in the location name record must be `ppcNBPTypeLocation`, and an alias location name specified by the location name record’s `nbpType` field is used.

The `networkVisible` field indicates whether the port should be made visible (for browsing as well as incoming network requests). If you specify `FALSE`, this port is not visible in the listing of available ports across a network (although it is still included within the local machine’s listing of available ports).

The `nbpRegistered` field returns `TRUE` if the location name specified was registered on the network.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PPCOpen` function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPC</td>
<td>$0001</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are

**Registers on entry**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>Pointer to a parameter block</td>
</tr>
<tr>
<td>D0</td>
<td>Selector (1)</td>
</tr>
</tbody>
</table>
PPCClose

You use the PPCClose function to close the port specified by the port reference number.

FUNCTION PPCClose (pb: PPCClosePBPtr; async: Boolean): OSErr;

pb A pointer to a PPCClose parameter block.
async A value that specifies whether the function is to be executed asynchronously (TRUE) or synchronously (FALSE).

Parameter block
→ ioCompletion PPCCompProcPtr Address of a completion routine
← ioResult OSErr Result code
→ portRefNum PPCPortRefNum Port reference number of port to close

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>noErr No error</td>
</tr>
<tr>
<td>-900</td>
<td>notInitErr PPC Toolbox has not been initialized yet</td>
</tr>
<tr>
<td>-902</td>
<td>nameTypeErr Invalid or inappropriate locationKindSelector in location name</td>
</tr>
<tr>
<td>-903</td>
<td>noPortErr Unable to open port or bad port reference number</td>
</tr>
<tr>
<td>-904</td>
<td>noGlobalsErr System unable to allocate memory, critical error</td>
</tr>
<tr>
<td>-909</td>
<td>badReqErr Bad parameter or invalid state for this operation</td>
</tr>
<tr>
<td>-910</td>
<td>portNameExistsErr Another port is already open with this name</td>
</tr>
<tr>
<td>-919</td>
<td>badPortNameErr PPC port record is invalid</td>
</tr>
<tr>
<td>-930</td>
<td>badServiceMethodErr Service method is other than ppcServiceRealTime</td>
</tr>
<tr>
<td>-931</td>
<td>badLocNameErr Location name is invalid</td>
</tr>
<tr>
<td>-1027</td>
<td>nbpDuplicateName Location name represents a duplicate on this computer</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of the PPCOpen function, see Listing 11-2 on page 11-21.
CHAPTER 11

Program-to-Program Communications Toolbox

DESCRIPTION

If your application calls this function asynchronously, you must specify in the 
ioCompletion field either the address of a completion routine or NIL. If you 
set ioCompletion to NIL, you should poll the ioResult field of the PPC parameter 
block (from your application’s main event loop) to determine whether the PPC Toolbox 
has completed the requested operation. A value in the ioResult field other than 1 
indicates that the call is complete. Note that it is unsafe to poll the ioResult field at 
interrupt time since the PPC Toolbox may be in the process of completing a call. See 
“PPC Toolbox Calling Conventions” beginning on page 11-14 for detailed information.
The portRefNum field specifies the PPC port identifier of the port to close. The port 
reference number must be a valid port reference number returned from a previous call to 
the PPCOpen function.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the PPCClose function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPC</td>
<td>$0009</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are

Registers on entry

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>Pointer to a parameter block</td>
</tr>
<tr>
<td>D0</td>
<td>Selector code</td>
</tr>
</tbody>
</table>

Registers on exit

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>Result code</td>
</tr>
</tbody>
</table>

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>notInitErr</td>
<td>–900 PPC Toolbox has not been initialized yet</td>
</tr>
<tr>
<td>noPortErr</td>
<td>–903 Bad port reference number</td>
</tr>
<tr>
<td>noGlobalsErr</td>
<td>–904 System unable to allocate memory, critical error</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of the PPCClose function, see Listing 11-19 on page 11-44.

Starting and Ending a Session

You use the PPCStart or StartSecureSession function to initiate a session with 
another port, and you use the PPCEnd function to end a session.
The PPCStart function initiates a session with the destination port specified in the name and location fields.

FUNCTION PPCStart (pb: PPCStartPBPtr; async: Boolean): OSErr;

pb A pointer to a PPCStart parameter block.
async A value that specifies whether the function is to be executed asynchronously (TRUE) or synchronously (FALSE).

Parameter block

| →   | ioCompletion | PPCCompProcPtr | Address of a completion routine |
| ←   | ioResult     | OSErr          | Result code                     |
| →   | portRefNum   | PPCPortRefNum  | Port reference number of this session |
| ←   | sessRefNum   | PPCSessRefNum  | Session reference number of this session |
| →   | serviceType  | PPCServiceType | Service type requested—must be ppcServiceRealTime |
| →   | resFlag      | SignedByte    | Reserved field—must be 0         |
| →   | portName     | PPCPortPtr    | Pointer to a PPCPortRec         |
| ←   | locationName | LocationNamePtr | Pointer to a LocationNameRec   |
| ←   | rejectInfo   | LongInt       | Value from PPCReject if session was rejected |
| →   | userData     | LongInt       | Application-specific data       |
| →   | userRefNum   | LongInt       | User reference number           |

DESCRIPTION

If your application calls the PPCStart function asynchronously, you must specify in the ioCompletion field either the address of a completion routine or NIL. If you set ioCompletion to NIL, you should poll the ioResult field of the PPC parameter block (from your application’s main event loop) to determine whether the PPC Toolbox has completed the requested operation. A value in the ioResult field other than 1 indicates that the call is complete. Note that it is unsafe to poll the ioResult field at interrupt time, since the PPC Toolbox may be in the process of completing a call. See “PPC Toolbox Calling Conventions” beginning on page 11-14 for detailed information.

If you call the PPCStart function asynchronously, you must not change any of the fields in the parameter block until the call completes. The port name and location name pointed to by the PPCStart parameter block record are owned by the PPC Toolbox until the call completes. These objects must not be deallocated or moved in memory while the call is in progress.
You specify the PPC port identifier in the `portRefNum` field. The port reference number is a reference number for the port through which you are requesting a session. The value you specify must correspond to the port reference number returned from the `PPCopen` function.

The `sessRefNum` field returns a session identifier. This number, which is provided by the PPC Toolbox, is used while data is being exchanged to identify a particular session. You must set the `serviceType` field to indicate that the session is to be connected in real time. For System 7, this field must always be set to the `ppcServiceRealTime` constant. You must set the `resFlag` field to 0.

The `portName` field must contain a pointer to a PPC port record. The `locationName` field should contain a pointer to a location name record or `NIL`. The PPC port record and the location name record specify the name and location of the PPC port to initiate a session with, and they are usually obtained from the `PPCBrowser` function. If the `locationKindSelector` field in the location name record is `ppcNoLocation` or if the `locationName` pointer is `NIL`, then the location is the local machine. If the `locationKindSelector` field in the location name record is `ppcNBPLocation`, then the location is a remote machine designated by the location name record's `nbpEntity` field.

If the `ioResult` field of the PPC parameter block returns a `userRejectErr` result code, the `rejectInfo` field contains the same value as the `rejectInfo` field in the `PPCReject` parameter block. The `rejectInfo` field is defined by your application.

The initiating port can specify any information in the `userData` field. The `PPCInform` function reports this data to the responding port upon its completion.

The `userRefNum` field specifies an authenticated user. The authentication mechanism of the PPC Toolbox identifies each user through an assigned name and a password. A user reference number of 0 indicates that you want to specify a guest.

**ASSEMBLY-LANGUAGE INFORMATION**

The trap macro and routine selector for the `PPCStart` function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPC</td>
<td>$0002</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are

**Registers on entry**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>Pointer to a parameter block</td>
</tr>
<tr>
<td>D0</td>
<td>Selector code</td>
</tr>
</tbody>
</table>

**Registers on exit**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>Result code</td>
</tr>
</tbody>
</table>
CHAPTER 11

Program-to-Program Communications Toolbox

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>No error</td>
</tr>
<tr>
<td>notInitErr</td>
<td>PPC Toolbox has not been initialized yet</td>
</tr>
<tr>
<td>nameTypeErr</td>
<td>locationKindSelector is not ppcNBPLocation or ppcNoLocation</td>
</tr>
<tr>
<td>noPortErr</td>
<td>Bad port reference number</td>
</tr>
<tr>
<td>noGlobalsErr</td>
<td>System unable to allocate memory, critical error</td>
</tr>
<tr>
<td>localOnlyErr</td>
<td>Network activity is currently disabled</td>
</tr>
<tr>
<td>destPortErr</td>
<td>Port does not exist at destination</td>
</tr>
<tr>
<td>sessTableErr</td>
<td>PPC Toolbox is unable to create a session</td>
</tr>
<tr>
<td>noUserNameErr</td>
<td>PPC Toolbox is unable to create a session</td>
</tr>
<tr>
<td>userRejectErr</td>
<td>Destination rejected the session request</td>
</tr>
<tr>
<td>noResponseErr</td>
<td>Unable to contact application</td>
</tr>
<tr>
<td>portClosedErr</td>
<td>The port was closed</td>
</tr>
<tr>
<td>badPortNameErr</td>
<td>PPC port record is invalid</td>
</tr>
<tr>
<td>networkErr</td>
<td>An error has occurred in the network</td>
</tr>
<tr>
<td>noInformErr</td>
<td>PPCStart failed because target application did not have an inform pending</td>
</tr>
<tr>
<td>authFailErr</td>
<td>User’s password is wrong</td>
</tr>
<tr>
<td>noUserRecErr</td>
<td>Invalid user reference number</td>
</tr>
<tr>
<td>badServiceMethodErr</td>
<td>Service method is other than ppcServiceRealTime</td>
</tr>
<tr>
<td>guestNotAllowedErr</td>
<td>Destination port requires authentication</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of the PPCStart function, see Listing 11-7 on page 11-34.

StartSecureSession

The StartSecureSession function prompts for user name and password and calls PPCStart—all in one synchronous procedure call. Use the StartSecureSession function whenever a port destination requires authentication.

FUNCTION StartSecureSession (pb: PPCStartPBPtr;
    VAR userName: Str32;
    useDefault: Boolean;
    allowGuest: Boolean;
    VAR guestSelected: Boolean;
    prompt: Str255): OSErr;

pb A pointer to a PPCStart parameter block.
userName A pointer to a 32-byte character string to be displayed as the user’s name.
useDefault A Boolean value that indicates whether you want the StartSecureSession function to use the default user identity (and possibly prevent the user identity dialog box from appearing). If so, specify TRUE; otherwise, specify FALSE.
allowGuest
A Boolean value that determines whether the Guest radio button in the
user identity dialog box is active (TRUE) or inactive (FALSE).

guestSelected
Returns TRUE if the user has logged on as a guest.

prompt
A line of text that the dialog box displays in place of the default prompt.
Specify NIL or an empty string to use the default prompt.

DESCRIPTION
Your program fills out a parameter block just as though it were calling the PPCStart
function. You specify all input fields in the parameter block except for the userRefNum
field. The userRefNum field is returned when the StartSecureSession function
successfully completes.

The userName parameter is a pointer to a 32-byte character string to be displayed as
the user’s name. If the Pascal string length is 0, the default user name is used. The
default user name is the name specified in the Sharing Setup control panel. The default
user name is returned in the userName buffer.

Set the useDefault parameter to TRUE if you want the StartSecureSession
function to use the default user identity (and possibly prevent the user identity dialog
box from appearing). The allowGuest parameter specifies whether the Guest radio
button in the user identity dialog box is active. You usually set it to the inverse of the
authRequired field in the port information record. For example, if authRequired is
TRUE, then allowGuest should be set to FALSE.

The guestSelected parameter returns TRUE if the user has logged on as a guest. The
prompt parameter of the StartSecureSession function allows you to specify a line
of text that the dialog box can display. Specify NIL or an empty string for the prompt
parameter to enable the PPC Toolbox to use the default prompt. The PPC Toolbox uses
the default string “Link to <port name> on <object string> as:”. The port name is obtained
from the name string of the port name, and the object string is obtained from the object
string of the location name.

Note
Do not call the StartSecureSession function from an application
that is running in the background, because the function displays several
dialog boxes on the user’s screen.

ASSEMBLY-LANGUAGE INFORMATION
The trap macro and routine selector for the StartSecureSession function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPC</td>
<td>$000E</td>
</tr>
</tbody>
</table>
The registers on entry and exit for this routine are

**Registers on entry**

A0 Pointer to a `StartSecureParams` record
D0 Selector code

**Registrs on exit**

D0 Result code

### RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>noErr</code> No error</td>
</tr>
<tr>
<td>-128</td>
<td><code>userCanceledErr</code> User decided not to conduct a session</td>
</tr>
<tr>
<td>-900</td>
<td><code>notInitErr</code> PPC Toolbox has not been initialized yet</td>
</tr>
<tr>
<td>-902</td>
<td><code>nameTypeError</code> <code>locationKindSelector</code> is not <code>ppcNBPLocation</code> or <code>ppcNoLocation</code></td>
</tr>
<tr>
<td>-903</td>
<td><code>noPortErr</code> Bad port reference number</td>
</tr>
<tr>
<td>-904</td>
<td><code>noGlobalsErr</code> System unable to allocate memory, critical error</td>
</tr>
<tr>
<td>-905</td>
<td><code>localOnlyErr</code> Network activity is currently disabled</td>
</tr>
<tr>
<td>-906</td>
<td><code>destPortErr</code> Port does not exist at destination</td>
</tr>
<tr>
<td>-907</td>
<td><code>sessTableErr</code> PPC Toolbox is unable to create a session</td>
</tr>
<tr>
<td>-915</td>
<td><code>noResponseErr</code> Unable to contact application</td>
</tr>
<tr>
<td>-916</td>
<td><code>portClosedErr</code> The port was closed</td>
</tr>
<tr>
<td>-919</td>
<td><code>badPortNameErr</code> PPC port record is invalid</td>
</tr>
<tr>
<td>-924</td>
<td><code>noUserRefErr</code> Unable to create a new user reference number</td>
</tr>
<tr>
<td>-925</td>
<td><code>networkErr</code> An error has occurred in the network</td>
</tr>
<tr>
<td>-926</td>
<td><code>noInformErr</code> PPCStart failed because target application did not have an inform pending</td>
</tr>
<tr>
<td>-930</td>
<td><code>badServiceMethodErr</code> Service method is other than <code>ppcServiceRealTime</code></td>
</tr>
<tr>
<td>-932</td>
<td><code>guestNotAllowedErr</code> Destination port requires authentication</td>
</tr>
</tbody>
</table>

### SEE ALSO

For an example of the use of the `StartSecureSession` function, see “Initiating a PPC Session” beginning on page 11-29.

### PPCEnd

Use the `PPCEnd` function to end a session. This function completes all outstanding asynchronous calls associated with the session reference number.

**FUNCTION PPCEnd (pb: PPCEndPBPtr; async: Boolean): OSErr;**

*pb* A pointer to a `PPCEnd` parameter block.

*async* A value that specifies whether the function is to be executed asynchronously (TRUE) or synchronously (FALSE).
If your application calls the PPCEnd function asynchronously, you must specify in the ioCompletion field either the address of a completion routine or NIL. If you set ioCompletion to NIL, you should poll the ioResult field of the PPC parameter block (from your application’s main event loop) to determine whether the PPC Toolbox has completed the requested operation. A value in the ioResult field other than 1 indicates that the call is complete. Note that it is unsafe to poll the ioResult field at interrupt time since the PPC Toolbox may be in the process of completing a call. See “PPC Toolbox Calling Conventions” beginning on page 11-14 for detailed information.

You provide a session identifier in the sessRefNum field to identify the session that you are terminating. The PPCStart, StartSecureSession, or PPCInform function returns the session reference number.

### ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the PPCEnd function are:

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPC</td>
<td>$0008</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are:

**Registers on entry**

- **A0** Point to a parameter block
- **D0** Selector code

**Registers on exit**

- **D0** Result code

### RESULT CODES

- **noErr** 0 No error
- **notInitErr** -900 PPC Toolbox has not been initialized yet
- **noGlobalsErr** -904 System unable to allocate memory, critical error
- **noSessionErr** -908 Invalid session reference number

### SEE ALSO

For an example of the use of the PPCEnd function, see Listing 11-18 on page 11-43.
Receiving, Accepting, and Rejecting a Session

You use the PPCInform function to receive session requests. After the PPCInform function completes (with the autoAccept field set to FALSE), you must accept or reject the session request using the PPCAccept or PPCReject function.

PPCInform

As long as a port has been opened, you can call the PPCInform function at any time. You can have any number of outstanding PPCInform functions.

FUNCTION PPCInform (pb: PPCInformPBPtr; async: Boolean): OSErr;

- pb: A pointer to a PPCInform parameter block.
- async: A value that specifies whether the function is to be executed asynchronously (TRUE) or synchronously (FALSE). You should execute the PPCInform function asynchronously.

Parameter block:

- ioCompletion: Address of a completion routine.
- ioResult: Result code.
- portRefNum: Port reference number of this session.
- sessRefNum: Session reference number of this session.
- serviceType: Service type of this session.
- autoAccept: If TRUE, session is accepted automatically.
- portName: Pointer to PPCPortRec, may be NIL.
- locationName: Pointer to LocationNameRec, may be NIL.
- userName: Pointer to Str32, may be NIL.
- userData: Application-specific data.
- requestType: Network or local request.

DESCRIPTION

If your application calls the PPCInform function asynchronously, you must specify in the ioCompletion field either the address of a completion routine or NIL. If you set ioCompletion to NIL, you should poll the ioResult field of the PPC parameter block (from your application’s main event loop) to determine whether the PPC Toolbox has completed the requested operation. A value in the ioResult field other than 1 indicates that the call is complete. Note that it is unsafe to poll the ioResult field at interrupt time since the PPC Toolbox may be in the process of completing a call. See “PPC Toolbox Calling Conventions” beginning on page 11-14 for detailed information.
If you call the PPCInform function asynchronously, you must not change any of the fields in the parameter block until the call completes. The port name, location name, user name, and buffer pointed to by the record of type PPCInformPBRec are owned by the PPC Toolbox until the call completes. These objects must not be deallocated or moved in memory while the call is in progress.

You provide the PPC port identifier in the portRefNum field. A PPCOpen function returns the port identifier. The sessRefNum field returns a session identifier.

The serviceType field indicates the service type. For system software version 7.0, this field always returns the ppcServiceRealTime constant.

If you set the autoAccept field to TRUE, session requests are automatically accepted as they are received. When the PPCInform function completes execution with a noErr result code and you set the autoAccept field to FALSE, you need to accept or reject the session.

▲ WARNING
If the PPCInform function (with the autoAccept parameter set to FALSE) returns a noErr result code, you must call either the PPCAccept function or the PPCReject function. The computer trying to initiate a session using the StartSecureSession function or the PPCStart function waits (hangs) until the session attempt is either accepted or rejected, or until an error occurs. ▲

The portName field must contain NIL or a pointer to a PPC port record. If the portName field contains NIL, then the name of the PPC port that initiated the session is not returned. If the portName field points to a PPC port record, then the PPC port record is filled with the name of the PPC port that initiated the session when the PPCInform function completes.

The locationName field must contain NIL or a pointer to a location name record. If the locationName field contains NIL, then the location of the PPC port that initiated the session is not returned. If the locationName field points to a location name record, then the location name record is filled with the location of the PPC port that initiated the session when the PPCInform function completes. If the locationKindSelector field of the location name record returned is ppcNoLocation, then the location is the local machine. If the locationKindSelector field of the location name record returned is ppcNBPLocation, then the location is a remote machine designated by the location name record’s nbpEntity field.

The userName field must contain NIL or a pointer to a 32-byte character string. If the userName field contains NIL, then the user name string is not returned. If the userName field points to a 32-byte character string, then the 32-byte character string is filled with the name of the user making the session request (if authenticated) when the PPCInform function completes.

When the PPCInform function completes, the userData field contains the user data provided by the application making the session request. This field is transparent to the PPC Toolbox. The application can send any data in this field.
When the PPCInform function completes, the requestType field contains either ppcRemoteOrigin or ppcLocalOrigin, depending on whether the session request is initiated by a computer across the network or by a port on the same computer.

**ASSEMBLY-LANGUAGE INFORMATION**

The trap macro and routine selector for the PPCInform function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPC</td>
<td>$0003</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are

**Registers on entry**

- A0  Pointer to a parameter block
- D0  Selector code

**Registers on exit**

- D0  Result code

**RESULT CODES**

- noErr  0  No error
- notInitErr  -900  PPC Toolbox has not been initialized yet
- noPortErr  -903  Unable to open port or bad port reference number
- noGlobalsErr  -904  System unable to allocate memory, critical error
- portClosedErr  -916  The port was closed

**SEE ALSO**

For an example of the use of the PPCInform function, see Listing 11-8 on page 11-36.

**PPCAccept**

Use the PPCAccept function to indicate that an application is willing to accept an incoming session request after a PPCInform function completes.

**FUNCTION**

```
FUNCTION PPCAccept (pb: PPCAcceptPBPtr; async: Boolean): OSErr;
```

- **pb**  A pointer to a PPCAccept parameter block.
- **async**  A value that specifies whether the function is to be executed asynchronously (TRUE) or synchronously (FALSE).
DESCRIPTION

If your application calls the PPCAccept function asynchronously, you must specify in the ioCompletion field either the address of a completion routine or NIL. If you set ioCompletion to NIL, you should poll the ioResult field of the PPC parameter block (from your application’s main event loop) to determine whether the PPC Toolbox has completed the requested operation. A value in the ioResult field other than 1 indicates that the call is complete. Note that it is unsafe to poll the ioResult field at interrupt time since the PPC Toolbox may be in the process of completing a call. See “PPC Toolbox Calling Conventions” beginning on page 11-14 for detailed information.

The sessRefNum field specifies a session identifier. Use the session reference number returned from the completed PPCInform parameter block to accept the session request.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the PPCAccept function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPC</td>
<td>$0004</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are

Registers on entry

- A0    Pointer to a parameter block
- D0    Selector code

Registers on exit

- D0    Result code

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>notInitErr</td>
<td>–900 PPC Toolbox has not been initialized yet</td>
</tr>
<tr>
<td>noGlobalsErr</td>
<td>–904 System unable to allocate memory, critical error</td>
</tr>
<tr>
<td>noSessionErr</td>
<td>–908 Invalid session reference number</td>
</tr>
<tr>
<td>badReqErr</td>
<td>–909 Bad parameter or invalid state for this operation</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of the PPCAccept function, see “Accepting or Rejecting Session Requests” beginning on page 11-37.
Use the PPCReject function to reject a session request after a PPCInform function completes.

FUNCTION PPCReject (pb: PPCRejectPBPtr; async: Boolean): OSErr;

pb A pointer to a PPCReject parameter block.
async A value that specifies whether the function is to be executed asynchronously (TRUE) or synchronously (FALSE).

Parameter block

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioCompletion</td>
<td>Address of a completion routine</td>
</tr>
<tr>
<td>ioResult</td>
<td>Result code</td>
</tr>
<tr>
<td>sessRefNum</td>
<td>Session reference number of session to reject</td>
</tr>
<tr>
<td>rejectInfo</td>
<td>Value to return if session is rejected</td>
</tr>
</tbody>
</table>

DESCRIPTION

If your application calls the PPCReject function asynchronously, you must specify in the ioCompletion field the address of a completion routine or NIL. If you set ioCompletion to NIL, you should poll the ioResult field of the PPC parameter block (from your application’s main event loop) to determine whether the PPC Toolbox has completed the requested operation. A value in the ioResult field other than 1 indicates that the call is complete. Note that it is unsafe to poll the ioResult field at interrupt time since the PPC Toolbox may be in the process of completing a call. See “PPC Toolbox Calling Conventions” beginning on page 11-14 for detailed information.

The sessRefNum field specifies a session to be rejected. This must be a valid session reference number returned from a previous PPCInform function. The rejectInfo field is an optional field. The application receiving a session request may specify any data in this field. The initiating application receives this information in the PPCStart parameter block.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the PPCReject function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPC</td>
<td>$0005</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are

<table>
<thead>
<tr>
<th>Registers on entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>Pointer to a parameter block</td>
</tr>
<tr>
<td>D0</td>
<td>Selector code</td>
</tr>
</tbody>
</table>
CHAPTER 11

Program-to-Program Communications Toolbox

PPC Toolbox Reference

11-71

RESULT CODES

For an example of the use of the PPCReject function, see page 11-39.

Reading and Writing Data

The PPCRead function reads incoming data from an application, and the PPCWrite function writes data to an application during a session.

PPCRead

Use the PPCRead function to read message blocks during a session.

FUNCTION PPCRead (pb: PPCReadPBPtr; async: Boolean): OSErr;

pb A pointer to a PPCRead parameter block.
async A value that specifies whether the function is to be executed asynchronously (TRUE) or synchronously (FALSE). You should execute the PPCRead function asynchronously.

Parameter block

| →  | ioCompletion    | PPCCmpProcPtr | Address of a completion routine |
| ←  | ioResult        | OSErr         | Result code                     |
| →  | sessRefNum      | PPCSessRefNum | Session reference number        |
| →  | bufferLength    | Size          | Length of data buffer           |
| ←  | actualLength    | Size          | Actual length of data read      |
| →  | bufferPtr       | Ptr           | Pointer to data buffer          |
| ←  | more            | Boolean       | TRUE if more data in this block to be read |
| ←  | userData        | LongInt       | Application-specific data       |
| ←  | blockCreator    | OSType        | Creator of block read           |
| ←  | blockType       | OSType        | Type of block read              |
DESCRIPTION

If your application calls the **PPCRead** function asynchronously, you must specify in the *ioCompletion* field either the address of a completion routine or NIL. If you set *ioCompletion* to NIL, you should poll the *ioResult* field of the PPC parameter block (from your application’s main event loop) to determine whether the PPC Toolbox has completed the requested operation. A value in the *ioResult* field other than 1 indicates that the call is complete. Note that it is unsafe to poll the *ioResult* field at interrupt time since the PPC Toolbox may be in the process of completing a call. See “PPC Toolbox Calling Conventions” beginning on page 11-14 for detailed information.

If you call the **PPCRead** function asynchronously, you must not change any of the fields in the parameter block until the call completes. The buffer pointed to by the record of data type **PPCReadPBRec** is owned by the PPC Toolbox until the call completes. These objects must not be deallocated or moved in memory while the call is in progress.

The *sessRefNum* field specifies a session to read data from. This must be a valid session reference number returned from a previous **PPCStart**, **StartSecureSession**, or **PPCInform** function. The *bufferLength* and *bufferPtr* fields specify the length and location of a buffer the message block will be read into. Your application must allocate the storage for the buffer. The *actualLength* field returns the actual size of the data read into your data buffer.

The *more* field returns **TRUE** if the provided buffer cannot hold the remainder of the message block. Your application may read a message block in several pieces. It is not necessary to have a buffer large enough to read in the entire message block, so a message block can span multiple calls to the **PPCRead** function.

Upon completion of the **PPCRead** function, the *userData*, *blockCreator*, and *blockType* fields contain information regarding the contents of the message block. You specify these fields using the **PPCWrite** function.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the **PPCRead** function are:

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPC</td>
<td>$0007</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are:

**Registers on entry**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>Pointer to a parameter block</td>
</tr>
<tr>
<td>D0</td>
<td>Selector code</td>
</tr>
</tbody>
</table>

**Registers on exit**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>Result code</td>
</tr>
</tbody>
</table>
RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>No error</td>
</tr>
<tr>
<td>notInitErr</td>
<td>PPC Toolbox has not been initialized yet</td>
</tr>
<tr>
<td>noGlobalsErr</td>
<td>System unable to allocate memory, critical error</td>
</tr>
<tr>
<td>noSessionErr</td>
<td>Invalid session reference number</td>
</tr>
<tr>
<td>badReqErr</td>
<td>Bad parameter or invalid state for this operation</td>
</tr>
<tr>
<td>sessClosedErr</td>
<td>The session has closed</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of the PPCRead function in conjunction with the PPCWrite function, see “Exchanging Data During a PPC Session” beginning on page 11-39.

PPCWrite

Use the PPCWrite function to write message blocks during a session.

FUNCTION PPCWrite (pb: PPCWritePBPtr; async: Boolean): OSErr;

pb A pointer to a PPCWrite parameter block.
async A value that specifies whether the function is to be executed asynchronously (TRUE) or synchronously (FALSE). You should execute the PPCWrite function asynchronously.

Parameter block

- ioCompletion PPCCompProcPtr Address of a completion routine
- ioResult OSErr Result code
- sessRefNum PPCSessRefNum Session reference number
- bufferLength Size Length of data buffer
- actualLength Size Actual length of data written
- bufferPtr Ptr Pointer to data buffer
- more Boolean TRUE if more data in this block to be written
- userData LongInt Application-specific data
- blockCreator OSType Creator of block written
- blockType OSType Type of block written

DESCRIPTION

If your application calls the PPCWrite function asynchronously, you must specify in the ioCompletion field either the address of a completion routine or NIL. If you set ioCompletion to NIL, you should poll the ioResult field of the PPC parameter block (from your application’s main event loop) to determine whether the PPC Toolbox has
completed the requested operation. A value in the ioResult field other than 1 indicates that the call is complete. Note that it is unsafe to poll the ioResult field at interrupt time since the PPC Toolbox may be in the process of completing a call. See “PPC Toolbox Calling Conventions” beginning on page 11-14.

If you call the PPCWrite function asynchronously, you must not change any of the fields in the parameter block until the call completes. The buffer pointed to by the record of data type PPCWritePBRec is owned by the PPC Toolbox until the call completes. These objects must not be deallocated or moved in memory while the call is in progress.

The sessRefNum field specifies a session identifier. This must be a valid session reference number returned from a previous PPCStart, StartSecureSession, or PPCInform function.

The bufferLength and bufferPtr fields specify the length and location of a buffer the message block is sent to. If the PPCWrite function returns a noErr result code, the actualLength field returns the actual size of the message block that was written.

Set the more field to TRUE to indicate that you will be using the PPCWrite function again to append data to this message block. Set the more field to FALSE to indicate that this is the end of the data in this message block.

The initiating port can specify any information in the userData field. The PPCRead function reports this data to the responding port upon its completion.

Set the userData, blockCreator, and blockType fields for each message block that you create. These fields can give the receiving application information about how to process the contents of the message block. They are ignored when you append information to a message block.

**ASSEMBLY-LANGUAGE INFORMATION**

The trap macro and routine selector for the PPCWrite function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPC</td>
<td>$0006</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are

**Registers on entry**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>Pointer to a parameter block</td>
</tr>
<tr>
<td>D0</td>
<td>Selector code</td>
</tr>
</tbody>
</table>

**Registers on exit**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>Result code</td>
</tr>
</tbody>
</table>
RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0</td>
</tr>
<tr>
<td>notInitErr</td>
<td>-900 PPC Toolbox has not been initialized yet</td>
</tr>
<tr>
<td>noGlobalsErr</td>
<td>-904 System unable to allocate memory, critical error</td>
</tr>
<tr>
<td>noSessionErr</td>
<td>-908 Invalid session reference number</td>
</tr>
<tr>
<td>badReqErr</td>
<td>-909 Bad parameter or invalid state for this operation</td>
</tr>
<tr>
<td>sessClosedErr</td>
<td>-917 The session has closed</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of the PPCWrite function in conjunction with the PPCRead function, see “Exchanging Data During a PPC Session” beginning on page 11-39.

Locating a Default User and Invalidating a User

You use the GetDefaultUser function to obtain a user reference number and the name of the default user. To invalidate a particular user name and corresponding password, use the DeleteUserIdentity function.

GetDefaultUser

The GetDefaultUser function returns the user reference number and the name of the default user.

FUNCTION GetDefaultUser (VAR userRef: LongInt;
                         VAR userName: Str32): OSErr;

userRef        If the GetDefaultUser function completes with no errors, then the userRef parameter returns the user reference number that represents the user name and password of the default user.

userName       The name of the default user.

DESCRIPTION

The default user is specified in the Sharing Setup control panel. This function is useful if your application uses the PPCStart function to initiate a session with an application that does not support guest access.

If the GetDefaultUser function completes with no errors, then the userRef parameter returns the user reference number that represents the user name and password of the default user. The userName parameter must contain NIL or a 32-byte character string. If the userName parameter contains NIL, then the user name string is...
not returned. If the userName parameter is a 32-byte character string, the 32-byte character string contains the user name that is specified in the Sharing Setup control panel when the GetDefaultUser function completes (with no errors).

▲ WARNING
If you are using Pascal, you cannot pass NIL for the userName parameter. For example, you cannot pass StringPtr(NIL) because Pascal performs range checking of string bounds. ▲

ASSEMBLY-LANGUAGE INFORMATION
The trap macro and routine selector for the GetDefaultUser function are

Trap macro Selector
_PPCC $000D

The registers on entry and exit for this routine are

Registers on entry
A0 Pointer to a GetDefaultUserParams record
D0 Selector code

Registers on exit
D0 Result code

RESULT CODES

noErr 0 No error
noDefaultUserErr -922 User has not specified owner name in Sharing Setup control panel
notLoggedInErr -923 Default user reference number does not yet exist

SEE ALSO
For an example of the use of the GetDefaultUser function, see Listing 11-20 on page 11-45.

DeleteUserIdentity

To invalidate a particular user name and corresponding password, use the DeleteUserIdentity function.

FUNCTION DeleteUserIdentity (userRef: LongInt): OSErr;

userRef The reference number representing the user and password to be deleted.
DESCRIPTION

The DeleteUserIdentity function deletes the user name and password corresponding to the user reference number.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the DeleteUserIdentity function are

<table>
<thead>
<tr>
<th>Trap macro</th>
<th>Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PPC</td>
<td>$000C</td>
</tr>
</tbody>
</table>

The registers on entry and exit for this routine are

**Registers on entry**

| A0 | Pointer to a DeleteUserParams record |
| D0 | Selector code                        |

**Registers on exit**

| D0 | Result code |

RESULT CODES

- **noErr** 0 No error
- **noUserRecErr** -928 Invalid user reference number

SEE ALSO

For an example of the use of the DeleteUserIdentity function, see “Invalidating Users” on page 11-44.

Application-Defined Routines

This section describes the routine syntax for completion routines and port filter functions.

Completion Routines for PPC Toolbox Routines

Your application can provide a pointer to a completion routine in the ioCompletion field of a PPC parameter block. You can provide completion routines only for PPC Toolbox routines that you execute asynchronously.
**MyCompletionRoutine**

You can provide a completion routine for a PPC Toolbox routine that you execute asynchronously.

```pascal
PROCEDURE MyCompletionRoutine (pb: PPCParamBlockPtr);

pb                        A pointer to the PPC parameter block passed to the PPC Toolbox function.
```

**DESCRIPTION**

If you specify a completion routine in the `ioCompletion` field of a PPC parameter block, it is called at interrupt time when the PPC Toolbox routine completes execution. The PPC Toolbox passes to your completion routine a pointer to the same PPC parameter block that your application passed to the PPC Toolbox routine.

▲ **WARNING**

Completion routines execute at the interrupt level and must preserve all registers other than A0, A1, and D0–D2. (Note that MPW C and MPW Pascal do this automatically.) Your completion routine must not make any calls to the Memory Manager, directly or indirectly, and it can’t depend on the validity of handles to unlocked blocks. The PPC Toolbox preserves the application global register A5. ▲

**SEE ALSO**

For examples of completion routines, see Listing 11-9 on page 11-37, Listing 11-11 on page 11-38, and Listing 11-13 on page 11-39.

**Port Filter Functions**

This section describes the port filter function that can be used by the `PPCBrowser` function.

**MyPortFilter**

You can provide a pointer to a port filter function in the `portFilter` parameter of the `PPCBrowser` function. You can use a port filter function to refine the list of PPC ports that the `PPCBrowser` function displays in the program linking dialog box.

```pascal
FUNCTION MyPortFilter (locationName: LocationNamePtr;
thePortInfo: PortInfoPtr): Boolean;
```

11-78 PPC Toolbox Reference
CHAPTER 11

Program-to-Program Communications Toolbox

locationName
A pointer to a location name record. This record specifies the location of
the PPC port currently under consideration for display in the program
linking dialog box.

thePortInfo
A pointer to a port information record. This record specifies the port
information for the PPC port currently under consideration for display in
the program linking dialog box.

DESCRIPTION
The PPCBrowser function calls your port filter function once for each port before it adds
that port to the dialog list. Your port filter function should return TRUE for each port that
should be displayed in the program linking dialog box, and FALSE for each port
that shouldn’t be displayed.

SEE ALSO
For an example of a port filter function, see Listing 11-3 on page 11-24. For a description
of the location name record, see page 11-49. For a description of the port information
record, see page 11-50.
Summary of the PPC Toolbox

Pascal Summary

Constants

CONST

{gestalt selectors}
gestaltPPCToolboxAttr = 'ppc '; {PPC Toolbox attributes}
gestaltPPCToolboxPresent = $0000; {PPC Toolbox is present}
gestaltPPCSupportsRealTime = $1000; {real time only in system } { software version 7.0}
gestaltPPCSupportsOutGoing = $0002; {support of outgoing } { sessions across a network}
gestaltPPCSupportsIncoming = $0001; {user enabled program } { linking in Sharing Setup } { control panel}

{service type}
ppcServiceRealTime = 1; {real time only in System 7}

{look-up type}
ppcNoLocation = 0; {there is no PPCLocName}
ppcNBPLocation = 1; {use AppleTalk NBP}
ppcNBPTypeLocation = 2; {use just the NBP type, fill } { in the rest with default}

{port type}
ppcByCreatorAndType = 1; {port type is specified as } { standard creator and type}
ppcByString = 2; {port type is in Pascal } { string format}

{session request type returned in the PPCInform function}
ppcLocalOrigin = 1; {session initiated on } { local computer}
ppcRemoteOrigin = 2; {session initiated on } { remote computer}
Data Types

TYPE

PPCServicesType = SignedByte; {service type}
PPCLocationKind = Integer; {look-up type}
PPCPortKinds = Integer; {port type}
PPCSessionOrigin = SignedByte; {local or remote}
PPCPortRefNum = Integer; {port reference number}
PPCSessRefNum = LongInt; {session reference number}

LocationNamePtr = `LocationNameRec;
LocationNameRec = RECORD
  locationKindSelector: PPCLocationKind; {which variant}
  CASE PPCLocationKind OF
    ppcNoLocation: {storage not used by this value}
    ppcNBPLocation: {NBP name entity}
      (nbpEntity: EntityName);
    ppcNBPTypeLocation: {just the NBP type string}
      (nbpType: Str32); {for the PPCOpen function}
END;
PortInfoPtr = `PortInfoRec;
PortInfoRec = RECORD
  filler1: SignedByte; {space holder}
  authRequired: Boolean; {authentication required}
  name: PPCPortRec; {port name}
END;

PPCPortPtr = `PPCPortRec;
PPCPortRec = RECORD
  nameScript: ScriptCode; {script identifier}
  name: Str32; {port name shown in program}
    {linking dialog box}
  portKindSelector: PPCPortKinds; {general category of application}
    CASE PPCPortKinds OF
      ppcByString: (portTypeStr: Str32); {32 characters}
      ppcByCreatorAndType: {4-character creator and type}
        (portCreator: OSType; portType: OSType);
    END;
END;
CHAPTER 11

Program-to-Program Communications Toolbox

PPCParamBlockPtr = ^PPCParamBlockRec;
PPCParamBlockRec =

RECORD
CASE Integer OF
  0: (openParam: PPCOpenPBRec); {PPCOpen params}
  1: (informParam: PPCInformPBRec); {PPCInform params}
  2: (startParam: PPCStartPBRec); {PPCStart params}
  3: (acceptParam: PPCAcceptPBRec); {PPCAccept params}
  4: (rejectParam: PPCRejectPBRec); {PPCReject params}
  5: (writeParam: PPCWritePBRec); {PPCWrite params}
  6: (readParam: PPCReadPBRec); {PPCRead params}
  7: (endParam: PPCEndPBRec); {PPCEnd params}
  8: (closeParam: PPCClosePBRec); {PPCClose params}
  9: (listPortsParam: IPCListPortsPBRec); {IPCListPorts params}
END;

PortInfoArrayPtr = ^PortInfoArray;
PortInfoArray = ARRAY[0..0] OF PortInfoRec;

PPCOpenPBPtr = ^PPCOpenPBRec;
PPCOpenPBRec =

RECORD
  qLink: Ptr; {private}
  csCode: Integer; {private}
  intUse: Integer; {private}
  intUsePtr: Ptr; {private}
  ioCompletion: PPCCompProcPtr; {address of a
                      { completion routine}
                      completion of operation}
  ioResult: OSErr; {private}
  reserved: ARRAY[1..5] OF LongInt;
                      {private}
  portRefNum: PPCPortRefNum; {PPC port identifier}
  filler1: LongInt; {space holder}
  serviceType: PPCServiceType; {real time only}
  resFlag: SignedByte; {reserved field}
  portName: PPCPortPtr; {name of port to be opened}
  locationName: LocationNamePtr; {location of port to be opened}
                       { opened}
  networkVisible: Boolean; {port is visible for}
                     { browsing}
  nbpRegistered: Boolean; {location name registered}
                       { on network}
END;
CHAPTER 11

Program-to-Program Communications Toolbox

**Summary of the PPC Toolbox**

```plaintext
PPCInformPBPtr = ^PPCInformPBRec;
PPCInformPBRec = RECORd
  qLink:      Ptr;          {private}
  csCode:     Integer;      {private}
  intUse:     Integer;      {private}
  intUsePtr:  Ptr;          {private}
  ioCompletion: PPCCompProcPtr;  {address of a completion }    
                             { routine}
  ioResult:   OSErr;        {completion of operation}
  reserved:   ARRAY[1..5] OF LongInt;
               {private}
  portRefNum: PPCPortRefNum; {identifier for requested }  
                  { port}
  sessRefNum: PPCSessRefNum; {session identifier}
  serviceType: PPCServiceType; {real time only}
  autoAccept: Boolean;      {automatic session }  
                          { acceptance}
  portName:   PPCPortPtr;   {name of port that }  
                  { initiated a session}
  locationName: LocationNamePtr; {location of port that }  
                               { initiated a session}
  userName:   StringPtr;    {name of user that }   
                { initiated a session}
  userData:   LongInt;      {application-defined}
  requestType: PPCSessionOrigin; {local or remote}
END;

PPCStartPBPtr = ^PPCStartPBRec;
PPCStartPBRec = RECORd
  qLink:      Ptr;          {private}
  csCode:     Integer;      {private}
  intUse:     Integer;      {private}
  intUsePtr:  Ptr;          {private}
  ioCompletion: PPCCompProcPtr;  {address of a completion }  
                             { routine}
  ioResult:   OSErr;        {completion of operation}
  reserved:   ARRAY[1..5] OF LongInt;
               {private}
  portRefNum: PPCPortRefNum; {identifier for requested }  
                  { port}
  sessRefNum: PPCSessRefNum; {session identifier}
  serviceType: PPCServiceType; {real time only}
```

Program-to-Program Communications Toolbox
Program-to-Program Communications Toolbox

resFlag: SignedByte; {reserved field}
portName: PPCPortPtr; {name of port to be opened}
locationName: LocationNamePtr; {location of port to be }
(reopen)
rejectInfo: LongInt; {rejection of session}
userData: LongInt; {application-specific}
userRefNum: LongInt; {specifies an authenticated }
(user)
END;

PPCAcceptPBPtr = ^PPCAcceptPBRec;
PPCAcceptPBRec =
RECORD
qLink: Ptr; {private}
csCode: Integer; {private}
intUse: Integer; {private}
intUsePtr: Ptr; {private}
ioCompletion: PPCCompProcPtr; {address of a completion }
{ routine}
ioResult: OSErr; {completion of operation}
reserved: ARRAY[1..5] OF LongInt;
{private}
filler1: Integer; {space holder}
sessRefNum: PPCSessRefNum; {session identifier}
END;

PPCRejectPBPtr = ^PPCRejectPBRec;
PPCRejectPBRec =
RECORD
qLink: Ptr; {private}
csCode: Integer; {private}
intUse: Integer; {private}
intUsePtr: Ptr; {private}
ioCompletion: PPCCompProcPtr; {address of a completion }
{ routine}
ioResult: OSErr; {completion of operation}
reserved: ARRAY[1..5] OF LongInt;
{private}
filler1: Integer; {space holder}
sessRefNum: PPCSessRefNum; {session identifier}
filler2: Integer; {space holder}
filler3: LongInt; {space holder}
filler4: LongInt; {space holder}
CHAPTER 11

Program-to-Program Communications Toolbox

rejectInfo: LongInt; {rejection of session}
END;

PPCWritePBPtr = PPCWritePBRec;
PPCWritePBRec = RECORD
  qLink: Ptr; {private}
  csCode: Integer; {private}
  intUse: Integer; {private}
  intUsePtr: Ptr; {private}
  ioCompletion: PPCCompProcPtr; {address of a completion }
    { routine}
  ioResult: OSErr; {completion of operation}
  reserved: ARRAY[1..5] OF LongInt;
    {private}
  filler1: Integer; {space holder}
  sessRefNum: PPCSessRefNum; {session identifier}
  bufferLength: Size; {length of buffer to be }
    { written}
  actualLength: Size; {actual size of data written}
  bufferPtr: Ptr; {location of buffer to be }
    { written}
  more: Boolean; {additional data to be }
    { written}
  filler2: SignedByte; {space holder}
  userData: LongInt; {application-specific}
  blockCreator: OSType; {creator of block to be }
    { written}
  blockType: OSType; {type of block to be written}
END;

PPCReadPBPtr = PPCReadPBRec;
PPCReadPBRec = RECORD
  qLink: Ptr; {private}
  csCode: Integer; {private}
  intUse: Integer; {private}
  intUsePtr: Ptr; {private}
  ioCompletion: PPCCompProcPtr; {address of a completion }
    { routine}
  ioResult: OSErr; {completion of operation}
  reserved: ARRAY[1..5] OF LongInt;
    {private}

Summary of the PPC Toolbox
CHAPTER 11

Program-to-Program Communications Toolbox

filler1: Integer; {space holder}
sessRefNum: PPCSessRefNum; {session identifier}
bufferLength: Size; {length of buffer to be read}
actualLength: Size; {actual size of the data}
{ read}
bufferPtr: Ptr; {location of buffer to be read}
more: Boolean; {additional data to be read}
filler2: SignedByte; {space holder}
userData: LongInt; {application-specific}
blockCreator: OSType; {creator of block to be read}
blockType: OSType; {type of block to be read}

END;

PPCEndPBPtr = 'PPCEndPBRec;
PPCEndPBRec = RECORD
  qLink: Ptr; {private}
csCode: Integer; {private}
intUse: Integer; {private}
intUsePtr: Ptr; {private}
ioCompletion: PPCCompProcPtr; {address of a completion routine}
ioResult: OSErr; {completion of operation}
reserved: ARRAY[1..5] OF LongInt; {private}
filler1: Integer; {space holder}
sessRefNum: PPCSessRefNum; {identifier of session to be terminated}
END;

PPCClosePBPtr = 'PPCClosePBRec;
PPCClosePBRec = RECORD
  qLink: Ptr; {private}
csCode: Integer; {private}
intUse: Integer; {private}
intUsePtr: Ptr; {private}
ioCompletion: PPCCompProcPtr; {address of a completion routine}
ioResult: OSErr; {completion of operation}
reserved: ARRAY[1..5] OF LongInt; {private}

Chapter 11

Program-to-Program Communications Toolbox

portRefNum: PPCPortRefNum; {identifier of port to be closed}

END;

IPCLListPortsPBPtr = ^IPCLListPortsPBRec;
IPCLListPortsPBRec = RECORD
  qLink: Ptr; {private}
csCode: Integer; {private}
intUse: Integer; {private}
intUsePtr: Ptr; {private}
ioCompletion: PPCCompProcPtr; {address of a completion routine}
ioResult: OSErr; {completion of operation}
reserved: ARRAY[1..5] OF LongInt; {private}
filler1: Integer; {space holder}
startIndex: Integer; {index to the port entry list}
requestCount: Integer; {number of entries to be returned}
actualCount: Integer; {actual number of port names}
portName: PPCPortPtr; {list of port names}
locationName: LocationNamePtr; {location of port names}
bufferPtr: PortInfoArrayPtr; {pointer to a buffer}
END;

PPC Toolbox Routines

Initializing the PPC Toolbox
FUNCTION PPCInit: OSErr;

Using the Program Linking Dialog Box
FUNCTION PPCBrowser (prompt: Str255; applListLabel: Str255;
defaultSpecified: Boolean;
VAR theLocation: LocationNameRec;
VAR thePortInfo: PortInfoRec;
portFilter: PPCFilterProcPtr;
theLocNBPType: Str32): OSErr;

Obtaining a List of Ports
FUNCTION IPCLListPorts (pb: IPCLListPortsPBPtr; async: Boolean): OSErr;
CHAPTER 11

Program-to-Program Communications Toolbox

Opening and Closing a Port

FUNCTION PPCOpen (pb: PPCOpenPBPtr; async: Boolean): OSErr;
FUNCTION PPCClose (pb: PPCClosePBPtr; async: Boolean): OSErr;

Starting and Ending a Session

FUNCTION PPCStart (pb: PPCStartPBPtr; async: Boolean): OSErr;
FUNCTION StartSecureSession (pb: PPCStartPBPtr; VAR userName: Str32;
useDefault: Boolean; allowGuest: Boolean;
VAR guestSelected: Boolean; prompt: Str255) : OSErr;
FUNCTION PPCEnd (pb: PPCEndPBPtr; async: Boolean): OSErr;

Receiving, Accepting, and Rejecting a Session

FUNCTION PPCInform (pb: PPCInformPBPtr; async: Boolean): OSErr;
FUNCTION PPCAccept (pb: PPCAcceptPBPtr; async: Boolean): OSErr;
FUNCTION PPCReject (pb: PPCRejectPBPtr; async: Boolean): OSErr;

Reading and Writing Data

FUNCTION PPCRead (pb: PPCReadPBPtr; async: Boolean): OSErr;
FUNCTION PPCWrite (pb: PPCWritePBPtr; async: Boolean): OSErr;

Locating a Default User and Invalidating a User

FUNCTION GetDefaultUser (VAR userRef: LongInt; VAR userName: Str32) : OSErr;
FUNCTION DeleteUserIdentity (userRef: LongInt): OSErr;

Application-Defined Routines

PROCEDURE MyCompletionRoutine (pb: PPCParamBlockPtr);
FUNCTION MyPortFilter (locationName: LocationNameRec;
thePortInfo: PortInfoRec): Boolean;
C Summary

Constants

CONST
enum {
    /\gestalt selectors*/
    #define gestaltPPCToolboxAttr 'ppc' /*PPC Toolbox attributes*/
    gestaltPPCToolboxPresent = $0000, /*PPC Toolbox is present*/
    gestaltPPCSupportsRealTime = $1000, /*real time only in system*/
        /* software version 7.0*/
    gestaltPPSupportsOutGoing = $0002, /*support of outgoing*/
        /* sessions across a network*/
    gestaltPPSupportsIncoming = $001 /*user enabled program*/
        /* linking in Sharing Setup*/
        /* control panel*/
};
enum {
    /*service type*/
    ppcServiceRealTime = 1 /*real time only in System 7*/
};
enum {
    /*look-up type*/
    ppcNoLocation = 0, /*there is no PPCLocName*/
    ppcNBPLocation = 1, /*use AppleTalk NBP*/
    ppcNBPTypeLocation = 2 /*use just the NBP type, fill*/
        /* in the rest with default*/
};
enum {
    /*port type*/
    ppcByCreatorAndType = 1, /*port type is specified as*/
        /* standard Mac creator and type*/
    ppcByString = 2 /*port type is in Pascal*/
        /* string format*/
};
enum {
    /*session request type returned in the PPCInform function*/
    ppcLocalOrigin = 1, /*session initiated on*/
        /* local computer*/
    ppcRemoteOrigin = 2 /*session initiated on*/
        /* remote computer*/
};
CHAPTER 11
Program-to-Program Communications Toolbox

Data Types

typedef unsigned char PPCServiceType; /*service type*/
typedef short PPCLocationKind; /*look-up type*/
typedef short PPCPortKinds; /*port type*/
typedef unsigned char PPCSessionOrigin; /*local or remote*/
typedef short PPCPortRefNum; /*port reference number*/
typedef long PPCSessRefNum; /*session reference number*/

struct PPCPortRec {
    ScriptCode nameScript; /*script identifier*/
    Str32 name; /*port name shown in program*/
    PPCPortKinds portKindSelector; /*general category of*/
        /* application*/
    union
        Str32 portTypeStr; /*32 characters*/
        struct
            OSType creator; /*4-character creator and*/
            OSType type; /*type*/
        } port;
    } u;
};
typedef struct PPCPortRec PPCPortRec;
typedef PPCPortRec *PPCPortPtr;

struct LocationNameRec {
    PPCLocationKind locationKindSelector; /*which variant*/
    union {
        EntityName nbpEntity; /*NBP name entity*/
        Str32 nbpType; /*just the NBP type string*/
    } u;
};
typedef struct LocationNameRec LocationNameRec;
typedef LocationNameRec *LocationNamePtr;

struct PortInfoRec {
    unsigned char filler1; /*space holder*/
    Boolean authRequired; /*authentication required*/
    PPCPortRec name; /*port name*/
};
typedef struct PortInfoRec PortInfoRec;
typedef PortInfoRec *PortInfoPtr;

typedef PortInfoRec *PortInfoArrayPtr;
typedef pascal Boolean (*PPCFilterProcPtr) (LocationNamePtr, PortInfoPtr);
/*procedures you need to write*/
/*ex: void MyCompletionRoutine(PPCParamBlkPtr pb)*/
/*ex: pascal Boolean MyPortFilter(LocationNamePtr, PortInfoPtr)*/
typedef ProcPtr PPCCompProcPtr;

#define PPCHeader
  Ptr qLink; /*private*/
  unsigned short csCode; /*private*/
  unsigned short intUse; /*private*/
  Ptr intUsePtr; /*private*/
  PPCCompProcPtr ioCompletion; /*address of a */
  /*completion routine*/
  OSErr ioResult; /*completion of operation*/
  unsigned long Reserved[5]; /*private*/

struct PPCOpenPBRec {
  PPCHeader
  PPCPortRefNum portRefNum; /*PPC port identifier*/
  long filler1; /*space holder*/
  PPCServiceType serviceType; /*real time only*/
  unsigned char resFlag; /*reserved field*/
  PPCPortPtr portName; /*name of port to be opened*/
  LocationNamePtr locationName; /*location of port to be */
  /*opened*/
  Boolean networkVisible; /*port is visible for */
  /*browsing*/
  Boolean nbpRegistered; /*location name registered */
  /*on network*/
};

typedef struct PPCOpenPBRec PPCOpenPBRec;
typedef PPCOpenPBRec *PPCOpenPBPtr;

struct PPCInformPBRec {
  PPCHeader
  PPCPortRefNum portRefNum; /*port identifier*/
  PPCSessRefNum sessRefNum; /*session identifier*/
  PPCServiceType serviceType; /*real time only*/
Program-to-Program Communications Toolbox

Boolean autoAccept; /*automatic session acceptance*/
PPCPortPtr portName; /*name of port that */
/* initiated a session*/
LocationNamePtr locationName; /*location of port that */
/* initiated a session*/
StringPtr userName; /*name of user that */
/* initiated a session*/
unsigned long userData; /*application-defined*/
PPCSessionOrigin requestType; /*local or remote*/

};

typedef struct PPCInformPBRec PPCInformPBPtr;

struct PPCStartPBRec {
   PPCHeader
   PPCPortRefNum portRefNum; /*identifier for requested */
   /* port*/
   PPCSessRefNum sessRefNum; /*session identifier*/
   PPCServiceType serviceType; /*real time only*/
   unsigned char resFlag; /*reserved field*/
   PPCPortPtr portName; /*name of port to be opened*/
   LocationNamePtr locationName; /*location of port to be opened*/
   unsigned long rejectInfo; /*rejection of session*/
   unsigned long userData; /*application-specific*/
   unsigned long userRefNum; /*specifies an authenticated user*/
};

typedef struct PPCStartPBRec PPCStartPBRec;
typedef PPCStartPBRec *PPCStartPBPtr;

struct PPCAcceptPBRec {
   PPCHeader
   short filler1; /*space holder*/
   PPCSessRefNum sessRefNum; /*session identifier*/
};

typedef struct PPCAcceptPBRec PPCAcceptPBRec;
typedef PPCAcceptPBRec *PPCAcceptPBPtr;

struct PPCRejectPBRec {
   PPCHeader
   short filler1; /*space holder*/
   PPCSessRefNum sessRefNum; /*session identifier*/
};

typedef struct PPCRejectPBRec PPCRejectPBRec;
typedef PPCRejectPBRec *PPCRejectPBPtr;

Summary of the PPC Toolbox
Chapter 11

Program-to-Program Communications Toolbox

Summary of the PPC Toolbox

typedef struct PPCRejectPBRec PPCRejectPBRec;
typedef PPCRejectPBRec *PPCRejectPBPtr;

struct PPCWritePBRec {
    PPCHeader
    short filler1; /*space holder*/
    PPCSessRefNum sessRefNum; /*session identifier*/
    Size bufferLength; /*length of buffer to be written*/
    Size actualLength; /*actual size of data written*/
    Ptr bufferPtr; /*location of buffer to be written*/
    Boolean more; /*additional data to be written*/
    unsigned char filler2; /*space holder*/
    unsigned long userData; /*application-specific*/
    OSType blockCreator; /*creator of block to be written*/
    OSType blockSize; /*type of block to be written*/
};

typedef struct PPCWritePBRec PPCWritePBRec;
typedef PPCWritePBRec *PPCWritePBPtr;

struct PPCReadPBRec {
    PPCHeader
    short filler1; /*space holder*/
    PPCSessRefNum sessRefNum; /*session identifier*/
    Size bufferLength; /*length of buffer to be read*/
    Size actualLength; /*actual size of the data read*/
    Ptr bufferPtr; /*location of buffer to be read*/
    Boolean more; /*additional data to be read*/
    unsigned char filler2; /*space holder*/
    unsigned long userData; /*application-specific*/
    OSType blockCreator; /*creator of block to be read*/
    OSType blockSize; /*type of block to be read*/
};

typedef struct PPCReadPBRec PPCReadPBRec;
typedef PPCReadPBRec *PPCReadPBPtr;

short filler2; /*space holder*/
long filler3; /*space holder*/
long filler4; /*space holder*/
unsigned long rejectInfo; /*rejection of session*/

};
CHAPTER 11

Program-to-Program Communications Toolbox

struct PPCEndPBRec {
    PPCHeader
    short filler1;  /* space holder */
    PPCSessRefNum sessRefNum;  /* identifier of session to */
    /* be terminated */
};

typedef struct PPCEndPBRec PPCEndPBRec;
typedef PPCEndPBRec *PPCEndPBPtr;

struct PPCClosePBRec {
    PPCHeader
    PPCPortRefNum portRefNum;  /* identifier of port to */
    /* be closed */
};

typedef struct PPCClosePBRec PPCClosePBRec;
typedef PPCClosePBRec *PPCClosePBPtr;

struct IPCListPortsPBRec {
    PPCHeader
    short filler1;  /* space holder */
    unsigned short startIndex;  /* index to the port entry list */
    unsigned short requestCount;  /* number of entries to */
    /* be returned */
    unsigned short actualCount;  /* actual number of port names */
    PPCPortPtr portName;  /* list of port names */
    LocationNamePtr locationName;  /* location of port names */
    PortInfoArrayPtr bufferPtr;  /* pointer to a buffer */
};

typedef struct IPCListPortsPBRec IPCListPortsPBRec;
typedef IPCListPortsPBRec *IPCListPortsPBPtr;

union PPCParamBlockRec {
    PPCOpenPBRec openParam;  /* PPCOpen params */
    PPCInformPBRec informParam;  /* PPCInform params */
    PPCStartPBRec startParam;  /* PPCStart params */
    PPCAcceptPBRec acceptParam;  /* PPCAccept params */
    PPCRejectPBRec rejectParam;  /* PPCReject params */
    PPCWritePBRec writeParam;  /* PPCWrite params */
    PPCReadPBRec readParam;  /* PPCRead params */
    PPCEndPBRec endParam;  /* PPCEnd params */
    PPCClosePBRec closeParam;  /* PPCClose params */
};

11-94 Summary of the PPC Toolbox
CHAPTER 11

Program-to-Program Communications Toolbox

IPCLListPortsPBRec listPortsParam; /*IPCLListPorts params*/

typedef union PPCParamBlockRec PPCParamBlockRec;
typedef PPCParamBlockRec *PPCParamBlockPtr;

PPC Toolbox Routines

Initializing the PPC Toolbox
pascal OSErr PPCInit (void);

Using the Program Linking Dialog Box
pascal OSErr PPCBrowser (ConstStr255Param prompt, ConstStr255Param applListLabel, Boolean defaultSpecified, LocationNameRec *theLocation, PortInfoRec *thePortInfo, PPCFilterProcPtr portFilter, ConstStr32Param theLocNBPTyp); Obtaining a List of Ports
pascal OSErr IPCListPorts (IPCListPortsPBPtr pb, Boolean async);

Opening and Closing a Port
pascal OSErr PPCOpen (PPCOpenPBPtr pb, Boolean async);
pascal OSErr PPCClose (PPCClosePBPtr pb, Boolean async);

Starting and Ending a Session
pascal OSErr PPCStart (PPCStartPBPtr pb, Boolean async);
pascal OSErr StartSecureSession (PPCStartPBPtr pb, Str32 userName, Boolean useDefault, Boolean allowGuest, Boolean *guestSelected, ConstStr255Param prompt);
pascal OSErr PPCEnd (PPCEndPBPtr pb, Boolean async);

Receiving, Accepting, and Rejecting a Session
pascal OSErr PPCInform (PPCInformPBPtr pb, Boolean async);
pascal OSErr PPCAccept (PPCAcceptPBPtr pb, Boolean async);
pascal OSErr PPCReject (PPCRejectPBPtr pb, Boolean async);
CHAPTER 11
Program-to-Program Communications Toolbox

Reading and Writing Data

pascal OSErr PPCRead (PPCReadPBPtr pb, Boolean async);
pascal OSErr PPCWrite (PPCWritePBPtr pb, Boolean async);

Locating a Default User and Invalidating a User

pascal OSErr GetDefaultUser (unsigned long *userRef, Str32 userName);
pascal OSErr DeleteUserIdentity (unsigned long userRef);

Application-Defined Routines

void MyCompletionRoutine (PPCParamBlockPtr pb);
pascal Boolean MyPortFilter (LocationNameRec locationName, PortInfoRec thePortInfo);

Assembly-Language Summary

Trap Macros

Trap Macros Requiring Routine Selectors

_Pack9

<table>
<thead>
<tr>
<th>Selector</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0D00</td>
<td>PPCBrowser</td>
</tr>
</tbody>
</table>

_PPc

<table>
<thead>
<tr>
<th>Selector</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>PPCInit</td>
</tr>
<tr>
<td>$0001</td>
<td>PPCOpen</td>
</tr>
<tr>
<td>$0002</td>
<td>PPCStart</td>
</tr>
<tr>
<td>$0003</td>
<td>PPCInform</td>
</tr>
<tr>
<td>$0004</td>
<td>PPCAccept</td>
</tr>
<tr>
<td>$0005</td>
<td>PPCReject</td>
</tr>
<tr>
<td>$0006</td>
<td>PPCWrite</td>
</tr>
<tr>
<td>$0007</td>
<td>PPCRead</td>
</tr>
<tr>
<td>$0008</td>
<td>PPCEnd</td>
</tr>
<tr>
<td>$0009</td>
<td>PPCClose</td>
</tr>
</tbody>
</table>

11-96 Summary of the PPC Toolbox
CHAPTER 11

Program-to-Program Communications Toolbox

Reading and Writing Data

pascal OSErr PPCRead (PPCReadPBPtr pb, Boolean async);
pascal OSErr PPCWrite (PPCWritePBPtr pb, Boolean async);

Locating a Default User and Invalidating a User

pascal OSErr GetDefaultUser (unsigned long *userRef, Str32 userName);
pascal OSErr DeleteUserIdentity (unsigned long userRef);

Application-Defined Routines

void MyCompletionRoutine (PPCParamBlockPtr pb);
pascal Boolean MyPortFilter (LocationNameRec locationName,
                           PortInfoRec thePortInfo);

Assembly-Language Summary

Trap Macros

Trap Macros Requiring Routine Selectors

_pack9

<table>
<thead>
<tr>
<th>Selector</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0D00</td>
<td>PPCBrowser</td>
</tr>
</tbody>
</table>

_packPPC

<table>
<thead>
<tr>
<th>Selector</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>PPCInit</td>
</tr>
<tr>
<td>$0001</td>
<td>PPCOpen</td>
</tr>
<tr>
<td>$0002</td>
<td>PPCStart</td>
</tr>
<tr>
<td>$0003</td>
<td>PPCInform</td>
</tr>
<tr>
<td>$0004</td>
<td>PPCAccept</td>
</tr>
<tr>
<td>$0005</td>
<td>PPCReject</td>
</tr>
<tr>
<td>$0006</td>
<td>PPCWrite</td>
</tr>
<tr>
<td>$0007</td>
<td>PPCRead</td>
</tr>
<tr>
<td>$0008</td>
<td>PPCEnd</td>
</tr>
<tr>
<td>$0009</td>
<td>PPCClose</td>
</tr>
</tbody>
</table>
### Result Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>noErr 0 No error</td>
</tr>
<tr>
<td>-50</td>
<td>paramErr Illegal parameter</td>
</tr>
<tr>
<td>-108</td>
<td>memFullErr Not enough memory to load PPCBrowser package</td>
</tr>
<tr>
<td>-128</td>
<td>userCanceledErr User decided not to conduct a session</td>
</tr>
<tr>
<td>-900</td>
<td>notInitErr PPC Toolbox has not been initialized yet</td>
</tr>
<tr>
<td>-902</td>
<td>nameTypeErr Invalid or inappropriate locationKindSelector in location name</td>
</tr>
<tr>
<td>-903</td>
<td>noPortErr Unable to open port or bad port reference number</td>
</tr>
<tr>
<td>-904</td>
<td>noGlobalsErr System unable to allocate memory, critical error</td>
</tr>
<tr>
<td>-905</td>
<td>localOnlyErr Network activity is currently disabled</td>
</tr>
<tr>
<td>-906</td>
<td>destPortErr Port does not exist at destination</td>
</tr>
<tr>
<td>-907</td>
<td>sessTableErr PPC Toolbox is unable to create a session</td>
</tr>
<tr>
<td>-908</td>
<td>noSessionErr Invalid session reference number</td>
</tr>
<tr>
<td>-909</td>
<td>badReqErr Bad parameter or invalid state for this operation</td>
</tr>
<tr>
<td>-910</td>
<td>portNameExistsErr Another port is already open with this name</td>
</tr>
<tr>
<td>-911</td>
<td>noUserNameErr User name unknown on destination machine</td>
</tr>
<tr>
<td>-912</td>
<td>userRejectErr Destination rejected the session request</td>
</tr>
<tr>
<td>-915</td>
<td>noResponseErr Unable to contact application</td>
</tr>
<tr>
<td>-916</td>
<td>portClosedErr The port was closed</td>
</tr>
<tr>
<td>-917</td>
<td>sessClosedErr The session has closed</td>
</tr>
<tr>
<td>-919</td>
<td>badPortNameErr PPC port record is invalid</td>
</tr>
<tr>
<td>-922</td>
<td>noDefaultUserErr User has not specified owner name in Sharing Setup control panel</td>
</tr>
<tr>
<td>-923</td>
<td>notLoggedInErr Default user reference number does not yet exist</td>
</tr>
<tr>
<td>-924</td>
<td>noUserRefErr Unable to create a new user reference number</td>
</tr>
<tr>
<td>-925</td>
<td>networkErr An error has occurred in the network</td>
</tr>
<tr>
<td>-926</td>
<td>noInformErr PPCStart failed because target application did not have an inform pending</td>
</tr>
<tr>
<td>-927</td>
<td>authFailErr User's password is wrong</td>
</tr>
<tr>
<td>-928</td>
<td>noUserRecErr Invalid user reference number</td>
</tr>
<tr>
<td>-930</td>
<td>badServiceMethodErr Service method is other than ppcServiceRealTime</td>
</tr>
<tr>
<td>-931</td>
<td>badLocNameErr Location name is invalid</td>
</tr>
<tr>
<td>-932</td>
<td>guestNotAllowedErr Destination port requires authentication</td>
</tr>
<tr>
<td>-1027</td>
<td>nbpDuplicate Location name represents a duplicate on this computer</td>
</tr>
</tbody>
</table>