AT&T 585-310-208
AT&T Comcode: 106835838
Issue 1
October 1992

Switch Integration Toolkit

Notice

While reasonable efforts were made to ensure that the information in this document was complete and accurate at the time of printing, AT&T can assume no responsibility for any errors. Changes and corrections to the information contained in this document may be incorporated into future reissues.

Your Responsibility for Your System's Security

You are responsible for the security of your system. AT&T does not warrant that this product is immune from or will prevent unauthorized use of common-carrier telecommunication services or facilities accessed through or connected to it. AT&T will not be responsible for any charges that result from such unauthorized use. Product administration to prevent unauthorized use is your responsibility and your system administrator should read all documents provided with this product to fully understand the features available that may reduce your risk of incurring charges.

Federal Communications Commission (FCC) Statement

This equipment generates, uses, and can radiate radio-frequency energy and, if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment.

Operation of this equipment in a residential area is likely to cause interference, in which case the user at his/her own expense will be required to take whatever measures may be required to correct the interference.

Trademarks

Voice Power is a trademark of AT&T, and AUDIX, CONVERSANT, and DEFINITY are registered trademarks of AT&T.

Acknowledgment

This document was prepared by the AT&T Technical Publications Department, Columbus, Ohio.

Contents

About This Document

This document describes how to create a switch integration package for use with a Voice Power application.

This document is designed so that you can quickly find information about how, when, and why to perform specific tasks.

INTENDED AUDIENCES

Developers for Voice Processing Co-marketers (VPCs) and Master Voice Processing Co-marketers (MVPCs) are the primary audience for this document.

PREREQUISITE SKILLS OR KNOWLEDGE

This document assumes familiarity with the following.

- a UNIX® editor, such as vi
- software development concepts and structures, including the C compiler
- shell programming
- tsm scripts development
- CONVERSANT® Intro R1.0
- knowledge of the switch for which the package is being developed

HOW THIS DOCUMENT IS ORGANIZED

- Chapter 1, *Switch Integration Basics,* introduces switch integration concepts.
- Chapter 2, *Installing the Toolkit,* explains how to install the Toolkit from diskette onto your development environment.
- Chapter 3, *Development Process,* outlines the steps of developing a switch integration package. Each step is then detailed in successive chapters.
- Chapter 4, *Development Structures,* covers several models for developing sections of code specific to switch integrations.
- Chapter 5, *Call Information Options,* details the switch integration options for exchanging data between the switch and the application.
- Chapter 6, *Message-Waiting Options,* covers possibilities for controlling message-waiting indicators.
- Chapter 7, *Transfer Options,* details blind, semi-intelligent, and intelligent transfers and how they apply to switch integration.
- Chapter 8, *Disconnect Options,* specifies how the switch notifies the application that a caller has hung up.
- Chapter 9, *Miscellaneous Options,* describes four additional switch integration options: leave-word-calling, message-waiting refresh, in-service notification, and day/night service change.
- Chapter 10, *Installing Your Switch Integration Package,* steps you through putting your switch integration package on diskette and installing it on the run-time environment.
- Appendix A, *Development Aids,* contains worksheets and a checklist to help you develop a switch integration package.

A list of abbreviations, a glossary and an index are also included in this document.

HOW TO USE THIS DOCUMENT

First, read Chapter 1, *Switch Integration Basics,* and Chapter 3, *Development Process,* as an introduction to switch integrations. Use Chapter 2, *Installing the Toolkit,* to install the Toolkit software on your development machine. Read Chapters 5 through 10, in order, to develop and install the package, referencing Chapter 4, *Development Structures* and Appendix A, *Development Aids* as necessary.

CONVENTIONS USED IN THIS DOCUMENT

The following typographic conventions are used in this document.

- Terminal keys that you press are shown in rounded boxes. For example, an instruction to press the enter, carriage return, or equivalent key is shown in this document as the following.

  Press (ENTER)

- Phone pad keys that you press are shown in square boxes. For example, an instruction to press zero is shown in this document as the following.

  Press [0]

- The word *enter* means to type a value and press (ENTER). For example, an instruction to type **y** and press (ENTER) is shown in this document as the following.

  Enter **y** to continue.

- Two or three keys that you press at the same time (that is, you hold down the first key while pressing the second key and, if appropriate, the third key as well) are shown together in a rounded box and are separated by hyphens. For example, an instruction to press and hold (ALT) while typing the letter *d* is shown in this document as the following.

  Press (ALT-d).

- Information that is displayed on your terminal screen — including screen displays, field names, prompts, and error messages — is shown in typewriter-style constant-width type. Information that you enter from your keyboard is shown in constant-width bold type. Here is an example.

  At the `Login ID?` prompt, enter **`snowfox`**

- Variables that the system supplies or that you must supply are shown in italic type. For example, a message that is displayed on the screen with one of your specific filenames might be shown generically in this document as the following.

  `Your file` *filename* `has been saved.`

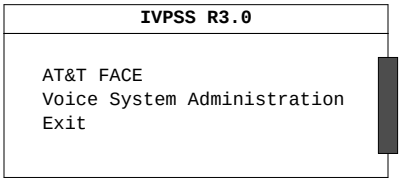
- The word *select* is used in this document to mean the following: move to the desired menu item using the arrow keys (highlight it) and press (ENTER).

- The Switch Integration Toolkit is referred to in this document as *the Toolkit*.

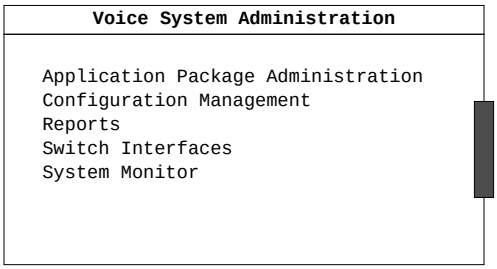
Series of Menu Selections

To perform a specific activity, you may have to pick through several menus to reach your desired destination. For example, to reach the VOICE EQUIPMENT window, you would have to do the following.

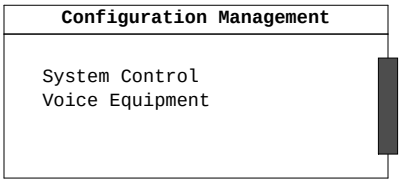
When you log on, the IVPSS R3.0 menu is displayed.



From the IVPSS R3.0 menu, select `Voice System Administration`.
This brings up the VOICE SYSTEM ADMINISTRATION menu.

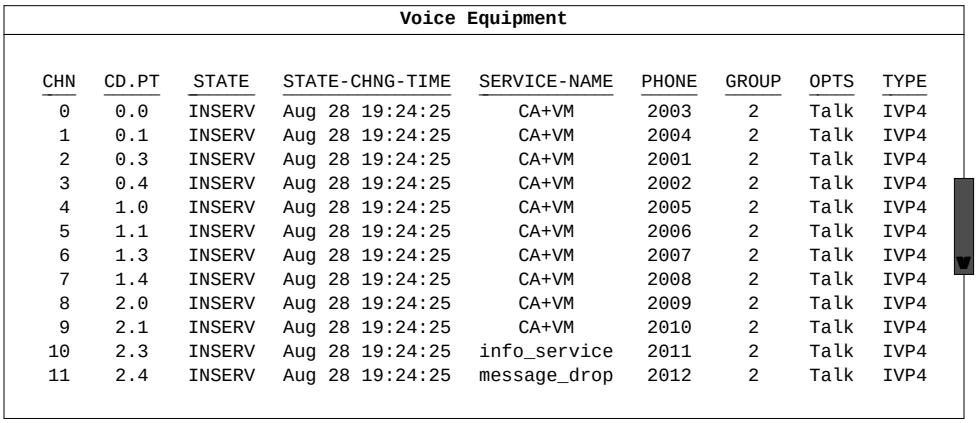


From the VOICE SYSTEM ADMINISTRATION menu, select `Configuration Management`.
This brings up the CONFIGURATION MANAGEMENT menu.



From the CONFIGURATION MANAGEMENT menu, select `Voice Equipment`.

This brings up the VOICE EQUIPMENT window.



Voice Equipment

CHN	CD.PT	STATE	STATE-CHNG-TIME	SERVICE-NAME	PHONE	GROUP	OPTS	TYPE
0	0.0	INSERV	Aug 28 19:24:25	CA+VM	2003	2	Talk	IVP4
1	0.1	INSERV	Aug 28 19:24:25	CA+VM	2004	2	Talk	IVP4
2	0.3	INSERV	Aug 28 19:24:25	CA+VM	2001	2	Talk	IVP4
3	0.4	INSERV	Aug 28 19:24:25	CA+VM	2002	2	Talk	IVP4
4	1.0	INSERV	Aug 28 19:24:25	CA+VM	2005	2	Talk	IVP4
5	1.1	INSERV	Aug 28 19:24:25	CA+VM	2006	2	Talk	IVP4
6	1.3	INSERV	Aug 28 19:24:25	CA+VM	2007	2	Talk	IVP4
7	1.4	INSERV	Aug 28 19:24:25	CA+VM	2008	2	Talk	IVP4
8	2.0	INSERV	Aug 28 19:24:25	CA+VM	2009	2	Talk	IVP4
9	2.1	INSERV	Aug 28 19:24:25	CA+VM	2010	2	Talk	IVP4
10	2.3	INSERV	Aug 28 19:24:25	info_service	2011	2	Talk	IVP4
11	2.4	INSERV	Aug 28 19:24:25	message_drop	2012	2	Talk	IVP4

This is a long and difficult way to show how to reach a menu. Therefore, a series of menu selections is shown in this document using the following convention.

Begin at the IVPSS R3.0 menu and pick the following sequence.

```
Voice System Administration
Configuration Management
Voice Equipment
```

In this example, the IVPSS R3.0 menu is the top level menu pick for this application. It is the first menu you see when logging on to the system. It is used consistently in all menu pick series, to serve as a point of reference regardless of where you are in the menu system.

Each subsequent menu pick is shown on its own line, so that you can enter the sequence at any point and still arrive at the desired menu. Each new line represents a different deeper-level menu from which you should make the selection shown.

TRADEMARKS AND SERVICE MARKS

The following trademarked products are mentioned in this document.

- CONVERSANT® is a registered trademark of AT&T.
- AUDIX® is a registered trademark of AT&T.
- Voice Power™ is a trademark of AT&T.
- DEFINITY® Communications System is a registered trademark of AT&T.
- UNIX® is a registered trademark of UNIX System Laboratories Inc.

RELATED RESOURCES

In addition to this document, you may also require the following resources.

Title	Select Code
CONVERSANT Intro Installation	585-312-110
CONVERSANT Intro Administration	585-312-114
CONVERSANT Intro Reference	585-312-113
CONVERSANT Intro User's Guide	585-312-112
CONVERSANT Intro Operations	585-312-111

In addition, you may also need documents from the Voice Power application sets. For ordering information, refer to the Business Communications Systems Publication Catalog (555-000-010).

HOW TO MAKE COMMENTS ABOUT THIS DOCUMENT

Reader comment cards are behind the title page of this document. While we have tried to make this document fit your needs, we are interested in your suggestions for improving it and urge you to complete and return a reader comment card.

If the reader comment cards have been removed from this document, please send your comments to the following address.

AT&T
Technical Publications Department
Room 22-2C11
11900 North Pecos Street
Denver, Colorado 80234

1. Switch Integration Basics

Switch integration refers to the sharing of information between a voice mail system and a switch in order to provide a seamless interface to callers and subscribers. There are various levels of switch integration. The following is a list of some features which can be incorporated as part of a switch integration package.

- detect and answer incoming calls
- recognize calling and called parties
- notify voice mail users that they have messages
- transfer callers to other numbers
- detect when a caller hangs up

Part of your job as a switch integration package developer is to determine which switch integration features are supported by a particular switch.

HOW CALL INFORMATION IS USED

Call information is the data passed from the switch to the voice mail system. Call information includes the following pieces of data.

- call type (direct, covered)
- calling party
- called party
- call origination (internal, external)
- reason for coverage (busy/ring no answer)

When the PBX transfers a call to AUDIX Voice Power, it also sends call information through a digital connection for example, to the Digital Communications Protocol (DCP) card or through a switch interface device (SID) to a communications port. This call information tells AUDIX Voice Power, for example, what type of call it is (internal, external, or covered) and who the caller is calling (extension or name).

Call information expedites the processing of a call. For example, with call information, subscribers can press # in response to the extension prompt when calling AUDIX Voice Power from their desks to get messages. This is because AUDIX Voice Power already knows the subscriber's extension via the call information it got from the PBX. By pressing #, the subscriber simply acknowledges that AUDIX Voice Power should use the PBX information it obtained (direct, internal, calling extension). After entering their passwords, subscribers are connected to their mailboxes.

Call information is used in a variety of ways in a single transaction. AUDIX Voice Power first uses PBX information to determine which service to provide. For example, if the call information says that this is an internal (one subscriber calling another), covered call (the called person was either on the phone or not there), then AUDIX Voice Power provides the call-answer service so that the calling party can leave a message. After determining the service, AUDIX Voice Power would reference the call information again to obtain the extension of the called person so that the caller is connected to the correct mailbox.

If your switch has no means of providing call information, the service provided to the caller is based on the name of the script assigned to the channel which answers the call. For example, if the call-answer script is assigned to all channels, all calls sent to the voice mail system, regardless of type, hear the call-answer greeting. This greeting prompts callers to reenter the extension of the person they were calling (because this information is not sent from the switch), and then the voice mail system connects them to the appropriate mailbox where they can leave a message.

Call information options for switch integration are covered in Chapter 5, *Call Information Options*.

HOW USERS ARE NOTIFIED OF MESSAGES WAITING

There are several ways a voice mail system can notify callers that they have voice mail. Notification capabilities are based on the switch and on the subscriber phone set type.

A *message-waiting lamp* on the phone set is the most common notification method. If the light is on, the user knows that there are new messages and can call the voice mail system to listen to them.

Dial tone modification is another notification method. Subscribers can check for new messages by lifting the handset and listening to the dial tone. A distinctive dial tone indicates the presence of messages, while the normal dial tone means no messages.

Finally, display phones for some switches have the ability to indicate when messages are waiting through a letter code shown in the display area.

Some voice mail systems (such as AUDIX Voice Power) support a notification method known as *outcalling*. This feature can be set up to call a subscriber at a specified number when new messages arrive. (Outcalling is not dependent on the switch. It can be used in addition to or instead of any of the above notification methods.)

AUDIX Voice Power Lodging allows the property management system to control the message-waiting lamps for hotel guests. In this case, your switch package does not need to support a message-waiting notification method.

Message-waiting options for switch integration are explained in Chapter 6, *Message-Waiting Options*.

HOW TRANSFERS WORK

A caller can be transferred to another number for a variety of reasons, including the following.

- caller pressed zero to transfer to an operator for assistance
- voice mail system failed to receive the expected call information
- caller selected to transfer to another extension or a number outside the switch

To interface with most switches, the voice mail system simply performs the transfer operation the way a person would transfer a call from his or her phone to another extension. A common transfer sequence is depress and release the switchhook (producing a flash), listen for a special dial tone (also known as *stutter*), dial the transfer to number, and hang up.

It is important to familiarize yourself with the transfer operations of a particular switch by using tip/ring (analog) telephones (as opposed to digital phones). A few switches support transfer requests over a voice mail interface. Some switches do not support transfers at all. Transfers are addressed in Chapter 7, *Transfer Options*.

HOW DISCONNECTS ARE HANDLED

A switch integration package must also dictate how the switch will notify the voice mail system when a caller hangs up (disconnects). Disconnect notification is important because it allows the voice mail system to free up the channel for another call. There are several types of disconnect signals that are recognized by Voice Power applications. Disconnects are covered in Chapter 8, *Disconnect Options*.

WHAT IS A SWITCH INTEGRATION DEVICE?

A Switch Integration Device (SID) is a link between a switch and a voice mail system which provides a standard interface for call information and message-waiting updates. Voice Power applications require the Simplified Message Desk Interface (SMDI) standard interface. The details of this interface are presented in Chapter 5, *Call Information Options*, and Chapter 6, *Message-Waiting Options*.

Because the SMDI protocol is an industry standard, you may wish to query switch or other equipment vendors concerning the availability of the device for a switch. If a SID is available for a switch, little effort is required for the call information and message-waiting updates portions of the switch integration package. However, there may still be transfer and disconnect issues.

2. Installing the Toolkit

This chapter describes how to install the Switch Integration Toolkit (the Toolkit) software into your development environment. You should install the Toolkit when you are ready to begin developing your switch integration package.

PREREQUISITES

Because the requirements differ, a distinction is made between the development environment and the run-time environment. The development environment is the hardware and software used to develop the switch integration package. The run-time environment is the hardware and software that the switch integration package is run on, for example, a test machine or a customer machine.

Development Environment

Before you can successfully install the Toolkit software, you must install the CONVERSANT Intro Application Development Software and the CONVERSANT Intro Advanced Application Toolkit Software. Refer to the documentation included with these packages for details on their installations.

Run-time Environment

The switch packages developed using the Toolkit are designed to run on the Integrated Voice Processing System Software Release (IVPSS) 3.0, along with at least one Voice Power application package, such as AUDIX Voice Power Release 3.0 or AUDIX Voice Power Lodging Release 3.0.

INSTALLATION PROCEDURE

Use the following procedure to install the Switch Integration Toolkit software.

1. Log in as root.

   The system displays the UNIX system prompt (#).

   NOTE

   You *must* be logged in as root to use the `installpkg` utility. If you are not logged in as root, log out and log back in as root.

2. Enter **`displaypkg`**

   The system displays a list of the software installed on the system.

3. Verify that the following two packages appear in the `displaypkg` listing.

   ```
   CONVERSANT Intro Application Development Software
   CONVERSANT Intro Advanced Application Toolkit Software
   ```

   NOTE

   If these packages are not listed, you must install them before continuing with this procedure. Otherwise, the Switch Integration Toolkit software installation will fail.

4. Enter **`installpkg`**

   The system displays the following message.

   ```
   Strike ENTER when ready or ESC to stop.
   ```

5. Insert floppy disk #1 for the Switch Integration Toolkit Software package into the floppy disk drive.

6. Press ENTER.

   Status messages are displayed as portions of the installation are completed.

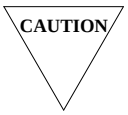
7. Continue loading disks as prompted by the following message.

   ```
   Reached end of medium on input.
   You may remove this floppy disk.
   To QUIT - strike <q> followed by <Enter>.
   To continue - insert floppy disk <number> and strike the
   <Enter> key.
   ```

8. When installation is complete, the system displays the following message.

```
The installation of the Switch Integration Toolkit Software
is now complete.
```

9. Remove the last floppy disk and store all the disks in a secure place.



Do not leave a floppy diskette in the drive.

The Switch Integration Toolkit software is now installed.

Be sure to read Chapter 1, *Switch Integration Basics,* and Chapter 3, *Development Process,* as an introduction to switch integrations. Then, read Chapters 5 through 10, in order, to develop and install the package, referencing Chapter 4, *Development Structures* and Appendix A, *Development Aids* as necessary.

REMOVING THE TOOLKIT

This procedure is provided in the event that you need to remove the Switch Integration Toolkit software from your development environment.

To remove the Toolkit software, do the following.

1. Log in to the system as root.

   The system displays the UNIX system prompt (#).

   NOTE

   You *must* be logged in as root to used the `removepkg` utility. If you are not logged in as root, log out and log back in as root.

2. Enter **`removepkg`**

   The system displays a list of installed packages numbered 1 to *n*.

3. Enter the number of the Switch Integration Toolkit Software package.

   The system displays the following message.

   ```
   Do you really want to remove the Switch Integration Toolkit
   Software Package?

   Strike ENTER when ready or ESC to stop.
   ```

4. Press ENTER

   To verify that the package is being removed, use Ctrl+S (stop screen scroll) and Ctrl+Q (start screen scroll) to view the following status messages.

   ```
   Removing system files for the Switch Integration Toolkit
   Software.

   The Switch Integration Toolkit Software has been successfully
   removed.

   The Switch Integration Toolkit Software is now removed.
   ```

3. Development Process

This chapter provides an overview of the switch integration package development process. This process includes the following.

- setting up the development environment
- selecting various options based on the capabilities of the switch
- coding switch-specific software, if necessary
- assembling the switch package on diskette
- installing the switch package on the run-time machine
- testing the switch package

SETTING UP THE DEVELOPMENT ENVIRONMENT

There are very few requirements on the directory structure for your switch integration package development environment. The following guidelines are meant to provide consistency and simplify the development of the package.

- Create a directory based on the name of the switch for which you are developing the package. For example, if the switch is manufactured by the Acme Switch Company, create a directory named **acme**.
- Create two subdirectories under **acme**: **src** for source files and **install** for the installable package.
- The subdirectories for **src** will depend on what parts of switch integration development are required for the Acme switch. Each major component (for example, a C process or script program) should have its own directory. In addition, one directory is needed for header files, and one directory is needed for libraries that you may create as part of switch integration development. Much of the switch package development process involves copying default files provided by the Toolkit then modifying them for use with your switch integration package; a separate directory to store these files is useful.
- Create a **pkg** subdirectory under **install**
- **pkg** subdirectories will store parts of the installable package. During the development process, you compile programs in the **src** directory and copy the results of the compilation (and any other files needed by the package) to the appropriate **pkg** subdirectories.

  The following is an example of the **pkg** subdirectory structure for the Acme switch.

Directory	Explanation
swin/data/params	required: contains switch parameters
swin/data/swinnames	required: contains switch name for administration
swin/data/swinrules	required: contains applications that work with this switch
vs/trans	needed if you write one or more scripts
acme/bin	needed if you write one or more processes
etc/conf/init.d	needed if you write one or more own processes (for inittab)

It is important to distinguish between the source directories and the target directories. The source directories (for example, **src**) contain source code, original copies of files, and other files which are not actual parts of the resulting switch package. Target directories (for example, **install**) include only those files that actually install on the customer machine.

CHOOSING OPTIONS

Chapters 5 through 9 require you to determine the capabilities of the switch in areas such as call information, message-waiting notification, transfers, and disconnects. To determine switch capabilities, you must familiarize yourself with the switch and its documentation.

Options for each switch integration feature are presented in sections Chapters 5 through 9. Read these sections and determine which option is appropriate for your particular switch. Then, follow the steps listed in that section to include the appropriate option in the package. These steps may include for example, copying a default file, changing parameters, or coding a script.

Some steps will ask to you record information on worksheets in Appendix A, *Development Aids*. Writing down information as you proceed through each step of development makes assembling the package later much easier. AT&T recommends that you make copies of the development aids in Appendix A before beginning so that the masters are always available for the development of another switch integration package.

WHEN IS CODING REQUIRED?

The Toolkit allows you to support many switches without any software development. The large number of parameters and options provided allow for great flexibility in accommodating various environments. However, you may encounter a switch which requires you to develop code to complete the switch integration package. If coding is an option for a particular component of the switch integration package, guidelines for development are presented in the chapter in which that component is discussed. The following list summarizes some of the situations in which coding may be required.

- If the switch supports a voice mail interface other than the SMDI protocol required by the SID software, you will need to develop one or more processes to handle the interface.
- If the way in which transfers are performed on the switch is significantly different from the default switch, scripts coding may be required.
- If the switch provides in-band signaling, a mechanism by which the switch generates a series of touch tones at the beginning of the call to specify calling and called parties, scripts coding may be required.
- If the switch supports message-waiting updates by dialing feature access codes, scripts coding may be required.
- If the switch requires channels to dial a feature access code to notify the switch that the channel is ready to receive phone calls, scripts coding may be required.

ASSEMBLING, INSTALLING, TESTING

A checklist is included in Appendix A, *Development Aids,* to help you verify that all features of switch integration have been considered in your package. Once you have determined that all pieces have been implemented, Chapter 10, *Installing Your Switch Integration Package*, explains how to put the package on diskette, install it on the run-time machine, and test basic functionality.

4. Development Structures

This chapter covers concepts and structures you will need throughout the development of a switch integration package. Chapters 5 through 9 will refer you as needed to the ideas presented here. However, you may want to familiarize yourself with this information before you begin coding.

- scripts and processes
- switch package administration
- tools for testing the switch package

SCRIPTS AND PROCESSES

Scripts and processes differ both in the programming language used to code them and the operations they support. Scripts consist of a series of operations that interface to the phone line; they are written in a special programming language called *native script*. Processes are written in C and are useful in interfacing to components other than the phone line, such as a voice mail serial link.

Scripts

The CONVERSANT Intro Advanced Application Toolkit document provides guidelines and examples for writing scripts. However, in order for your switch integration scripts to work properly with Voice Power applications, you must also do the following.

1. Declare the variables and structures you need for your scripts in a file using the standard C language constructs. The example below uses **vpc.c** as its variables and structures filename. Note that the only variable types supported by the script tools are the following: *int*, *char* (including array), and structures containing *int* and/or *char* types.
2. Put the directory **/toolkit/bin** in your path using the following command.

   ```
   PATH=$PATH:/toolkit/bin
   ```

3. Using the **sw_script.c** file provided by the Toolkit in the **/toolkit/src/scripts** directory and the variables and structures file you created (**vpc.c** in this example), execute `c2tas` to convert the C language variables to script variables.

   ```
   cat sw_script.c vpc.c | c2tas -I/att/include -I/att/msgipc \
   -I/toolkit/include -a 0 -o vpc.h
   ```

   NOTE

   sw_script.c must be the first in the list of files to concatenate.

4. The total amount of space used for variables and structures cannot be more than 500 bytes. Ensure that this limit has not been exceeded by looking at the last line of the **vpc.h** file. This line contains the variable with the highest address; it must be less than 500.
5. Using the #include syntax in C, include the following header files in your script source.
   - **codestyle.h**
   - **mesg.h**
   - **tsm_dip.h**
   - **swin_siscr.h**
   - **sw_scr2scr.h**
   - **vpc.h** (the file created in the previous step)
6. Put the following command in front of all the header files in the script source.

   ```
   taspp off
   ```

7. Put the following command after all the header files in the script source.

   ```
   taspp on
   ```

8. Run the following command (assuming your script name is **vpc.t**).

   ```
   taspp -I/att/include -I/att/msgipc -I/toolkit/include \
   -o vpc.o vpc.t
   ```

9. Compile the script, using the file just created, using the following command.

```
/vs/bin/tas -e -o .vpc.T vpc.o
```

The script **.vpc.T** must be moved or copied to the **/vs/trans** directory of the run-time machine. Note the name of the script; it starts with a period (.) to prevent it from showing up as a script which could be assigned to a channel. The switch integration scripts cannot be assigned to a channel; they are executed when needed by the Voice Power script assigned to the channel.

While developing your scripts, keep in mind the following.

- The information passed from the Voice Power application to your script is found in the to_info structure in the **sw_scr2scr.h** header file.
- The operations your script must perform, which depend on the type of script (call information, message-waiting, transfer, or in-service notification) may include collecting and parsing touch tones, sending messages to **swindip** to get extensions for message-waiting updates, performing transfer operations, or dialing a feature access code.
- The information your script must pass to the Voice Power application comes in several forms.
  - — The variable *from_info_type* is set to indicate which type of switch integration script is running: FROM_CALLINFO if call information, FROM_TRANSFER if transfer, or FROM_SOFTCHECK if message-waiting or in service notification.
  - — The from_info structure is populated with the appropriate information, depending on the script type.
  - — Register 3 must be set to the defined value EXECU_SWIN, which informs the application script that a switch integration script has returned to the application script.

The Toolkit provides a sample transfer script, which you may find useful as a guideline for the development of any type of switch integration script. Pay particular attention to how it receives information from the application script, how it passes information back, and how it returns control to the application. The script source file is **s75xfer.t**, and it can be found in the **/toolkit/src/scripts/** directory.

Processes

Processes are divided into two categories: data interface processes (DIPs) and pure processes. A DIP is a process with an associated *message queue* by which it receives messages from other processes. Pure processes are similar to DIPs. However, a pure process does not have a message queue (does not receive messages), though it can send messages to DIPs, and it does not call `VSstartup`. In the example design of the reader and writer in this chapter, the reader is a pure process, and the writer is a DIP.

The CONVERSANT Intro Advanced Application Toolkit recommends that you use the DynaDIP feature to name DIPs. It includes a DynaDIP template and instructions on how to compile the DIP. The DynaDIP feature allows you to use an alphanumeric name for the DIP, rather than a number. Using alphanumeric names instead of numbers to reference DIPs accomplishes several goals.

- Uniqueness of DIP names is important. Being able to use alphanumeric characters provides more possibilities for naming.
- Because the names can be meaningful, they are are easier to remember than numbers.
- Alphanumeric names minimize conflicts you may encounter with the existing DIP numbers of Voice Power products.

To compile processes, four directories containing header files are needed. Explanations of the first three directories are provided the *CONVERSANT Intro Advanced Application Toolkit* document. The last directory is specific to switch integration package development; it contains header files supplied by the Toolkit

- **/att/include**
- **/att/msgipc**
- **/att/msgipc/etmsgs**
- **/toolkit/include**

Your source code must #include the **swin_proc.h** file in **/toolkit/include**. **swin_proc.h** contains the mconts and structures (related to switch integration) that processes can send or receive. These are shown in the following table.

Mcont	Structure	Comments
CALL_INFO	swin_call_info	for sending call information to **swindip**
MWL_CTL	mwl_ctl	for receiving mwl updates and responding with status
TX_REQ	tx_req	for receiving transfer requests and responding with status
CHAN_INSERV	chan_inserv	for receiving notification that a channel is in service
PUT_LWC	put_lwc	for sending leave-word-calling messages to **swindip**
MASS_REFRESH	mass_refresh	for sending request to refresh mwls
DAY_NIGHT_SVC	day_night_svc	for sending day/night service change
SYNC_CALL	sync_call	for receiving sync message and responding

The switch integration features (Chapters 5 through 9) that use these messages explain them in detail.

CONVERSANT Intro Advanced Application Toolkit document covers the procedure for accessing a DIP by its DynaDIP name. Your switch process will interface with the DynaDIP **swindip**, available to you as the define `SWINDIP_NAME`.

For processes to operate properly on the run-time machine, you must read and follow the instructions in the *Auto Startup Via inittab* section of Chapter 6, *Data Interface Process*, of the *CONVERSANT Intro Advanced Application Toolkit* reference document. In addition, you must place the following command in your package's `Install` and `Remove` shells.

```
/bin/touch /etc/inittab
```

Refer to the *Putting the Package Together* section of Chapter 10, *Installing Your Switch Integration Package*, for instructions on how to modify the `Install` and `Remove` shells.

Reader and Writer Processes

As part of switch package development, you may have to code a *reader* process and a *writer* process. A reader process reads messages from the switch over the voice mail interface. A writer process writes messages to the switch. Separating the code into two processes allows the writer process to receive messages from all sources, including the reader process; the reader can then be dedicated to the voice mail serial link.

The reader's job is to read lines from the serial link, passing the information on to the writer. High-level pseudocode for the reader might be as follows.

```
initialize (port, variables, etc.)

while true do

    read a line from the serial link

    parse the line

    pass the information to the writer process

done
```

The writer's job is to wait on messages from any source. If a message is received from the reader, then a message must be sent to **swindip**, which serves as the switch package interface process. The writer may also receive messages from **swindip**, which causes it to write a message to the switch. The message types as they relate to each switch integration feature are explained in detail in in Chapters 5 through 9. Not all the messages shown below may be applicable to your switch. The following is possible high-level pseudocode for the writer. Note that one writer can serve all features.

```
initialize (port, variables, etc.)

while true do

    wait for message

    if message contains call information from reader,
        send call information to swindip

    if message is a transfer request from swindip,
        write transfer request packet to switch

    if message is transfer results from reader,
        send transfer results to swindip

    if message is a message-waiting update from swindip,
        write message-waiting packet to switch

        if reader will not provide message-waiting results from switch,
              send message-waiting results to swindip

    if message is message-waiting results from reader,
        send message-waiting results to swindip

    if message is leave-word-calling from reader,
        send leave-word-calling to swindip

    if message is an in-service notification from swindip,
        write in-service notification to switch

    if message is message-waiting refresh from reader,
        send message-waiting refresh to swindip

    if message is day/night service change from reader,
        send day/night service change to swindip

done
```

Based on this design, you are responsible for the reader to switch interface, the writer to switch interface, and the reader to writer interface. The writer to **swindip** interface is detailed for each message in Chapters 5 through 9.

SWITCH PACKAGE ADMINISTRATION

You may find that you need to prompt for information during the installation of the switch package and/or that some switch information needs be available to the administrator for modification purposes, for example, device type or baud rate. Using the Voice Power product's menu structure, you can create a form which can be used by the installer as well as the administrator.

The *CONVERSANT Intro Advanced Application Toolkit* describes a package, ETIP Designer Release 2.0 which you can use to develop your own administration forms. If you use this package, your administration forms will have the same look and feel as the Voice Power products.

The *CONVERSANT Intro Advanced Application Toolkit* also describes how to insert administration forms into an application package menu structure. However, you are developing a switch package, not an application. Therefore, a different mechanism is required for administration. It is done as part of the **Install** shell script of your switch package. The following is an example of how the **Install** shell script should be modified if administration forms are part of your switch integration package. In the example, the administration form is a file **acme_admin**, and it is in the directory **/acme/bin**.

```
/bin/grep "^ACME " /swin/data/si_admin >/dev/null 2>&1
if [ $? -ne 0 ]
then
      echo "ACME \"ACME Administration\" /acme/bin/acme_admin" \
      >>/swin/data/si_admin
fi
```

So that your switch package administration can be removed from the run-time environment, you must modify the **Remove** script to include the following.

```
/bin/grep -v "^ACME " /swin/data/si_admin > /tmp/si_admin$$
/bin/mv /tmp/si_admin$$ /swin/data/si_admin
```

The following is a list of brief explanations about the sections of code for the installation and removal of the switch package administration shown above.

- `ACME`: a tag for checking and removal of the line from the file
- `/swin/data/si_admin`: the file containing switch package administration entries
- `\"ACME Administration\"`: the name of the switch package administration form that is displayed on the screen
- `/acme/bin/acme_admin`: the administration form for the switch package

TOOLS FOR TESTING YOUR SWITCH PACKAGE

The Toolkit provides a few utilities you may find useful while developing and testing your switch package. The tools are designed to work on the run-time machine which has been loaded with Integrated Voice Processing System Software Release 3.0 and a Voice Power application, such as AUDIX Voice Power Release 3.0.

The tools allow you to test several of the capabilities of switch integration: call information, transfers, and disconnect detection. Although no tools are provided to test message-waiting lamps, the applications themselves are useful for these tests; in addition to refreshing the status of all extensions each time the Voice System is started, they support a background refresh which results in periodic updates of the message-waiting status of extensions with messages.

To put the tools in your switch package, create the following directories under **install/pkg**.

Directory	Explanation
vs/trans	if you have not already created this directory as part of script development
toolkit/bin	for the test software

Copy the test software to the appropriate directories by doing the following.

```
cd /toolkit/bin
cp tooldip go development-directory/install/pkg/toolkit/bin
cp call.T transfer.T development-directory/install/pkg/vs/trans
```

development-directory is the full path name of the development directory for this package.

When installing your switch package, or immediately after, create the directory **/toolkit/data** on the run-time machine by using the following command.

```
mkdir /toolkit/data
```

To use the test software, you need to specify how you want the tests to be run by filling in parameters in a file. The filename is **/toolkit/data/data_file**, and the contents consists of a line for each channel participating in the test. Each line can be in one of two forms.

- `call` *chan ext num-times length-of-call interval*
- `transfer` *chan ext num-times*

The test variables are as follows.

- *chan* is the channel number.
- *ext* is the extension to call (or to transfer to).

  The extension called might be the voice mail retrieval number, a subscriber with coverage who is busy or does not answer, or a channel of the voice system.
- *num-times* is the number of times to call (or attempt a transfer).
- *length-of-call* is the amount of time of the call (in seconds).
- *interval* is the amount of time between calls (in seconds).

The `call` instruction can be useful for verifying the call information mechanism, if the switch package supports call information, or for verifying disconnects.

The `call` instruction, using the *chan* specified, places a call to *ext*. It holds the channel for the *length-of-call* then disconnects. After waiting the *interval*, the instruction begins again. It will do this *num-times*. In this way, the `call` instruction is an effective test of call information and disconnects.

Transfer testing is useful for verifying that the system can reliably detect conditions such as busy and ring-no-answer. For switchhook flash transfers, it also verifies that transfers are initiated successfully; this may require detecting the special dial tone (stutter).

The **/toolkit/data/data_file** can contain a combination of `call` and `transfer` instructions. However, a channel can only be assigned to one instruction.

To use the test software, do the following.

1. Make sure the voice system is running.
2. Modify **/toolkit/data/data_file** with the desired testing parameters.
3. Enter **`/toolkit/bin/go`**

   This shell assigns the requested test scripts to the channels based on the entries in the **/toolkit/data/data_file** file.

4. Tests on channels assigned to `call` are initiated automatically.

   For channels assigned to the `transfer` instruction, do one of the following (depending on what you wish to test).

   a. Take the *transfer to* extension phone set off-hook to test the busy case.

   b. Leave the *transfer to* extension phone set on-hook to test the ring/no answer case. Make sure that the *transfer to* extension and *does not* have coverage.

5. For each channel assigned to the `transfer` instruction, initiate testing by calling the extension number of the channel from a phone other than the *transfer to* extension.

   After calling the channel, leave the phone off the hook so that testing can continue. If you have the `transfer` instruction assigned to more than one channel, you will have to initiate a call to each channel using a different phone.

   The channel will continually attempt to transfer the call to the extension specified.

6. As each channel completes its testing, an entry is written to the output file **/toolkit/data/out_file**; this file will show the results of the transfer testing and the number of calls made to each channel. (See sample output in this section.) For further testing modify the **/toolkit/data/data_file** and reexecute the `/toolkit/bin/go` shell.

Voice Power applications provide a report which shows the number of calls and the disposition of the calls; if you reset this report before your test, you can compare the call counts from **/toolkit/data/out_file** with the report. Refer to the the Voice Power application documentation set for more information.

NOTE

If the transfer test tools detect a successful completion of the transfer, they will quit, regardless of the number of cycles completed.

The following is a sample input file.

```
call 0 1234 1000 20 10
transfer 1 2345 1000
```

The following is a sample output file.

```
call channel 0, 1000 calls
transfer chan 1, 0 success, 1000 busy, 0 no-answer, 0 failure
```

5. Call Information Options

This chapter consists of four sections, corresponding to the four options for call information. Only one of the four sections is applicable for a particular switch package.

SWITCH INTEGRATION DEVICE SOFTWARE

The Toolkit includes the software for collecting call information from a Switch Integration Device for integrated calls. If such a device is available for the switch for which the switch package is being developed, please follow the instructions below on how to use this software. If no such device is available, please disregard this section and proceed to the next section for other possible ways to collect call information.

Components of the Switch Integration Device Software

The switch integration device software consists of two processes which read and write to the SID and an administration form with an associated help file. The following is a list of the SID software components.

Component	Description
/toolkit/sid/bin/sidrdr	Process which reads from the SID
/toolkit/sid/bin/sidwtr	Process which writes to the SID
/toolkit/sid/bin/sid_admin	Form to administer SID connections
/toolkit/sid/txt/h_sid.txt	Help file for administration form

Interface Supported by the Switch Integration Device Software

The Switch Integration Device software provided by the Toolkit implements the Centrex Simplified Message Desk Interface (SMDI) protocol. The messages for call information are defined as follows.

```
Physical link: RS232C, administrable baud rate, 7 bits, even parity, 1 stop bit.

Call Information Packet Format (messages from SID to voice mail system):

    Inside two party call:

      [cr][lf]MDdddttttaxxxxxxx[bl]yyyyyyy[bl][cr][lf][^y]


    Outside two party call:

      [cr][lf]MDdddttttaxxxxxxx[bl][bl][cr][lf][^y]


    Inside direct call:

      [cr][lf]MDdddtttta[bl]yyyyyyy[bl][cr][lf][^y]


    Outside direct call:

      [cr][lf]MDdddtttta[bl][bl][cr][lf][^y]



    [cr] - carriage return
    [lf] - line feed
    [bl] - blank
    [^y] - control-y

    ddd - message desk number (000 to 063, ignored by SID software)
    tttt - logical terminal number (0001 to 2047, 1 more than channel number)
    xxxxxxx - called party
    yyyyyyy - calling party
    a=D - direct call (inside or outside)
    a=A - forward or send all calls (inside or outside two party call)
    a=B - busy (inside or outside two party call)
    a=N - no answer (inside or outside two party call)
```

How to Use the Switch Integration Device Software

In order to use the SID software, do the following steps.

1. The software provided by the Toolkit under the **/toolkit/sid** directory must be installed under the **install/pkg/sid** directory. This can be accomplished with the following commands.

   ```
   cd /toolkit
   find sid -print | cpio -pdumv development-directory/install/pkg
   ```

   development-directory is the full path name of the development directory for this package.

2. When you install your package on the run-time system, the empty directory **/sid/data** must be created, using the following command in the **Install** shell script.

   ```
   mkdir /sid/data
   ```

3. The administration form **/sid/bin/sid_admin** must be executed to set up the **sidrdr** and **sidwtr** processes. This form allows the person installing your switch integration package to specify information about the interfaces to the SID. For each SID (there can be more than one, depending on the switch and the number of Voice Power applications installed on the system), you specify the *tty* port used to connect to the SID, the baud rate of the connection, and any notes in the comment field you wish to include.

   The administration form needs to be installed; refer to the *Switch Package Administration* section of Chapter 4, *Development Structures,* for details.

4. The first line of the `params` file must be set to the following.

   ```
   callinfo 15 3 2
   ```

   Write the above information on the first (call information) line in the *Params Worksheet* of Appendix A, *Development Aids*.

`callinfo`	Indicates that the switch package is integrated with the switch for call information
`15`	The number of seconds that call information from the SID is considered usable (so that old call information is not used by a new call)
`3`	The number of times an application should retry when asking for call information, if it is not present initially
`2`	The amount of time the application should wait between the retry attempts while asking for call information.

IN-BAND SIGNAL HANDLING

If the switch for which the switch package being developed provides in-band signaling for call information, please read this section for details on how to make use of that information. If not, please turn to the next section for other possible ways of collecting call information.

You must develop a script to collect the touch tones generated by the switch at the beginning of a call. For information on the development of scripts, see Chapter 4, *Development Structures*.

To specify that you are using a script to collect call information, the first line of the **params** file must be set to the following.

`callscript` *script-name*

script-name is the name of the call information script (without the .T) in the **/vs/trans** directory. Write the above information on the first (call information) line in the *Params Worksheet* of Appendix A, *Development Aids*.

Your script uses the following information in the to_info structure from the application script.

Field	Description
`scriptname`	application script name
`talkfile`	talkfile containing phrase for half-second wait time
`halfsecond`	phrase number to talk for half-second wait time

When the call information script has finished its processing, it must do the following to pass the information back to the application script.

1. Set the *from_info_type* variable to FROM_CALLINFO.
2. Populate the following fields in the from_info structure.

Field	Possible Values
`instruction`	GETCALLINFO if call information collected NOINFO if did not receive call information OOSINFO if switch indicated that the channel is out of service NOCALL if touch tones indicated there was no call
`cgext`	calling party extension, null string if not known
`cgname`	calling party name, null string if not known
`cdext`	called party extension, null string if not known
`cdname`	called party name, null string if not known
`reason`	RDIRECT if direct call RCOVERAGE if forwarded call
`cgloc`	CINSIDE if inside call

	COUTSIDE if outside call
`covtype`	COV_BUSY if busy COV_NA if no answer COV_FWD if forwarded
`ci_dial_tone`	CI_DIAL_TONE_YES if dial tone detected while getting call information CI_DIAL_TONE_NO if dial tone not detected CI_WINK_DETECT if wink received while getting call information

3. Set register 3 to the defined value EXECU_SWIN.

```
load (r.3, im.EXECU_SWIN)
```

4. Execute the application script.

```
execu (ch.ti__scriptname, 1)
```

In addition to collecting call information, your script can also send three messages (structures defined in **swin_comm.h**) to **swindip**.

Mcont	Structure	Comments
PUT_LWC	put_lwc	leave-word-calling feature of switch `sendext` is the sending extension `rcvext` is the receiving extension
MASS_REFRESH	mass_refresh	switch wants all message-waiting status refreshed no parameters with this message
DAY_NIGHT_SVC	day_night_svc	switch indicates change in day/night service `param` is DAY_SVC_PARAM if day service `param` is NIGHT_SVC_PARAM if night service

These messages are sent to **swindip** using the `dbase` instruction. Use the defined value SWINDIP_NAME as the destination of the message. Note that none of these messages are returned by **swindip**; set the response interval to 0 with the `nwitime` script instruction.

WRITING YOUR OWN CALL INFORMATION PROCESS

If the switch supports a voice mail interface which can be accessed via a serial link, you will want to develop your own call information software. How the software interfaces to the switch is your responsibility; you will need to understand the protocol provided by the switch. You are also responsible for determining the physical connection between the switch and the voice mail system.

Refer to Chapter 4, *Development Structures*, for guidelines on the organization of call information software.

To specify that your package contains a process for call information, write the following on the first (call information) line in the *Params Worksheet* of Appendix A, *Development Aids*.

`callinfo` *aging retries interval*

aging the window of seconds in which call information is considered valid (0 means the call information is always valid)

retries The number of times a Voice Power script should request call information if it has not been received

interval The amount of time the script should wait between requests

Your process sends the call information to **swindip** using the mcont CALL_INFO and the structure swin_call_info. The following table describes the fields of this structure (in addition to the standard header, including *mchan*).

Field	Description
chext	phone number of channel receiving call information; nullstring if unknown
cgext	calling party extension; null string if unknown
cgname	calling party name; null string if unknown
cdext	called party extension; null string if unknown
cdname	called party name; null string if unknown
reason	RDIRECT if direct call, RCOVERAGE if forwarded
cgloc	CINSIDE if inside call, COUTSIDE if outside call
covtype	COV_BUSY if busy, COV_NA if no answer, COV_FWD if forwarded

If your call information process needs to be notified by the Voice Power script so that it can finish processing the call before the script can start talking, specify the following on the last (sync) line of the *Params Worksheet* of Appendix A, *Development Aids*.

`sync` *dip-id*

dip-id is the DynaDIP identifier for your process. If your call information process does not need to be notified by the Voice Power script, the last line should read `nosync`.

If the sync line is set to `sync` *dip-id*, your process will receive a message from the Voice Power script after it has retrieved the call information. This will cause the script to wait until it receives a response from your process (actually, the message is passed through **swindip** in each direction). When your process replies to this message, after it has done the necessary processing, it should send a message to the channel via **swindip** with a response indicating whether the channel should continue with the call or should hang up (in case the additional processing detected the original caller had hung up). The following messages are involved.

Mcont	Structure	Comments
SYNC_CALL	sync_call	sent by script to **swindip** to process
SYNC_CALL	sync_resp	sent by process to **swindip** to script, `g_param` is SYNC_CONTINUE if the channel should continue with call `g_param` is SYNC_HANGUP if the channel should hangup

NO CALL INFORMATION

If none of the above options for call information apply to the switch, you will have to operate the Voice Power application in its non-integrated mode. This means callers will have to reenter the extension to leave a message; subscribers retrieving messages must enter their own extension.

In order to specify that the switch package does not support call information, write the following on the first (call information) line in the *Params Worksheet* of Appendix A, *Development Aids*.

```
nonintegrated
```

NOTE

The AUDIX Voice Power System R3.0 documentation set does not cover information, for example prompt sequences, pertaining to the non-integrated mode.

6. Message-Waiting Options

This chapter consists of four sections, corresponding to the four options for message-waiting notification. Only one of the four sections is applicable for a particular switch package.

The Toolkit provides the file **active_mwl** which contains the default parameters for the AT&T System 75 PBX. It can be found in the **/toolkit/src/files** directory. You need to make a copy of this file (keeping the name **active_mwl**) and modify it as needed for your package. If you develop a script to perform message-waiting updates you may need to modify the first three lines. The fourth line applies for all message-waiting options; it indicates whether to suppress the default parameters for the on and off codes on the MESSAGE WAITING PARAMETERS form. The following table shows the **active_mwl** default file.

Line number	Default Value	Possible Values	Comments
1	`0`	integer, `any`	channel number used to perform updates
2	`*4`	phone pad sequence	sequence for message-waiting on
3	`#4`	phone pad sequence	sequence for message-waiting off
4	`variable`	`variable` or `fixed`	display on/off codes on form

SWITCH INTEGRATION DEVICE SOFTWARE

The Toolkit includes the software for performing messaging-waiting updates through the Switch Integration Device. It is the same software described in Chapter 5, *Call Information Options*, for call information. If such a device is available for the switch for which the switch package is being developed, please follow the instructions below on how to use this software. If no such device is available, please disregard this section and proceed to the next section for other possible ways to perform message-waiting updates.

Components of the Switch Integration Device Software

The software consists of two processes which read and write to the SID and an administration form with an associated help file. The following is a list of the SID software components.

Component	Description
/toolkit/sid/bin/sidrdr	Process which reads from the SID
/toolkit/sid/bin/sidwtr	Process which writes to the SID
/toolkit/sid/bin/sid_admin	Form to administer SID connections
/toolkit/sid/txt/h_sid.txt	Help file for administration form

Interface Supported by the Switch Integration Device Software

The Switch Integration Device software provided by the Toolkit implements the Centrex Simplified Message Desk Interface (SMDI) protocol. The messages for message-waiting are defined as follows.

```
Physical link: RS232C, administrable baud rate, 7 bits, even parity, 1 stop bit.

Message Waiting Packet Format (messages from voice mail system to SID):

    Turning on message waiting:

      OP:MWI[bl]xxxxxxx![^d]


    Turning off message waiting:

      RMV:MWI[bl]xxxxxxx![^d]


Invalid Station Number Packet (message from SID to voice mail system):

      [cr][lf]MWIxxxxxxx[bl]INV[cr][lf][dl][dl][^y]



    [cr] - carriage return
    [lf] - line feed
    [bl] - blank
    [^y] - control-y
    [^d] - control-d
    [dl] - deletion character (hex ff)


    xxxxxxx - station number
```

How to Use the Switch Integration Device Software

In order to use the SID software, do the following steps.

NOTE

Skip steps 1 through 3 if you already installed the Toolkit's SID software during the call information option development.

1. The software provided by the Toolkit under the **/toolkit/sid** directory must be installed under the **install/pkg/sid** directory. This can be accomplished with the following commands.

   ```
   cd /toolkit
   find sid -print | cpio -pdumv development-directory/install/pkg
   ```

   development-directory is the full path name of the development directory for this package.

2. When you install your package on the run-time system, the empty directory **/sid/data** must be created, using the following command.

   ```
   mkdir /sid/data
   ```

3. The administration form **/sid/bin/sid_admin** must be executed to set up the **sidrdr** and **sidwtr** processes. This form allows the person installing your switch integration package, to specify information about the interfaces to the SID. For each SID (there can be more than one, depending on the switch and the number of Voice Power applications installed on the system), you specify the *tty* port used to connect to the SID, the baud rate of the connection, and any comments you wish to include.

   The administration form needs to be installed; refer to the *Switch Package Administration* section of Chapter 4, *Development Structures,* for details.

4. The fifth line of the **params** file must be set to the following.

   ```
   digital_mwl 28
   ```

   Write the above information on the fifth (message-waiting) line in the *Params Worksheet* of Appendix A, *Development Aids*.

   `digital_mwl` Indicates that the switch package uses a digital link (to the SID) for message-waiting updates

   `28` The DIP number of the **sidwtr** process which receives message-waiting messages from **swindip** and writes them on the link to the SID

5. Change the fourth line of the **active_mwl** file to `fixed`, indicating that the on and off parameters are not applicable and should not be presented to the administrator on the MESSAGE WAITING PARAMETERS form.

SCRIPT FOR MESSAGE-WAITING

If the switch for which the switch package is being developed requires an analog channel to dial feature access codes to turn message-waiting indicators on or off, please read this section for details on how to develop a script to perform these functions. If not, please turn to the next section for information on other possible ways of performing message-waiting updates. For information on the development of scripts, see the Chapter 4, *Development Structures*.

To specify that you are using a script to perform message-waiting updates, the fifth line of the **params** file must be set to the following.

`analog_mwl` *script-name*

script-name is the name of the message-waiting script (without the .T) in the **/vs/trans** directory. Write the above information on the fifth (message-waiting) line in the *Params Worksheet* of Appendix A, *Development Aids*.

All four lines of the **active_mwl** file are important for your message- waiting script. The first line indicates which channel must be used to perform the updates. It can either be a number, such as `0`, if only that channel performs updates, or `any`, if any channel can be used. For some switches, if a phone (channel) turns a lamp on, only the same phone (channel) can turn it off. If this is the case with your switch, you must specify a single channel for message-waiting updates.

The second and third lines of the **active_mwl** file are the sequences to turn on message-waiting lamps and to turn off message-waiting lamps, respectively. These two sequences are passed to the script when it wants to perform a message-waiting update.

If the fourth line of the **active_mwl** file is set to `variable`, on/off sequences can be changed by the administrator of the voice mail system via the MESSAGE WAITING PARAMETERS form. AT&T recommends that you use the `variable` setting if MWL on/off sequences can be changed on the switch. If you change the fourth line of the **active_mwl** file to `fixed`, the on/off parameters are not presented on the MESSAGE WAITING PARAMETERS form and therefore cannot be changed by the administrator.

The only information your script uses from the application script is its name, which is found in the `scriptname` field of the to_info structure. When the message-waiting script has finished its processing, it must do the following to pass the information back to the application script.

1. Set the *from_info_type* variable to FROM_SOFTCHECK.
2. Populate the following fields in the from_info structure.

Field	Description
`softtype`	set to SOFT_MWL
`mwl_success`	count of successful message-waiting updates
`mwl_failure`	count of failed message-waiting updates

3. Set register 3 to the defined value EXECU_SWIN.

```
load (r.3, im.EXECU_SWIN)
```

4. Execute the application script.

```
execu (ch.ti__scriptname, 1)
```

In order to perform message-waiting updates, the script must send messages to **swindip** to obtain the extensions to be updated. The following messages (structures defined in **swin_siscr.h**) are sent between the script and **swindip**.

Mcont	Structure	Comments
GET_MWL	get_mwl	(to **swindip**) get an extension to be updated `mwlstat` is NO_PREV_HELD if this is first request for script `mwlstat` is PREV_MWL_SUCCESS if second or later, previous were successful `mwlstat` is PREV_MWL_HELD if second or later, previous status unknown (maximum 50 updates can be held per call)
MWLINFO	mwlinfo	(from **swindip**) extension to be updated `mwlaction` is MWL_ON for ON, MWL_OFF for off, or MWL_NONE if done `mwlext` is null-terminated string of extension `mwlon` is sequence to turn on mwl `mwloff` is sequence to turn off mwl
DONE_MWL	done_mwl	(both ways) script says it is done
FAIL_MWL	fail_mwl	(both ways) script says it failed to do mwls, all pending updates are requeued

These messages are sent to **swindip** using the `dbase` instruction. Use the defined value SWINDIP_NAME as the destination of the message.

The following pseudocode shows how the messages are used by the script.

```
send GET_MWL with NO_PREV_HELD to swindip

    if response is MWL_NONE,
        then goto successful_end

    perform mwl operation

    if only one mwl update per call,
         if successful,
             goto successful_end
         else
             goto failure_end
loop:

    /* once a failure is detected, no more updates can be done */
    if previous update failed,
        goto failure_end

    /* if the script knows the previous update succeeded, it tells */
    /* swindip so it doesn't have to keep remembering all the extensions */
    if previous update successful,
        send GET_MWL with PREV_MWL_SUCCESS to swindip

    /* if the script won't know about the success of failure of any of */
    /* the extensions until the end, it tells swindip to keep remembering */
    /* the extension, in case it fails at the end; this allows swindip to */
    /* requeue all pending mwl updates. */
    if status of previous update unknown yet,
        send GET_MWL with PREV_MWL_HELD to swindip

    if response is MWL_NONE,
        then goto successful_end

    perform mwl operation

    if script can still do more, and haven't reached limit of 50
    pending extensions,
        goto loop

successful_end:
    send DONE_MWL to swindip

    exec the application script

failure_end:
    send FAIL_MWL to swindip

    exec the application script
```

WRITING YOUR OWN MESSAGE WAITING PROCESS

If the switch supports a voice mail interface which can be accessed via a serial link, you will want to develop your own message-waiting software. How that software interfaces to the switch is your responsibility; you will need to understand the protocol provided by the switch. You are also responsible for determining the physical connection between the switch and the voice mail system.

For information on the development of processes, see the Chapter 4, *Development Structures*.

To indicate that you are providing a process to perform message-waiting updates, write the above information on the fifth line (message-waiting) in the *Params Worksheet* of Appendix A, *Development Aids*.

`digital_mwl` *dip-id*

dip-id is the DynaDIP name of the process.

Change the fourth line of the **active_mwl** file to `fixed`, indicating that the on and off parameters are not applicable and should not be shown on the MESSAGE WAITING PARAMETERS form.

Your message-waiting process will receive messages from **swindip** with mcont set to MWL_CTL and with the structure mwl_ctl. This structure contains two fields of information for your process (in addition to the standard header, including *mchan*).

Field	Description
`mwlext`	extension to be updated for message-waiting indicator
`mwlaction`	MWL_ON if to be turned on, MWL_OFF if to be turned off

Your process then replies to **swindip** using the same message, the same structure, and the following fields populated with data (in addition to the standard header, including *mchan*).

Field	Description
`mwlext`	extension updated for message-waiting indicator
`mwlaction`	MWL_ON if turned on, MWL_OFF if turned off
`mwl_stat`	MWL_SUCCESS if successful, MWL_FAILURE if failed

NO MESSAGE WAITING

If none of the above choices for message-waiting apply to the switch, you will have to indicate to the Voice Power application that message-waiting updates are not supported. This means callers will have to rely on outcalling or some other feature of the Voice Power application as notification of messages received.

In order to specify that the switch package does not support message-waiting updates, write the above information on the fifth line (message-waiting) in the *Params Worksheet* of Appendix A, *Development Aids*.

```
no_mwl
```

Change the fourth line of the **active_mwl** file to `fixed`, indicating that the on/off parameters are not applicable and should not be shown on the MESSAGE WAITING PARAMETERS form.

7. Transfer Options

This chapter covers the types of transfers and the options for handling them in a switch integration package.

TRANSFER TYPES

Voice Power applications support three types of transfers. The type of transfer used in a given situation depends on the information the application has about the number to which the call is being transferred.

- **Blind**: This type of transfer is used when the number to which the call is being transferred is a subscriber who has been administered for switch call coverage. Switch call coverage means that if the subscriber is on the phone or does not answer the phone, the switch routes the call to another destination, such as the voice mail system. When the voice mail system performs a blind transfer, it does not wait for call progress tones from the switch to indicate the status of the call. After initiating the transfer and dialing the number, it simply drops off the call.
- **Semi-intelligent**: This type of transfer is used when the voice mail system does not know whether the number being transferred to has switch call coverage or not. After initiating the transfer and dialing the number, the voice mail system stays on the line to determine the status of the call. If busy is detected, the call is returned to the calling party and the voice mail system informs the caller that the number is busy. If ringing is detected, the voice mail system drops off the call; this allows the caller to hear the ringing. If voice energy is detected (someone answers the phone), the voice mail system informs that person that a call is being transferred to him or her, then drops off the call.
- **Intelligent**: This type of transfer (sometimes referred to as supervised) is used when the voice mail system knows the number being transferred to does *not* have switch call coverage. If busy is detected, the call is returned to the calling party and the voice mail system informs the caller that the number is busy. If ringing is detected, the voice mail system stays on the line until someone answers the phone or until a specified number of rings (administerable) is reached. If the limit of rings is reached, the call is returned to the calling party and the voice mail system informs him or her that the called party is not available. If voice energy is detected (someone answers the phone), the voice mail system informs that person that a call is being transferred to him or her, then drops off the call.

For each type of transfer, you must choose the method by which the transfer is performed.

The most common method of performing a transfer uses the switchhook flash. To interface with most switches, the voice mail system simply performs the transfer operation the way a person would transfer a call from his or her analog phone to another extension: depress and release the switchhook (producing a flash), listen for a special dial tone (also known as *stutter*), dial the transfer to number, and hang up.

A digital link is another method of indicating to the switch that a call is being transferred. Using a process, messages exchanged between the voice mail system and the switch are used to initiate transfers and to report the status of transfers. Even if digital transfers are supported, you still may want to use the analog transfer mechanism as a backup in the event that the digital link goes down or is unavailable. The Toolkit allows you to specify a backup option.

The Toolkit also allows you to indicate that your switch package does not support transfers for one or more of the three types described above.

SWITCHHOOK FLASH TRANSFER

In order to perform transfers of any type using the switchhook flash, a number of files and parameters must be set by the switch package. These include the following.

- length of the flash
- call progress tone pattern
- call progress tone timing
- call progress tone frequencies
- sequences to initiate and complete transfers
- sequences for recovering from busy or no answer calls

Call progress tones indicate call status, for example, dial tone, stutter dial tone, busy, ringing.

Flash Duration

The flash duration is the length of time (in milliseconds) that the switchhook is depressed before it is released. The default for flash duration is 500 milliseconds; you can specify any value between 300 to 1550 milliseconds.

1. The flash duration is indicated in the **userTunable** file; a default file is available to you in the **/toolkit/src/files** directory. If you need to modify the flash value, make a copy of the default file (keeping the name **userTunable**) and change the value on the second line.

   NOTE

   You may have already made a copy of the **userTunable** file in order to modify wink disconnect interval in Chapter 8, *Disconnect Options*. If so, make the flash duration change in that same copy.

2. Indicate that you will be providing a **userTunable** file by writing a p on the seventh line (userTunable flag) of the *Registration File Worksheet* in Appendix A, *Development Aids*. (You may have already done this as part of Chapter 8, *Disconnect Options*.)

Call Progress Tone Detection

In order to detect the various call progress tones required for switchhook flash transfers, several steps are required.

1. Determine the frequencies of the tones. Some switches use the same frequencies as the default switch (AT&T System 75 PBX). If so, the default frequencies file can be used. If not, you must specify the frequencies for each of the tones. And for each combination of the frequencies, you must specify whether the combination indicates the possibility of a particular call progress tone.
2. Specify the range of allowed values for the timing of the tones, such as the length of the tone and the length of the silence between tones.
3. Specify the pattern which matches the tone, so that the tone can be recognized by the system.

Call Progress Tone Frequencies

The default frequencies for call progress tones are as follows:

```
350
    \
      dial tone
    /
440
    \
      busy
    /
480
    \
      ring tone
    /
620
```

If these default frequencies match your switch, you will probably not have to modify the filter frequencies or their corresponding trigger values, which are described in the next section. The Toolkit allows you to specify up to five different frequencies. In fact, the default settings use all five: the first two are used for dial tone only; the third and fourth are for busy, and the fourth and fifth are for ring tone. Defining the first two as dial tone only actually allows the default switch settings to detect dial tone even if the switch does not use 350 and 440 frequencies for dial tone, for example, because of dial tone training.

If your switch does not match the default settings, for example, frequencies other than 440 and 480 for busy, frequencies other than 480 and 620 for ring tone, or more than two frequencies for dial tone, you will need to define your own frequencies.

To define the frequencies, execute the program **`/toolkit/bin/coef`**. It will prompt you for each of the five possible frequencies and the range for each frequency. The range is used to determine the limits of an acceptable frequency. For example, if the frequence is 480, and you want the accepted range to be 477 to 483, enter 3 for the range. If you have fewer than five frequencies to define, enter 0 for both the frequency and the range of unused frequencies.

The default frequencies are as follows.

Filter	Frequency	Range
1	350	1
2	440	1
3	440	3
4	480	3
5	620	3

When the program is complete, it will create a file named **filters**. You will need this file when you put the switch package together. If you are using the default **filters** file, write a `-` (dash) on the sixth line (filters flag). in the *Registration File Worksheet* in Appendix A, *Development Aids*. If you are create a **filters** file using `/toolkit/bin/coef`, write a `p` on the sixth line (filters flag). in the *Registration File Worksheet* in Appendix A, *Development Aids*. This indicates you will be providing your own **filters** and **triggers** files. The contents of the file consists of four floating point numbers for each frequency which are used internally to detect the specified frequency.

Dial tone training is the ability of the switch to soft seize a channel and analyze the dial tone for two frequencies which are then used for dial tone detection. Note that if dial tone training is enabled, it will overwrite the first two frequencies definitions. The default switch parameters are set to do dial tone training when the voice system is started; the frequencies for busy and ring-tone are not affected by dial tone training. On the third line of the *Registration File Worksheet* in Appendix A, *Development Aids*, write `Y` to enable dial tone training or `N` to disable dial tone training. The fourth line allows you to indicate whether the administrator of the voice mail system can change the dial tone training option as initially set by the switch package. Write `Y` to enable the administrator to turn dial tone training on or off. Write `N` to prevent the administrator from changing the dial tone training setting.

Dial Tone Training	Administrator Control	Result
Y	Y	Dial tone training initially on but administrator can override setting at any time
Y	N	Dial tone training initially on, setting cannot be changed
N	Y	Dial tone training initially off, but administrator can override setting at any time
N	N	Dial tone training initially off, setting cannot be changed

Call Progress Tone Triggers

The five frequencies defined in the previous section correspond to 32 combinations of frequencies, some of which represent valid call progress tones. For example, for the default switch parameters, the combination of 480 and 620 represents the ring-tone. However, 440 and 620 do not represent a valid call progress tone.

To simplify the process of defining trigger combinations, execute the program **`/toolkit/bin/trigger`**. For each of the 32 possible combinations, it displays the combination number (0 - 31), followed by the combination of the filter numbers for the five frequencies defined in the previous section. You select the signal that the combination represents by entering the appropriate character. The choices are as follows.

Signal	Description
b for busy d for dial tone r for ringing	these three are used for call progress tone detection during analog call transfer
f for faxcng	normally not defined initially— an application may define it temporarily to detect the fax tone at the beginning of the call
i for interflow 4 for 480	these two are defined for single frequency tones to be recognized as valid sounds, but not used in any way by the package. They do not get reported as call progress tones or human speech.

The default switch parameters, based on the default filter frequencies presented in the previous section, are as follows.

Trigger	Signal	Comments
0	0	no frequencies
1	0	350 (trained) only
2	0	440 (trained) only
3	d	350 (trained) and 440 (trained)
4	i	440 only
5	i	440 and 350 (trained)
6	i	440 and 440 (trained)
7	d	350 (trained), 440 (trained), and 440
8	4	480 only
9	0	350 (trained) and 480
10	0	440 (trained) and 480
11	d	350 (trained), 440 (trained), and 480
12	r	440 and 480
13	r	350 (trained), 440, and 480
14	r	440 (trained), 440, and 480
15	r	350 (trained), 440 (trained), 440, and 480
16	0	620 only
17	0	350 (trained) and 620
18	0	440 (trained) and 620
19	d	350 (trained), 440 (trained), and 620
20	0	440 and 620
21	0	350 (trained), 440, and 620
22	0	440 (trained), 440, and 620
23	d	350 (trained), 440 (trained), 440, and 620
24	b	480 and 620
25	b	350 (trained), 480, and 620
26	b	440 (trained), 480, and 620
27	b	350 (trained), 440 (trained), 480, and 620
28	0	440, 480, and 620
29	0	350 (trained), 440, 480, and 620
30	0	440 (trained), 440, 480, and 620
31	0	350 (trained), 440 (trained), 440, 480, and 620

Note that some trial and error may be necessary to reliably detect the call progress tones in the presence of noise.

When the program is complete, it produces the file **triggers** which you will need later when you put your switch package together. The contents of this file are strings representing the possible call progress tones. If you create a **triggers** file, using `/toolkit/bin/trigger`, refer to the *Registration File Worksheet* in Appendix A, *Development Aids*, and verify that you wrote a `p` on the sixth (filters flag) line. This indicates you will be providing your own **filters** and **triggers** files.

Call Progress Tone Timing

Call progress tone timing is the detection ranges for on and off cycles of a tone. In the default switch (AT&T System 75), for example, the busy tone is a series of on and off intervals of 500 milliseconds. Due to peculiarities in the detection algorithms, the on interval tends to be reported shorter than the actual, while the off interval tends to be longer. Therefore, the timing for busy is defined as 250 to 500 milliseconds on, 500 to 750 milliseconds off. The complete default timing values are as follows.

Number	Min on	Max on	Min off	Max off	Comments
1	500	2000	3500	4500	(ring tone)
2	250	500	500	750	(busy tone)
3	100	250	250	450	(reorder)
4	200	575	2550	3700	(faxcng tone)
5	1000	3000	0	0	(dial tone)
6	5	500	5	500	(stutter dial tone)

The faxcng timing is not switch dependent and should therefore be maintained as shown. Note that to represent a steady tone (dial tone), the min off and max off are set to 0.

NOTE

The default timing file only contains 6 timing entries. However, you can define up to 16.

The Toolkit includes a default file **cad.timing**, in the directory **/toolkit/src/files**. If you need to make modifications, copy the default file first (keeping the name **cad.timing**), then edit the copy. If you modify this file, write a `p` on the fifth line (cadence flag) in the *Registration File Worksheet* in Appendix A, *Development Aids*. This indicates you will be providing your own **cad.timing** and **cad.pattern** files.

If you are using the default **cad.timing** and **cad.pattern** files, write a `d` on the fifth line (cadence flag) in the *Registration File Worksheet* in Appendix A, *Development Aids*.

Call Progress Tone Patterns

The pattern for call progress tones specifies how many intervals of the timing defined in the previous section represent a reported call progress tone. For example, the default stutter dial tone consists of three short bursts of dial tone followed by steady dial tone. Using the timing table above, this would be three intervals of timing number 6 followed by one of timing 5.

You can define a total of eight patterns. The patterns consist of one or more pairs (limit 16) of tones and timings, followed by the call progress tone actually reported. The tones can be one of the four reportable tones from the **triggers** file: ring (for ring tone), busy, faxcng, and dial (for dial tone). The timing value is one of the timing numbers defined above.

The call progress tone to be reported based on the pattern can be one of the following: ring (for ring tone), busy, reorder (also known as "fast" busy), faxcng, dial (for dial tone), and stutter (also known as special dial tone). The default pattern file is as follows.

Pattern	Tone
ring.1	ring
busy.2 busy.2	busy
busy.3 busy.3	reorder
faxcng.4 faxcng.4	faxcng
dial.5	dial
dial.6 dial.6 dial.6 dial.5	stutter

The faxcng timing is not switch dependent and should therefore be maintained as shown.

The Toolkit includes the default patterns file **cad.pattern**, in the **/toolkit/src/files** directory. If you need to make modifications, copy the default file first (keeping the name **cad.pattern**), then edit the copy. If you modify this file, verify that you wrote a p on the fifth line (cadence flag) in the *Registration File Worksheet* in Appendix A, *Development Aids*. This indicates you will be providing your own **cad.timing** and **cad.pattern** files.

Transfer Sequences

When switchhook flash transfer is used for any of the three types of transfers (blind, semi-intelligent, or intelligent), you must first determine if the default transfer script will work with your switch. The default transfer script was designed to work with the AT&T System 75 PBX. However, with modifications to the sequences it uses, the default transfer script works with many other switches as well.

Transfer sequences include the following operations.

- initiating a transfer
- completing a transfer
- conferencing caller and called parties
- recovering from a busy number
- recovering from a no-answer number
- recovering when no call progress tones are detected

In addition to the above sequences, two timer values can be modified.

- time to wait for call progress tones after initiating a transfer
- time to wait for call progress tones while recovering from busy, no answer, or no detection of call progress tones.

Sequences consist of the following characters.

Character	Explanation
f	perform a switchhook flash
F	perform a switchhook flash and wait for one of: S: stutter dial tone D: dial tone N: no call progress tones
e	dial the extension or number being transferred to
h	hang up
p	half-second pause
t	talk transferred to phrase
g	talk conference phrase
0-9, A-D, *, #	dial the indicated touch tone

The phrase of words that an application speaks to the called party while the calling party is on hold is the *transferred to phrase*. For example, "A call is being transferred to you. Please wait." The *conference phrase* is the words an application speaks to both parties after they have been conferenced together. For example, "You may go ahead now" or a barely audible tone.

Each sequence can be up to fifteen characters. You should be aware of some special actions of these sequences.

- If your switch does not support the conference feature, the conference sequence should end with an h indicating that a hangup was done instead.
- Recovery from failure to receive call progress tones can either be a successful transfer or an unsuccessful transfer. For example, sometimes no call progress tones are detected when the called party answers quickly but does not speak loudly enough to cause the voice system to detect that the party answered. If the transfer is to be considered successful, the sequence should include both the conference and hang up operations. If the transfer is to be unsuccessful, then the sequence should include whatever operations are required to return to the original caller.

The default sequences are defined as follows.

Sequence	Operation	Explanation
FSe#	initiate	flash, wait for stutter, dial number, then #
h	complete	hangup
tfg	conference	talk to called party, flash, talk to both
fppFN	busy	flash, wait a second, flash, make sure no call progress tones
fppFN	no answer	flash, wait a second, flash, make sure no call progress tones
tfgh	no tones	conference then hangup (treat as successful call)

The default values for waiting for call progress tones are the following.

7 seconds after initiating transfer
3 seconds after recovering from busy or no answer

The sequences and timing values are entries in a file: **active_xfer**. The Toolkit includes the default file **active_xfer**, in the **/toolkit/src/files** directory. If you need to make modifications, copy the default file first (keeping the name **active_xfer**), then edit the copy.

If **active_xfer** file (as is or with modified transfer sequences) works for your switch, enter the following information on the *Params Worksheet* in Appendix A, *Development Aids*.

- line 2: `analog_blind default`
- line 3: `analog_semi default`
- line 4: `analog_intel default`

Transfer Scripts

If you find that your switch does not conform to the operations described in the previous section of this document (using the **active_xfer** file as is or with modified transfer sequences), you must develop a script to perform switchhook flash transfers. For information on the development of scripts, see the Chapter 4, *Development Structures*.

To specify that you are providing your own transfer script(s), write the following information on the *Params Worksheet* in Appendix A, *Development Aids*.

- line 2: `analog_blind` *script-name*
- line 3: `analog_semi` *script-name*
- line 4: `analog_intel` *script-name*

script-name is the name of the script supplied by the switch package to perform switchhook transfers (without the ".T"). *script-name* can be, but is not required to be, the same for each of the three types of transfers.

For call information and message-waiting, the only information scripts use from the application script is its name, which is found in the `scriptname` field of the to_info structure. For transfers, in addition to the name of the script, all of the remaining fields of the to_info structure are passed from the application script to your transfer script. The following table presents the fields and their explanation.

Field	Description
`number`	phone number to be transferred to
`type`	transfer type: XFER_BLIND for blind transfer XFER_SEMI_INT for semi-intelligent transfer XFER_INTEL for intelligent transfer
`xfer_init`	sequence to initiate a transfer
`xfer_dropoff`	sequence to complete a transfer
`xfer_conf`	sequence to conference calling and called parties
`xfer_busy`	sequence to retrieve a busy call
`xfer_na`	sequence to retrieve a no-answer call
`xfer_notones`	sequence to retrieve or complete a call if no call progress tones are heard
`notones_timout`	number of seconds to wait for call progress tones
`daft_timout`	number of seconds to wait for no call progress tones
`num_rings`	number of rings to count for intelligent transfers
`talkfile`	talkfile containing application phrases spoken during transfers
`halfsecond`	phrase to talk for half-second wait time
`beingtransferred`	phrase to talk to called party during transfer
`goahead`	phrase to talk to calling and called parties when conferenced

The transfer sequences described above are the same as those in the *Transfer Sequences* section of this chapter.

When the transfer script has finished its processing, it must do the following to pass the results of the transfer back to the application script.

1. Set the *from_info_type* variable to FROM_TRANSFER.
2. Populate the following fields in the from_info structure.

Field	Description
`xferresults`	XFER_SUCCESS if successful XFER_BUSY if busy XFER_NA if no answer XFER_FAILURE if failed
`xferhangup`	XFER_HANGUP if caller hung up during transfer, XFER_NOHANGUP if caller did not hang up during transfer

3. Set register 3 to the defined value EXECU_SWIN.

```
load (r.3, im.EXECU_SWIN)
```

4. Execute the application script.

```
execu (ch.ti__scriptname, 1)
```

As a guide to developing your own transfer script, the default script for the AT&T System 75 PBX is provided with the Toolkit. The source for the file can be found in the **s75xfer.t** file in the **/toolkit/src/scripts** directory. Two other files used to compile the script are included in this directory: **sw_xfer.c** contains variables used by the script and **sw_xfer.h** contains defined symbols. The file **Xfer_us.h** results from using *c2tas* as described in Chapter 4, *Development Structures*; **sw_xfer.c** is shown in the example as **vpc.c** and **Xfer_us.h** is shown as **vpc.h**.

DIGITAL TRANSFERS

A digital link is another method of indicating to the switch that a call is being transferred. Using a process, messages exchanged between the voice mail system and the switch are used to initiate transfers and to report the status of transfers.

Writing Your Own Transfer Process

If the switch supports a voice mail interface which can be accessed via a serial link and that voice mail interface supports call transfers, you will want to develop your own digital transfer software. How that software interfaces to the switch is your responsibility; you will need to understand the protocol provided by the switch. You are also responsible for determining the physical connection between the switch and the voice mail system.

Refer to the Chapter 4, *Development Structures* for guidelines on the organization of software. Also, read the *Using Switchhook Flash Transfer as Backup* section of this chapter for information on specifying your own digital transfer process.

Your transfer process will receive messages from **swindip** with mcont set to TX_REQ and with the structure tx_req. This structure contains three fields of information for your process (in addition to the standard header, including *mchan*).

Field	Description
`number`	phone number to be transferred to
`type`	transfer type: XFER_BLIND if blind XFER_SEMI_INT if semi-intelligent XFER_INTEL if intelligent
`num_rings`	number of rings to wait for intelligent transfer

Your process then replies to **swindip** with the same message, the same structure, and the following fields populated (in addition to the standard header, including *mchan*).

Field	Description
`xferresults`	results of transfer: XFER_SUCCESS if successful XFER_BUSY if busy XFER_NA if no answer XFER_FAILURE if transfer failed
`number`	phone number transferred to
`type`	transfer type: XFER_BLIND if blind XFER_SEMI_INT if semi-intelligent XFER_INTEL if intelligent
`num_rings`	number of rings to wait for intelligent transfer

Using Switchhook Flash Transfer as Backup

If your switch supports digital transfers *and* switchhook flash transfers, you may wish to specify the switchhook flash method as a backup when the digital link is down. You will need to do the following.

1. Review the *Switchhook Flash Transfer* section of this chapter to determine how to implement the switchhook flash transfer, including flash duration, call progress tone frequencies, triggers, timing, patterns, transfer sequences, and transfer scripts (if necessary).
2. Enter the following information on the *Params Worksheet* in Appendix A, *Development Aids*.
   - line 2: `digital_blind` *dip-id script-name*
   - line 3: `digital_semi` *dip-id script-name*
   - line 4: `digital_intel` *dip-id script-name*

   dip-id is the process identifier (DynaDIP name) associated with the process which receives digital transfer requests. *script-name* is either `default` if the default script is used, or it is the name of the script supplied by the switch package to perform switchhook transfers.

If switchhook flash is not a backup option for any of the transfer types, enter the following information on the *Params Worksheet* in Appendix A, *Development Aids*.

- line 2: `digital_only_blind` *dip-id*
- line 3: `digital_only_semi` *dip-id*
- line 4: `digital_only_intel` *dip-id*

dip-id is the process identifier (DynaDIP name) associated with the process which receives digital transfer requests.

NO TRANSFERS

If your switch does not support transfers, specify the following information on the *Params Worksheet* in Appendix A, *Development Aids*.

- line 2: `no_blind`
- line 3: `no_semi`
- line 4: `no_intel`

Under these circumstances, a caller attempting to transfer would hear something similar to the following: "Your call cannot be transferred."

8. Disconnect Options

This chapter consists of five sections, corresponding to the five options for disconnect. Not all of these sections are applicable for a particular switch package.

A Voice Power application needs to know when a caller disconnects from a call for many reasons; a few are listed here.

- stop recording (a message) after a caller hangs up
- free up the channel for other calls
- flag a subscriber as having logged out, so that subscriber can log in again later

SPECIFYING WINK PARAMETER

If the switch provides a wink disconnect signal (drops loop current for a specified amount of time when one party drops off the call), you must set a parameter, using the Toolkit, based on the length of the disconnect. The parameter specifies the minimum duration of the loss of loop current to signify a disconnect; any loss of loop current shorter than the minimum is ignored.

The allowed range of values for the minimum wink disconnect interval is between 300 and 800 milliseconds. Note that the default value is 300 milliseconds. The minimum wink disconnect is indicated in the **userTunable** file in the **/toolkit/src/files** directory. If you need to modify the minimum wink disconnect, make a copy of the file (keeping the name **userTunable**) and change the value on the third line.

NOTE

You may have already made a copy of the **userTunable** file in order to modify flash duration in Chapter 7, *Transfer Options*. If so, make the wink interval change in that same copy.

Also, indicate that you will be providing a **userTunable** file by entering a p on the seventh (userTunable flag) line of the *Registration File Worksheet* in Appendix A, *Development Aids*. (You may have already done this as part of Chapter 7, *Transfer Options*.)

DIAL TONE AS DISCONNECT

No matter what other types of disconnect signals are expected for a particular switch, the detection of dial tone is always handled as an indication of disconnect. The dial tone which signifies disconnect must be the switch dial tone. (See the the *Switchhook Flash Transfer* section of Chapter 7, *Transfer Options*). Other sources of dial tone, such as the Central Office, may not trigger the disconnect signal, if they do not match the switch dial tone in frequency.

The Voice Power application takes care of the few places where dial tone is not a disconnect signal. For example, in the Outcalling feature, the application originates the call by soft seizing the channel causing dial tone. Also, during call transfers, dial tone is just one of several call progress tones the application is looking for; in this case, it may or may not be a disconnect, depending on how transfers work on that switch.

SPECIFYING CALL PROGRESS TONES TO STOP CODING

A few switches provide one of the other standard call progress tones as an indication that a party has disconnected from a call. The Toolkit will allow you to specify that the ring tone, busy tone, and/or reorder tone indicate that the caller has hung up while recording a message. These call progress tones are not treated as a disconnect any other time.

The Toolkit provides you with the default **tsm.rc** file; if you need to modify it, make a copy of it first (keeping the name **tsm.rc**), then edit the copy. The **tsm.rc** has three lines, one for each call progress tone. Each line contains the symbol for the call progress tone, an equal sign (=), and `1` or `0`. `1` means that the call progress tone indicates a disconnect during coding; `0` means that the call progress tone does not indicate a disconnect during coding. For example, if reorder tone is the only tone to cause disconnect, the entries in **tsm.rc** would look like the following.

```
TR_D_RINGTONE=0
TR_D_BUSYTONE=0
TR_D_REORDERTONE=1
```

If you modify the **tsm.rc** file, write a `p` on the ninth (tsm.rc flag) line of the *Registration File Worksheet* in Appendix A, *Development Aids*.

DIGITAL DISCONNECT SIGNAL

A digital link is another method of indicating to the voice mail system that a caller has disconnected.

Writing Your Own Disconnect Process

If the switch supports a voice mail interface which can be accessed via a serial link, and that voice mail interface supports a message from the switch indicating a call disconnect, you will want to develop your own digital disconnect software. How that software interfaces to the switch is your responsibility; you will need to understand the protocol provided by the switch. You are also responsible for determining the physical connection between the switch and the voice mail system.

For information on software development, see Chapter 4, *Development Structures*.

Your process will use the SOFT_DISC mcont and the soft_disc structure when sending the digital disconnect message to **swindip**. Depending on the data from the switch, you will have to specify the channel number in the standard header (*mchan* in **mhdr**), or the channel phone number in the `chext` field of the soft_disc structure.

Use of the digital disconnect feature does not require registration in any file.

NO DISCONNECT

If the switch does not provide a wink disconnect signal, a call progress tone (dial tone, ring tone, busy, or reorder tone), or a digital disconnect message, the Voice Power application will have to rely on silence and time-outs for detecting disconnects. In this case, to avoid inadvertent disconnects, do the following.

- Set the wink disconnect interval to its maximum allowed value (800 milliseconds).
- Modify the **tsm.rc** file so that ring tone, busy tone, and reorder tone do not trigger a disconnect. (This is the default.)

See the *Specifying Wink Parameter* and *Specifying Call Progress Tone to Stop Coding* sections of this chapter for information on setting these parameters.

Voice Power applications will stop recording a message if five seconds of silence are detected. This silence is not a direct indication of the caller disconnecting; the application must assume a caller is still on the line and it, therefore, prompts the caller with several options. Only after several prompts (usually three) without a touch tone will the application disconnect.

For switches that do not provide any disconnect indication, voice mail subscribers should be trained to press the disconnect sequence (for example, AUDIX Voice Power uses **X) when finished with their call. This has two benefits.

- The channel is freed up to handle additional calls.
- The subscriber is logged out immediately, thereby allowing him or her to log in again without waiting for the prompting and time-outs to expire.

9. Miscellaneous Options

This chapter consists of four sections corresponding to miscellaneous features of Voice Power applications. Not all of these sections are applicable for a particular switch package.

- **leave-word-calling:** Many switches support a feature where caller A can notify caller B that B should call A. If this feature results in a message over the voice mail serial link or via in-band signaling, a Voice Power application can put a message in B's mailbox to call A.
- **in-service notification:** Some switches need to know if a voice mail system is ready to accept calls on each of its channels. If so, a message can be sent over the voice mail serial link or a script can be run to dial a feature access code to notify the switch of channel status.
- **message-waiting refresh:** A switch may request that the voice mail system send the message-waiting on/off status of all the extensions. The request may come from the voice mail serial link or via in-band signaling.
- **day/night service change:** Voice Power applications support day and night services for automated attendants. If the switch has a day/night service change feature, it can notify the application (via a message on the voice mail serial link or in-band signaling) when the service is to be changed.

LEAVE-WORD-CALLING FEATURE

The leave-word-calling feature can be supported by either a voice mail interface to the switch or by in-band signaling from the switch.

Voice Mail Serial Link Support

If your switch supports a serial link that provides the leave-word-calling feature, you will probably want to incorporate the handling of this feature with the processes that handle call information. Refer to Chapter 4, *Development Structures,* for details.

The message your process sends to **swindip** for leave-word-calling uses the mcont PUT_LWC and the structure put_lwc. The following two fields are important to leave-word-calling: `sendext`, the null-terminated extension of the sender, and `rcvext`, the null-terminated extension of the receiver of the message. As a result, a leave-word-calling message is placed in the receiver's mailbox; it informs the receiver that the sender requests a call back. If the receiving extension is not a subscriber on a Voice Power system, the message is ignored by **swindip**. Also, not all Voice Power applications support this feature; if they do not, the application will ignore the message.

In-band Signaling Support

If your switch provides an in-band signal which indicates a leave-word-calling message, your call information script must send a message to **swindip** using the `dbase` instruction. Use the defined value SWINDIP_NAME as the destination of the message. The mcont is PUT_LWC and the structure is put_lwc. The following two fields are important for leave-word-calling: `sendext`, the null-terminated extension of the sender, and `rcvext`, the null-terminated extension of the receiver of the message. Note that this message is not returned by **swindip**; set the response interval to 0 with the `nwitime` script instruction.

This message results in a leave-word-calling message being placed in the receiver's mailbox; it informs the receiver that the sender requests a call back. If the receiving extension is not a subscriber on a Voice Power system, the message is ignored by **swindip**. Also, not all Voice Power applications support this feature; if they do not, the application will ignore the message.

IN-SERVICE NOTIFICATION

The in-service notification feature can be supported by either a voice mail interface to the switch or by a script dialing a feature access code to the switch.

Voice Mail Serial Link Support

If your switch supports a voice mail interface which requires that a message be sent to the switch when a channel is placed in service, you must do the following.

1. Write a `p` on the eighth (adjunctUtil flag) line of the *Registration File Worksheet* in Appendix A, *Development Aids*.
2. Copy the executable `adjunctUtil` from **/toolkit/src/files** to your development area, for inclusion in your package.
3. Write the following data on the sixth (in service) line of the *Params Worksheet* in Appendix A, *Development Aids*.

   `digital_inserv` *dip-id*

   dip-id is the DynaDIP identifier for your process to receive the in-service notification message.
4. Your process must receive the message from **swindip**, with mcont CHAN_INSERV and with structure chan_inserv. There are no fields other than the standard header, including *mchan*, which specifies the in-service channel.

Script Support

If a script is required to dial a feature access code to notify the switch that a voice mail channel is in service, you must do the following.

1. Write a `p` on the eighth (adjunctUtil flag) line of the *Registration File Worksheet* in Appendix A, *Development Aids*.
2. Copy the executable `adjunctUtil` from **/toolkit/src/files** to your development area for inclusion in your package.
3. Enter the following data on the sixth (in service) line of the *Params Worksheet* in Appendix A, *Development Aids*.

   `analog_inserv` *scriptname*

   scriptname is the name of the script you write which dials the feature access code.
4. You must develop a script to perform the in-service notification to the switch. Refer to Chapter 4, *Development Structures*, for information on developing scripts.

The only information your script uses from the application script is its name, which is found in the `scriptname` field of the to_info structure. When the in-service notification script has finished its processing, it must do the following to return to the application script.

1. Set the *from_info_type* variable to FROM_SOFTCHECK.
2. Populate the following fields in the from_info structure.

Field	Value
`softtype`	SOFT_INSERV
`inserv_flag`	INSERV_SUCCESS if successful INSERV_FAILURE if failed

3. Set register 3 to the defined value EXECU_SWIN.

```
load (r.3, im.EXECU_SWIN)
```

4. Execute the application script.

```
execu (ch.ti__scriptname, 1)
```

MESSAGE-WAITING REFRESH

The message-waiting refresh feature can be supported by either a voice mail interface to the switch or by in-band signaling from the switch.

Voice Mail Serial Link Support

If your switch supports a serial link that provides the message-waiting refresh feature, you will probably want to incorporate the handling of this feature with the processes that handle call information. Refer to Chapter 4, *Development Structures,* for details.

The message your process sends to **swindip** for message-waiting refresh uses the mcont MASS_REFRESH and the structure mass_refresh. There are no fields associated with this message other than the standard header. This message causes **swindip** to refresh the message-waiting status of all the extensions known to the Voice Power application.

In-band Signaling Support

If your switch provides an in-band signal which indicates a request to refresh all message-waiting indicators, your call information script must do the following.

- Send a message to **swindip** using the `dbase` instruction.
- Use the defined value SWINDIP_NAME as the destination of the message.
- The mcont is MASS_REFRESH and the structure is mass_refresh. There are no fields associated with this message other than the standard header.
- This message is not returned by **swindip**; set the response interval to 0 with the `nwitime` script instruction.

DAY/NIGHT SERVICE CHANGE

The day/night service change feature can be supported by either a voice mail interface to the switch or by in-band signaling from the switch.

Voice Mail Serial Link Support

If your switch supports a serial link that provides the day/night service change feature, you will probably want to incorporate the handling of this feature with the processes that handle call information. Refer to Chapter 4, *Development Structures,* for details.

- The message your process sends to **swindip** for day/night service change uses the mcont DAY_NIGHT_SVC and the structure day_night_svc.
- Set the `param` field to DAY_SVC_PARAM if the service is changing to day or to NIGHT_SVC_PARAM if the service is changing to night.
- If the Voice Power application supports an automated attendant, this message is sent by **swindip** to the process which supports the automated attendant, resulting in the appropriate service being presented to callers.

In-Band Signaling Support

If your switch provides an in-band signal which indicates a change in day/night service, your call information script must do the following.

- Send a message to **swindip** using the `dbase` instruction.
- Use the defined value SWINDIP_NAME as the destination of the message.
- The mcont is DAY_NIGHT_SVC and the structure is day_night_svc. Set the `param` field to DAY_SVC_PARAM if the service is changing to day or to NIGHT_SVC_PARAM if the service is changing to night.
- Note that this message is not returned by **swindip**; set the response interval to 0 with the `nwitime` script instruction.

10. Installing Your Switch Integration Package

This chapter details how to put your switch integration package on diskette, install it on the run-time environment, and test it.

PUTTING THE PACKAGE TOGETHER

Before you put your switch package on a diskette, you may wish to review the *Checklist* in Appendix A, *Development Aids*, to verify that you have considered all components of a switch integration package. In addition, do the following.

- The Toolkit provides utilities to copy the files that make up your package onto a diskette. The tools are found in the **/toolkit/src/install** directory. Verify that you have the following files in your **install** directory.

Tool	Purpose
MKflop	calls MKfiles and MKSize, moves files to diskette
MKfiles	creates list of files that make up the package
MKSize	determines the space required for installing the package
Install	sample install script, which you will modify
Remove	sample remove script, which you will modify
Name	package name, which you will modify

MKflop, **MKfiles**, and **MKSize** can be used without modification.

Name must be modified to include the name of your package as displayed by the UNIX utility `installpkg`.

Install and **Remove** must be modified based on the name of your switch package and on the options your package requires. Refer to the comments provided within the **Install** and **Remove** files for information on how to modify them for your package.

NOTE

The administration name file, parameters file, and rules file explained in the following bullets *must* have the same filename, for example, **acme**. The registration file is not required to have the same name as the other three files, but it is recommended for consistency.

- All switch packages require a registration file, which contains flags indicating which other files (such as, **filters** and **triggers**) are installed as part of the package. Refer to the *Registration File Worksheet* in Appendix A, *Development Aids*, for the contents of this file. Each line on the worksheet represents a field in the file; the file consists of a single line, with each field separated by a blank. The name of the file is up to you; it should reflect the name of the switch. For example, the **acme** file looks like the following.

  ```
  "Acme Switch" acme N N p - p - p
  ```

  The directory under **install/pkg** where this file is placed is also up to you. AT&T suggests placing the file and all the files it references (**cad.pattern**, **cad.timing**, **filters**, **triggers**, **userTunable**, **adjunctUtil**, and/or **tsm.rc**) under the directory **install/pkg/swin/data**.

- All switch packages require an administration name (limit 55 characters) which is in a file in the **install/pkg/swin/data/swinnames** directory. This name is used by the `Install` shell and is usually derived from the name of the switch. For example, the **acme** administration name file contains the following.

  ```
  Acme Switch Integration Package
  ```

- All switch packages require a parameters file in the **install/pkg/swin/data/params** directory. This file specifies how call information, transfers, message-waiting, and other features are performed by this switch package. Refer to the *Params Worksheet* in Appendix A, *Development Aids*, for contents of this file. Each line on the worksheet represents a line in the file; therefore, the file consists of exactly seven lines. For example, the following is the **acme** parameters file.

  ```
  callinfo 15 3 2
  analog_blind default
  analog_semi default
  analog_intel default
  digital_mwl acmedip
  no_inserv
  nosync
  ```

- All switch packages require a rules file in the **install/pkg/swin/data/swinrules** directory. The first line of this file shows the maximum number of Voice Power applications that can use this switch package; it can either be a number, such as `1`, or the word `all`, indicating no limit. The rest of the file lists the names of the applications which can use the switch integration package. For example, the following is the **acme** rules file.

```
all
AUDIX Voice Power R3.0
AUDIX Voice Power Lodging R3.0
```

  The two application names shown in this example are the only two possibilities at this time. If your switch package has been designed for only one of the applications, list only that application name in the file. If an application package is not listed, it cannot use the switch package.

- If a switch package supports switchhook flash transfers (either directly or as a backup to digital transfers), the **active_xfer** file must be in the **install/pkg/swin/data/** directory.
- If a switch package supports analog message-waiting updates, the **active_mwl** file must be in the **install/pkg/swin/data/** directory.

NOTE

Put the **active_mwl** file in the **install/pkg/swin/data/** directory even if only to suppress the display of the message-waiting on/off codes on the MESSAGE WAITING PARAMETERS form.

USING INSTALLPKG

To prepare your switch package for installation on a customer machine, you must first put it on diskette(s), using the following steps.

1. Change directory to the **install** directory. This directory should contain the **/pkg** subdirectory and the following files: **Install**, **Remove**, **Name**, **MKflop**, **MKfiles**, and **MKSize** files.
2. Make the **MKflop**, **MKfiles**, and **MKSize** files executable, with the following command.

   ```
   chmod +x MK*
   ```

3. Execute **MKflop** using the following.

   ```
   ./MKflop
   ```

4. Follow the instructions presented on the screen. Your installable switch package may require more than one diskette. Simply insert the second diskette when prompted.
5. Remove the diskette when prompted.

   Your package is now ready for installation on a customer machine.

6. Enter **`displaypkg`** to verify that the customer machine has the following packages installed.

   — Integrated Voice Processing System Software Release 3.0

   — AUDIX Voice Power Application Software Release 3.0

   - and/or -

   AUDIX Voice Power Lodging Software Release 3.0

7. Enter **`installpkg`** to install your switch package. (You must be logged in as root.)
8. Follow the instructions presented on the screen.

   While your package is installing, be sure to watch for any error messages. As an indication that the installation is working properly, you should see the following messages.

   — `Switch package associated with AUDIX VOICE POWER 3.0` (or `LODGING`, or both)

   – *Switch package name* `has been successfully installed`

VERIFYING THE INSTALLATION

To verify that the switch package installation was successful, do the following.

1. Enter **/vs/bin/ivpss_menu**
2. Begin at the IVPSS R3.0 menu and select the following.

   ```
   Switch Interfaces
   Data Interfaces
   Application/Switch Interface Association
   ```

3. Verify that the name of your switch package is shown as the choice for each application package.
4. Press CANCEL (F6).
5. Select `Switch Interface Package Administration`.
6. Verify that your package administration is a choice.

   If your package does not supply administration, you will see a message stating this.
7. Press CANCEL (F6) twice.
8. Select `Analog Interfaces`.
9. Verify that your switch is shown as the title of the form and that the parameters shown make sense for your package.
10. Press CANCEL (F6) several times until you reach the `unix#` prompt.
11. If your switch package uses scripts, do the following.
    a. Change to the **/vs/trans** directory.
    b. Enter **ls -a**
    c. Verify that the listing shows all necessary script names.
12. If your switch package uses processes, do the following.
    a. Enter **/vs/bin/start_vs** to start the voice system.
    b. Enter **who -rp**
    c. Verify that the listing shows all necessary processes are running.

       If the processes are not running, make sure that the **/etc/conf/init.d** directory contains a file with all your processes.

ACCEPTANCE TESTING OF THE PACKAGE

In order to determine if your switch package is functioning properly with the Voice Power voice mail system, you should make several test calls. The types of calls you make depend on the capability of the switch and your switch package. For example, if your package does not support call information, then call information testing is not required.

Set up two test subscribers on the Voice Power application and on the switch. Create personal greetings and record several messages for each subscriber.

Depending on your switch and your package, the following test calls should be made.

1. Direct call to the voice mail system: system should allow you to retrieve messages.
2. Call to a subscriber with switch call coverage whose phone is busy: system should play prompt, allow you to leave a message for that subscriber.
3. Call to a subscriber with switch call coverage who does not answer: system should play prompt, allow you to leave a message for that subscriber.
4. Leave a message for a subscriber: subscriber's message-waiting indicator should go on.
5. Listen to and delete subscriber's messages: subscriber's message-waiting indicator should go off.
6. Hang up while leaving a message: message should terminate immediately, with possibly a short interval of a call progress tone or up to five seconds of silence.
7. Attempt various types of transfers, including blind (subscriber has switch call coverage), intelligent (subscriber does not have switch call coverage), and semi-intelligent (transfer to a number which is not a subscriber). For intelligent and semi-intelligent transfers, include cases where the called number is busy or rings without answer.

REMOVING THE PACKAGE

To remove your switch package from the run-time environment, do the following.

1. Log in as root.
2. Enter **`removepkg`**
3. The system displays a list of installed packages numbered 1 to *n*.
4. Enter the number of your switch integration package.

   The system displays the following message.

   `Do you really want to remove` *Switch Package Name*?

   `Strike ENTER when ready or ESC to stop.`
5. Press ENTER

   To verify that the package is being removed, use Ctrl+S (stop screen scroll) and Ctrl+Q (start screen scroll) to view the following status messages.

   `Removing system files for` *Switch Package Name*.

   `/vs/bin/util/remove switch` *Name of switch* `successfully uninstalled.`

   `The` *Switch Package Name* `has been successfully removed.`

   `The` *Switch Package Name* `is now removed.`

Watch for error messages which may indicate needed modifications in the `Remove` shell script of the package.

A. Development Aids

This appendix contains two worksheets and a checklist to aid you in developing a switch integration package. AT&T recommends that you make copies of the development aids in this appendix before beginning so that the masters are always available for the development of another switch integration package.

As you develop a switch integration package, you are directed to fill in the worksheets. When you have finished developing your switch integration package all blanks should have a value. If they do not, either choose the default or reread the section where the parameter is discussed. Use the index to locate items you are unfamiliar with.

REGISTRATION FILE WORKSHEET

All switch packages require a registration file which contains flags indicating other files (such as, **filters** and **triggers**) that are part of the package.

As you develop a switch integration package, you are directed to fill in the worksheet. You may be directed to the same item more than once because certain files contain several customizable components. If a blank has already been filled in, there is no need to write the letter or character again. Simply verify that it matches what you are currently being asked to write.

1. Package name (in double quotes)

   _____

2. Package id (filename, for internal use as directory name)

   _____

3. Dial tone training flag (`Y` for enabled, `N` for disabled): _____
4. Dial tone training changeable flag (`Y` for yes, `N` for no): _____
5. Cadence flag (`d` for default, `p` if **cad.pattern** and **cad.timing** provided): _____
6. Filters flag (`p` if **filters** and **triggers** provided, `-` (dash) if not): _____
7. userTunable flag (`d` for default, `p` if **userTunable** file provided): _____
8. adjunctUtil flag (`p` if **adjunctUtil** program provided, `-` (dash) if not): _____
9. tsm.rc flag (`d` for default, `p` if **tsm.rc** file provided): _____

Each line on the worksheet represents a field in the file; the file consists of a single line, with each field separated by a blank. The name of the file is up to you; it should reflect the name of the switch. For example, the **acme** file looks like the following.

```
"Acme Switch" acme N N p - p - p
```

The directory under **install/pkg** where this file is placed is also up to you. AT&T suggests placing the file and all the files it references (**cad.pattern**, **cad.timing**, **filters**, **triggers**, **userTunable**, **adjunctUtil**, and/or **tsm.rc**) under the directory **install/pkg/swin/data**.

For more information on use the registration file, see Chapter 10, *Installing Your Switch Integration Package*.

PARAMS WORKSHEET

All switch packages require a parameters file in the **install/pkg/swin/data/params** directory. This file specifies how call information, transfers, message-waiting, and other features are performed by this switch package. Each line on the worksheet represents a line in the file; therefore, the file consists of exactly seven lines. For example, the following is the **acme** parameters file.

```
callinfo 15 3 2
analog_blind default
analog_semi default
analog_intel default
digital_mwl acmedip
no_inserv
nosync
```

For more information on the file, see Chapter 10, *Installing Your Switch Integration Package*.

1. callinfo: _____

File Line	Description
`nonintegrated`	call information not supported
`callinfo` *aging times interval*	integrated *aging* is the # of seconds call data is valid *times* is the # of times to retry for data *interval* is the # of seconds between retries
`callscript` *script-name*	integrated with in-band signaling *script-name* is the script to collect touch tones

2. blind transfer: _____

File Line	Description
`no_blind`	blind transfers not supported
`analog_blind` *script-name*	switchhook flash transfers *script-name* is the name of script or `default`
`digital_blind` *dip-id script-name*	switchhook flash transfers *dip-id* is the DynaDIP identifier for transfer DIP *script-name* is the name of script or `default`
`digital_only_blind` *dip-id*	switchhook flash transfers *dip-id* is the DynaDIP identifier for transfer DIP

3. semi-intelligent transfer: _____

File Line	Description
`no_semi`	semi-intelligent transfers not supported
`analog_semi` *script-name*	switchhook flash transfers *script-name* is the name of script or `default`
`digital_semi` *dip-id script-name*	switchhook flash transfers *dip-id* is the DynaDIP identifier for transfer DIP *script-name* is the name of script or `default`
`digital_only_semi` *dip-id*	switchhook flash transfers *dip-id* is the DynaDIP identifier for transfer DIP

4. intelligent transfer: _____

File Line	Description
`no_intel`	intelligent transfers not supported
`analog_intel` *script-name*	switchhook flash transfers *script-name* is the name of script or `default`
`digital_intel` *dip-id script-name*	switchhook flash transfers *dip-id* is the DynaDIP identifier for transfer DIP *script-name* is the name of script or `default`
`digital_only_intel` *dip-id*	switchhook flash transfers *dip-id* is the DynaDIP identifier for transfer DIP

5. mwl: _____

File Line	Description
`no_mwl`	message-waiting not supported
`analog_mwl` *script-name*	analog message-waiting *script-name* is the name of script or `default`
`digital_mwl` *dip-id*	digital message-waiting *dip-id* is the DynaDIP identifier for message-waiting DIP

6. in service: _____

File Line	Description
`no_inserv`	inservice notification not required
`analog_inserv` *script-name*	analog inservice notification *script-name* is the name of script
`digital_inserv` *dip-id*	digital inservice notification *dip-id* is the DynaDIP identifier for inservice DIP

7. sync: _____

File Line	Description
`nosync`	synchronization message not required
`sync` *dip-id*	synchronization message required *dip-id* is the DynaDIP identifier for sync DIP

CHECKLIST

Before you put your switch package on a diskette, you may wish to review this checklist to verify that you have considered all components of a switch integration package. For more information on assembling the switch integration package, see Chapter 10, *Installing Your Switch Integration Package*.

Required Files

File	✔
Install	
Remove	
Name	
/swin/data/swinnames/*switch-name*	
/swin/data/params/*switch-name*	
/swin/data/swinrules/*switch-name*	
/swin/data/*switch-name* (registration file)	

Optional Files

File	✔
/swin/data/active_xfer	
/swin/data/active_mwl	
/swin/data/filters	
/swin/data/triggers	
/swin/data/cad.pattern	
/swin/data/cad.timing	
/swin/data/userTunable	
/swin/data/adjunctUtil	
/swin/data/tsm.rc	
/vs/trans/*call-info-script*.T	
/vs/trans/*blind-transfer-script*.T	
/vs/trans/*semi-intel-transfer-script*.T	
/vs/trans/*intel-transfer-script*.T	
/vs/trans/*message-waiting-script*.T	
/vs/trans/*inservice-notification-script*.T	
/etc/conf/init.d/*switch-name*	
/*switch-name*/bin/*switch-processes*	
/sid/bin/sidrdr	
/sid/bin/sidwtr	
/sid/bin/sid_admin	
/sid/txt/h_sid.txt	

Abbreviations

ALT	assembly load and test
AT&T	American Telegraph and Telephone
AUDIX	Audio Information Exchange
CDH	call data handling
CO	central office
COM2	serial communications port 2
COR	class of restriction
COS	class of service
DCE	data communications equipment
DCP	Digital Communications Protocol
DID	direct inward dialing
DIO	disk input/output
DIP	data interface process
DTE	data terminal equipment
EIA	Electronic Industries Association
ET	error tracker
FACE	framed access command environment
FMLI	form and menu language interpreter
FOOS	facility out of service
I/O	input/output
IRQ	interrupt request
IVP4	Integrated Voice Processing board (4 channels)
IVPSS	Integrated Voice Processing system software
K	kilobytes
LED	light emitting diode
LWC	leave word calling
MANOOS	manually out of service
Mbytes	megabytes
MTC	maintenance

MWL	message-waiting lamp
PBX	private branch exchange
PC	personal computer
PEC	price element code
POST	power-on self test
RAM	random access memory
ROM	read-only memory
SA	software associate
SID	switch integration device
SIMM	single in-line memory module
SMDI	simplified message desk interface
SS	software specialist
TRIP	tip/ring input process
TSC	technical support center
TSS	technical support services
TSM	transaction state machine
VDC600	video display card 600
VROP	voice response output process
WGS	work group systems

Glossary

blind transfer See **transfer, blind**.

call information The data passed from the switch to the voice mail system. It includes the following pieces of data: call type (direct, covered), calling party, called party, call origination (internal, external), and reason for coverage (busy/ring no answer).

call progress tones Tones which indicate call status, for example, dial tone, stutter dial tone, busy, ringing.

data interface process (DIP) A process with an associated message queue by which it receives messages from other processes. See also, **pure process**.

development environment The hardware and software used to develop a switch integration package.

dial tone training The ability of the switch to soft seize a channel and analyze the dial tone for two frequencies which are then used for dial tone detection.

DIP See **data interface process**.

dynamic dip (dynadip) The DynaDIP feature of CONVERSANT Intro allows you to use an alphanumeric name for the DIP, rather than a number.

intelligent transfer See **transfer, intelligent**.

process A development structure written in C and useful in interfacing to components other than the phone line, such as a voice mail serial port.

pure process A process that can send messages to DIPS. Pure processes do not have a message queue (do not receive messages), and they do not call `VSstartup`. See also, **data interface process**.

reader process A process that reads messages from the switch over the voice mail interface.

run-time environment The hardware and software that the switch integration package is run on, for example, a test machine or a customer machine.

script A development structure that consists of a series of operations that interface to the phone line; they are written in a special programming language called *native script*.

semi-intelligent transfer See **transfer, semi-intelligent**.

SID See **switch integration device**.

simplified message desk interface A standard protocol for the exchange of information, for example, call information transmitted between the switch and an application.

source directory Contains source code, original copies of files, and other files which are not actual parts of the resulting switch package (for example, `src`).

SMDI	See **simplified message desk interface**.
swindip	The switch package data interface process.
switch integration device (SID)	A link between a switch and a voice mail system, which provides a standard interface for call information and message-waiting updates.
target directory	Includes only those files that actually install on the customer machine (for example, `install`).
transfer, blind	A transfer in which the voice mail system does not wait for call progress tones from the switch to indicate the status of the call.
transfer, intelligent	A transfer in which the voice mail system stays on the line until someone answers the phone or until a specified number of rings (administerable) is reached. If the limit of rings is reached, the call is returned to the calling party and the voice mail system informs him or her that the called party is not available. If busy is detected, the call is returned to the calling party and the voice mail system informs the caller that the number is busy.
transfer, semi-intelligent	A transfer in which the voice mail system stays on the line to determine the status of the call. If busy is detected, the call is returned to the calling party and the voice mail system informs the caller that the number is busy. If ringing is detected, the voice mail system drops off the call; this allows the caller to hear the ringing. If voice energy is detected (someone answers the phone), the voice mail system informs that person that a call is being transferred to him or her, then drops off the call.
Voice Power application	An IVPSS R3.0-based voice mail application for which the switch integration package provides a PBX interface.
writer process	A process that writes messages to the switch.

Index

W