Runtime Guide

IDL Version 5.2
November, 1998 Edition

Restricted Rights Notice

The IDL® software program and the accompanying procedures, functions, and documentation described herein are sold under license agreement. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement.

Limitation of Warranty

Research Systems, Inc. makes no warranties, either express or implied, as to any matter not expressly set forth in the license agreement, including without limitation the condition of the software, merchantability, or fitness for any particular purpose.

Research Systems, Inc. shall not be liable for any direct, consequential, or other damages suffered by the Licensee or any others resulting from use of the IDL software package or its documentation.

Most Current Documentation

Because changes may be made to IDL after documentation has gone to press, please consult IDL's hypertext online help system for the most current version of this document.

Permission to Reproduce this Manual

Purchasers of IDL licenses are given limited permission to reproduce this manual provided such copies are for their use only and are not sold or distributed to third parties. All such copies must contain the title page and this notice page in their entirety.

Acknowledgments

IDL documentation is printed on recycled paper. Our paper has a minimum 20% post-consumer waste content and meets all EPA guidelines.

Contents

Chapter 1:
Building IDL Runtime Applications 1
What is IDL Runtime? 1
When is it Appropriate to use IDL Runtime? 1
Licensing Runtime Applications 2
Creating Runtime Applications 3
Using IDL Insight or LIVE_* in Runtime Applications 5
Using LINKIMAGE with Runtime Applications 5
Using Callable IDL with Runtime Applications 5
Online Help for Runtime Applications 6
Packaging Runtime Applications 8

Chapter 2:
A Simple Runtime Application 9
The rt_app Application 9
Creating the rt_var.sav File 10
Creating the rt_app.pro File 10
Creating the rt_app.sav File 13
Running the rt_app Application 14

Chapter 3:
IDL Runtime for Windows 15
Runtime Executable Files on the IDL CD-ROM 16
Building a Distribution Tree for your Application 16
Licensing Issues 18
The IDL.INI File 19
Using Runtime Applications 20

Chapter 4:
IDL Runtime for Macintosh 23
Creating a Runtime Executable 24
Creating a Runtime Save File 25
Building a Distribution Tree for your Application 26
Licensing Issues 27
Using Runtime Applications 28

Chapter 5:
IDL Runtime for Unix and VMS 29
Building a Distribution Tree for your Application 30
Licensing Issues 31
Using Runtime Applications 31

Appendix A:
Runtime Applications and the IDL DataMiner A-1
Building a Windows Application that uses the DataMiner A-2
Building a Macintosh Application that uses the DataMiner A-4
Building a Unix Application that uses the DataMiner A-5

Chapter 1

Building IDL Runtime Applications

What is IDL Runtime?

Runtime IDL runs programs written in the IDL language, but does not provide access to the IDL command line or command window. It does not accept IDL commands interactively. You can create IDL programs and bundle them with runtime IDL for distribution to users who do not have access to the full version of IDL.

You distribute your runtime IDL application in the form of an IDL save/restore file. When your user starts the program, runtime IDL restores the `.SAV` file and executes the routines contained therein.

When is it Appropriate to use IDL Runtime?

Runtime IDL is best suited to applications where the user does not need access to the full scope of IDL's features. Runtime IDL is appropriate for:

- Vertical-market packages developed in IDL but which appear to the user as stand-alone applications.
- Software designed for use by operators or technicians who do not need IDL's full range of analytical tools.
- Situations in which the developer does not want end-users to be able to modify functions written in the IDL language.
- Organizations with existing investments in IDL code, where some mixture of runtime and standard IDL licenses may be cost effective.

There are drawbacks, however, to distributing applications in runtime IDL packages. If users need advanced analytical tools outside the scope of your application, they may prefer to use standard IDL. As a developer of runtime IDL applications you are responsible for licensing each copy, and for creating installation scripts or programs for your application. Applications written in IDL and designed to be run with standard IDL do not share these drawbacks.

Licensing Runtime Applications

Copies of runtime IDL are protected either by a software license manager or by a combination of software and hardware license keys. As a runtime IDL application developer, you are responsible for providing licenses to your end-users.

Note There are different hardware license keys for the runtime and developer versions of IDL. *On the Macintosh, both hardware keys cannot be installed on your machine at the same time.* This means that you will need to use your developer version hardware key while developing your runtime application, then remove the key and switch to a runtime hardware key to test your application. On Windows machines, you can install both hardware keys simultaneously.

Research Systems strongly recommends that you distribute your runtime IDL applications pre-licensed. This means that you will have to obtain the proper licensing information from Research Systems and individually license each copy of your application before delivering it to the end-user. You should have received IDL installation documentation for all the platforms on which your application will run—these *Installation Guides* contain the information you will need in order to license your application. Specific notes on licensing your application under Unix/VMS, Microsoft Windows, and the Macintosh OS are included in the following chapters.

Creating Runtime Applications

Use a fully functional copy of IDL to develop your runtime application. All of IDL's functions and procedures—along with functions and procedures you may have written in the IDL language—are available. Note that your application must be started by calling an IDL procedure named `MAIN`. The `MAIN` procedure must be included in the `.SAV` file restored when runtime IDL first starts.

Once you have created and tested your application, use the IDL RESOLVE_ALL procedure to ensure that all routines called by your application are compiled, then use the IDL SAVE procedure with the ROUTINES keyword to create an IDL save/restore file containing the `MAIN` procedure and any other routines used by the program.

Note that runtime IDL does not compile `.PRO` files. Because of this, you will need to create `.SAV` files for any procedures or functions used by your application that are not included in the main `.SAV` file.

Using the SAVE Procedure

The IDL SAVE procedure creates binary files containing compiled IDL code. Files created with the SAVE procedure should use the `.SAV` file extension. When a `.SAV` file is *restored* inside an IDL session (by specifying the name of the `.SAV` file when starting runtime IDL or by using the IDL RESTORE procedure), the IDL code modules included in the file become available for use.

To use the SAVE procedure to create a `.SAV` file containing your IDL program code, do the following:

1. Exit and restart IDL. Because you will be saving all of the IDL routines compiled in an IDL session, exiting IDL and restarting gives you a clean slate from which to work.

2. Compile your application. Use the `.RUN` or `.COMPILE` executive commands to compile your program code. For example, if your program is named `myprogram`, use the command

   ```
   IDL> .COMPILE myprogram
   ```

   Note that you must create an IDL procedure named MAIN that starts your program; runtime IDL will call the MAIN procedure. When the MAIN procedure exits, so does runtime IDL.

3. Use the RESOLVE_ALL procedure to ensure that all of the IDL code used by your program is compiled. The RESOLVE_ALL procedure iteratively resolves (by compiling) any uncompiled user-written or library procedures or functions that are called in any already-compiled procedure or function. After calling

RESOLVE_ALL, every routine used by your program should be compiled in the current IDL session.

```
IDL> RESOLVE_ALL
```

4. Use the SAVE procedure to create a .SAV file containing your program code. Note that the SAVE procedure cannot create a single .SAV file that contains both program code and saved IDL variables; use the ROUTINES keyword to save all of the compiled routines in your save file. For example, if your program is named `myprogram`, use the command

```
IDL> SAVE, /ROUTINES, FILENAME='myprogram.sav'
```

Secondary .SAV Files

If your application requires that you provide data in IDL variables, you will need to save the IDL variables in a separate .SAV file which is restored by a routine in your application. (Generally, the MAIN program calls the IDL RESTORE procedure to restore all secondary .SAV files.)

Keywords to the SAVE procedure allow you to save IDL variables, system variables, and common block definitions. For example, to save the definitions of all of the variables and common block definitions used in your compiled program in a file named `myvariables.sav`, use the command

```
IDL> SAVE, /VARIABLES, /COMM, FILENAME='myvariables.sav'
```

To restore the values of the variables saved in `myvariables.sav` from within your runtime application, include the line

```
RESTORE, 'myvariables.sav'
```

in your MAIN program.

If you prefer not to include all of the IDL routines required by your application, you can save them in one or more separate .SAV files and use the RESTORE procedure in your MAIN program to restore the routines:

```
RESTORE, 'extra_routines.sav'
```

Note If you choose to create secondary .SAV files containing routines, you will need to exit IDL and restart before calling the SAVE procedure to create the secondary file.

Further Information

See the sections on ".COMPILE", "RESOLVE_ALL", "RESTORE", and "SAVE" in the IDL *Reference Guide* for details on using these routines.

Using IDL Insight or LIVE_* in Runtime Applications

If your runtime application uses functionality from IDL Insight, you must include the file `insight.sav` in your runtime application's distribution. The `insight.sav` file is located in the `hook` subdirectory of the `lib` directory of the IDL distribution.

Similarly, if your runtime application uses any of the LIVE_* IDL routines, you will need to include the `insight.sav` file in your distribution. The LIVE_* tools are built using Insight functionality, and restore portions of the `insight.sav` when they run.

Using LINKIMAGE with Runtime Applications

If your runtime IDL application relies on code written in languages other than IDL and linked into IDL using the LINKIMAGE procedure, you must make sure that the routines declared with LINKIMAGE are linked into IDL *before* they are called. In practice, the best way to do this is to make the calls to LINKIMAGE in your MAIN program, and include the code that uses the linked routines in a secondary `.SAV` file. In this case your MAIN program may look something like this:

```
PRO main
LINKIMAGE, 'link_function', 'new.dll'
                                        Link the external code.
RESTORE, 'secondary.sav'                Restore code that uses linked code.
myapp                                   Run your application.
END
```

In this scenario, the IDL code that calls the LINK_FUNCTION routine (the routine linked into IDL in the LINKIMAGE call) is contained in the secondary `.SAV` file `'secondary.sav'`.

Note When creating your secondary `.SAV` file, you will need to issue the LINKIMAGE command before calling the SAVE procedure to link your routine into IDL after you have exited and restarted. The RESOLVE_ALL routine does not resolve routines linked to IDL with the LINKIMAGE procedure.

Using Callable IDL with Runtime Applications

If you use IDL as a called library in your runtime IDL application, you should be aware of the following limitations. See the *Advanced Development Guide* for details.

1. You must use the `IDL_RuntimeExec()` function to execute your runtime application after initializing with `IDL_Init()`.

2. Under Unix and VMS, use the flag `IDL_INIT_RUNTIME` or `IDL_INIT_EMBEDDED` in the call to `IDL_Init()` to check out a runtime IDL license rather than a regular IDL license. Under Windows and on the Macintosh, runtime licensing is handled outside the Callable IDL application.

Online Help for Runtime Applications

The online help system used by IDL emulates (or uses, in the case of IDL for Windows) the Microsoft Windows Help viewer on all supported platforms. If you wish to supply online help along with your runtime IDL application, you have a number of options, which vary in complexity, cost, and level of integration with IDL's hypertext online help viewer.

Note Research Systems has no connection with the companies that produce these applications, and does not make any claims as to the applications' suitability.

Provide a Hypertext Help Viewer

As mentioned above, the online help system used by IDL emulates (or uses, in the case of IDL for Windows) the Microsoft Windows Help viewer on all supported platforms. The help viewer is part of the Microsoft Windows system software; if your application runs on other platforms and you wish to provide a help system similar to IDL's you will need to purchase help viewer software. The difficulty and expense involved in creating such files depends largely on the platform(s) involved.

Microsoft Windows

For Microsoft Windows systems, help files are relatively easy to create. Files must be created in the Rich Text Format (RTF) and compiled with Microsoft's help compiler. The help compiler is part of the Windows Software Developer's Kit, and is now included in several Microsoft programming products, including the Visual C++ development environment. The help compiler may also be available from the Microsoft ftp site (`ftp.microsoft.com`) or other Microsoft online software libraries, at little or no cost.

The Windows help system is often referred to as "WinHelp." The two components are the viewer (`WINHLP32.EXE`, found in the main `WINDOWS` directory of all Windows systems), and the help compiler. There are a number of third-party "help authoring systems" that simplify the creation of WinHelp compatible RTF files. Also, a number of third party books describe the WinHelp creation process—*Developing Online Help for Windows95*, by Scott Boggan, David Farkas, and Joe

Welinske (International Thompson Computer Press, 1996, ISBN: 185032211-2), is one that we have found useful.

Macintosh

For Macintosh, we at Research Systems use QuickHelp, a WinHelp-compatible compiler and viewer licensed from Altura Software, Inc. The QuickHelp compiler uses the same RTF files as are used by the Microsoft Help compiler. Altura Software can be contacted at the following address:

Altura Software, Inc.
510 Lighthouse Avenue, Suite 5
Pacific Grove, CA 93950
Phone: 408-655-8005
Fax: 408-655-9663
E-mail: info@altura.com
http://www.altura.com

Unix and VMS

For Unix and VMS, we use the HyperHelp compiler and viewer from Bristol Technology, Inc. Bristol makes a number of compilers that can compile source files in RTF, FrameMaker's MIF (Maker Interchange Format), SGML (Standard Generalized Markup Language), and their own simple HyperHelp Text (HHT) format. We use the MIF compiler to create hypertext help files from the same FrameMaker files that produce our hardcopy manuals.

Bristol also makes a product called Bridge that takes compiled HyperHelp files and converts them to RTF files that can be compiled with the Windows and Macintosh help compilers described above. In this way, we can create help files for all supported IDL platforms from a single source.

Bristol Technology can be contacted at the following address:

Bristol Technology, Inc.
241 Ethan Allen Highway
Ridgefield, CT 06877
Phone: 203-438-6969
Fax: 203-438-5013
E-mail: info@bristol.com
http://www.bristol.com

Accessing Hypertext Help Files

If you are providing hypertext help viewers, you can use the ONLINE_HELP procedure in your runtime IDL programs to display help files and control the viewer. See "ONLINE_HELP" in the IDL *Reference Guide*.

Create HTML Files for Use with a World Wide Web Browser

If you want to provide hypertext online help for your application, but do not want to use a dedicated (and potentially expensive) help viewer, consider using HTML (HyperText Markup Language) files in conjunction with a WWW (World-Wide Web) browser such as Netscape, Mosaic, or Microsoft's Internet Explorer. WWW browsers are available for all platforms supported by IDL. Note, however, that this option requires that your customers must procure a web browser in order to view your help system.

IDL provides a utility procedure that will convert properly-formatted text into HTML documents. See "MK_HTML_HELP" in the IDL *Reference Guide* for details.

Create Text Files for Display by IDL Routines

If your online documentation needs are minimal, or if you don't want to create hypertext online help files, consider using simple text files and displaying them using IDL's XDISPLAYFILE procedure. Using text files and XDISPLAYFILE is simple and free. See "XDISPLAYFILE" in the IDL *Reference Guide* for details.

Packaging Runtime Applications

Procedures for packaging your runtime IDL application depend on the operating system used by the target hardware platform. Specific notes on creating and packaging applications for Unix/VMS, Microsoft Windows, and the Macintosh OS are included in the following chapters.

Chapter 2

A Simple Runtime Application

In this chapter, we will create a very simple example runtime application. This example is not designed to touch on every situation you may encounter when developing runtime applications, but rather to give you a feel for how runtime applications are developed using IDL.

The rt_app Application

The example application we will create is named '`rt_app`'. It does the following:

1. Displays a surface plot of a data variable provided by the application developer (you), using a modified version of the IDL SURFACE procedure.
2. Allows the user to choose a JPEG image file for display.

Because `rt_app` consists of both IDL routines and IDL variable data, the "package" we will create includes two IDL SAVE files: `rt_var.sav` and `rt_app.sav`. The `rt_var.sav` file will contain the data variable displayed by

the modified SURFACE procedure. The `rt_app.sav` file will contain the IDL language code that defines the modified SURFACE procedure as well as the MAIN procedure.

Note This example application uses files in the full IDL distribution. As a result, it will not work properly with a runtime IDL distribution tree that does not include the `examples` subdirectory.

Creating the rt_var.sav File

The first thing we will do is create the `rt_var.sav` file. To create the `.SAV` file, do the following:

1. Start your full version of IDL. You must use a fully-licensed version of IDL to create your application; IDL with a runtime license does not give you access to the IDL command prompt.
2. At the IDL command prompt, enter the following lines:

```
peak = SHIFT(DIST(40), 25, 15)
peak = EXP(-(peak/15)^2)
SAVE, peak, FILENAME='rt_var.sav'
```

3. Exit IDL.

You should find the file `rt_var.sav` in the directory from which you started IDL.

Creating the rt_app.pro File

Next, we will create an IDL `.PRO` file containing the program code used in our application. To create the `.PRO` file, start your full version of IDL. Open a new file in IDL's built-in editor and save the file as 'rt_app.pro'. Add the lines shown below (don't include the numbers; they are included here for our convenience in explaining what the program does):

```
1  PRO mysurf, data
2  WINDOW, /FREE, XSIZE=350, YSIZE=300, RETAIN=2, $
3      TITLE='My Surface Window'
4  SURFACE, data, CHARSIZE=2.0, XMARGIN=[4,2], YMARGIN=[1,1]
5  END
6
7  PRO main
```

```
1  RESTORE, 'rt_var.sav'
2  mysurf, peak
3  IF (STRUPCASE(DIALOG_MESSAGE('Destroy Window', /INFO)) EQ 'OK') $
4     THEN WDELETE
5  image_path = FILEPATH('data', SUBDIR=['examples'])
6  image_file = DIALOG_PICKFILE(FILTER='*.jpg', PATH=image_path)
7  IF (image_file NE '') THEN BEGIN
8    READ_JPEG, image_file, image, ctable, COLORS=!D.TABLE_SIZE-1
9    image_size = SIZE(image)
10   WINDOW, /FREE, XSIZE=image_size[1], YSIZE=image_size[2], RETAIN=2
11   TVLCT, ctable
12   TV, image
13   IF (STRUPCASE(DIALOG_MESSAGE('Destroy Window', /INFO)) EQ 'OK') $
14     THEN WDELETE
15   ENDIF
16 END
```

Each line of this file is discussed below:

1

Define the procedure `mysurf`, which accepts a single argument named `data`.

2-3

Create a window to contain the surface plot.

4

Draw a surface plot of `data` in the window, using the character size and margins specified.

5

End the procedure.

7

Create the procedure `main`. Note that `main` does not take any arguments.

8

Restore the file `rt_var.sav`. This creates the variable `peak` in the current IDL context.

9

Call the `mysurf` procedure on the variable `peak`.

10-11

Display a dialog that closes the plot window when the user click's "OK".

12

Create a variable that contains the path to the `data` subdirectory of the `examples` directory of the IDL distribution.

13

Display a dialog that allows the user to choose a JPEG image file from the `data` directory.

14

Test to be sure the user chose a file. If the user clicked the "Cancel" button, the variable `image_file` will contain an empty string.

15

Read the JPEG file into the variable `image`, saving the color table and adjusting it to the number of colors available.

16

Use the SIZE command to determine the dimensions of the image.

17

Create a window the same size as the image.

18

Load the image's color table.

19

Display the image.

20-21

Display a dialog that closes the image window when the user click's "OK".

22

End of the IF statement

23

End of the `main` program.

After you have entered the code, save the rt_app.pro file. Select “Compile” from the Run menu to compile the code, then enter `main` at the IDL command prompt to run the code. You should see the surface plot, a dialog to remove the plot, a dialog allowing you to choose an image file (try `rose.jpg`), the image, and a dialog to remove the image.

Creating the rt_app.sav File

To create the `.SAV` file of the `rt_app` application, first exit and restart IDL. This clears IDL's memory of existing program units and allows you to save only the procedures you want in the final application save file. Next, do the following:

1. Start your full version of IDL.
2. At the IDL command prompt, enter:

   ```
   .COMPILE rt_app.pro
   RESOLVE_ALL
   ```

   You should see the following in the IDL command log window:

   ```
   IDL> .compile rt_app.pro
   % Compiled module: MYSURF.
   % Compiled module: MAIN.
   IDL> RESOLVE_ALL
   % Compiled module: RESOLVE_ALL.
   Resolving functions: FILEPATH
   % Compiled module: FILEPATH.
   IDL>
   ```

   This output shows that you have compiled the IDL code for the modules `mysurf`, `main`, `FILEPATH`, and `RESOLVE_ALL`.
3. Enter the following command at the IDL command prompt:

   ```
   SAVE, /ROUTINES, FILENAME='rt_app.sav'
   ```

   Note that all four compiled routines (including RESOLVE_ALL) are included in your `.SAV` file. If you do not wish to include the compiled version of RESOLVE_ALL in your `.SAV` file, either compile all of the routines your application uses individually (without using RESOLVE_ALL), or specify the names of the routines to include in your `.SAV` file when you call the SAVE

procedure. Using RESOLVE_ALL and saving all of the compiled routines ensures that your `.SAV` file contains all the code your application needs.

4. Exit IDL.

Running the rt_app Application

Now we are ready to run `rt_app` as a runtime application. The procedure for executing a runtime application differs from platform to platform; see the sections titled “Using Runtime Applications” in the following chapters for platform-specific details.

For example, suppose we have written the `rt_app` application for a Unix platform. To run `rt_app` in runtime mode, we would place the two files we have created—`rt_var.sav` and `rt_app.sav`—in the same directory, and in that directory enter the following at the Unix shell prompt:

```
idl -rt=rt_app.sav
```

When we issue this command, IDL checks out a runtime license and restores the file `rt_app.sav`. After restoring the file, IDL attempts to execute the `main` procedure. In the case of `rt_app`, the `main` procedure restores the `rt_var.sav` file, then calls the `mysurf` procedure on the restored variable. The `main` procedure then displays dialogs and an image file chosen by the user.

Your application will no doubt be more complicated than `rt_app`, but the procedures for creating save files and running the application will remain the same. The following chapters explain which parts of the IDL distribution you must include with your application, and discuss platform-specific details.

Chapter 3

IDL Runtime for Windows

Under Microsoft Windows 95, and Windows NT, runtime IDL uses a special executable file named `idlrt.exe`. This file is included on your IDL CD-ROM. Runtime IDL differs from standard IDL in the following ways:

- It will not accept commands interactively.
- It will not compile `.PRO` files. Note that this means that runtime IDL will not compile `.PRO` files found in the directories specified by the !PATH system variable in order to find unknown routines at runtime.
  Runtime IDL *will* restore `.SAV` files found in the directories specified by !PATH.
- Instead of presenting the user with an interactive prompt, runtime IDL starts execution by calling a procedure called `MAIN`, which is assumed to be in the `.SAV` file specified at startup. If no `.SAV` file is specified, runtime IDL looks for a file named `runtime.sav` in the current directory. When `MAIN` returns, runtime IDL ends the session.

Runtime Executable Files on the IDL CD-ROM

When you build a distribution tree for your runtime application, you will need to include the following special files from the IDL CD-ROM.

`idlrt.exe`	*The runtime IDL executable.*
`hinstall.exe`	*An installer for the Windows NT HASP service.*

The files are included in the unpacked IDL distribution on the CD-ROM disk. There are different versions of the files for different windows platforms—you will need to include the correct versions for the Windows platforms you support. Use of these files is discussed later in this section. The following sections describe where to find the files. In each case, <*cd-rom*> is the name of your CD-ROM drive.

Windows 95

The files are included in:

<*cd-rom*>`\idl51\bin\win95`

Note that you do not need `hinstall.exe` for Windows 95 installations.

Windows NT (Intel Processors)

The files are included in:

<*cd-rom*>`\idl51\bin\winnt`

Windows NT (DEC Alpha Processors)

The files are included in:

<*cd-rom*>`\idl51\bin\alpha`

Building a Distribution Tree for your Application

You will need to build a special copy of the full IDL distribution for each of the windows platforms your application runs on, containing support for only your application. Do this by removing unused files from the full distribution and by adding files used by your application. The easiest way to create a distribution tree is to begin with a *copy* of the standard IDL distribution (the `idl50` directory, usually located in `\rsi` on the disk drive that contains the `WINDOWS` directory) and do the following:

1. Remove all `.PRO` files from the `lib` directory and its subdirectories. `.PRO` files are ignored in runtime mode.
2. If your application uses Insight functionality or any of the LIVE_* routines, include the file `insight.sav` from the `hook` subdirectory of the `lib` directory. Otherwise, remove `insight.sav` and `demo.sav` from the `hook` directory.

3. If your application does not use Insight, remove the `examples` directory. If your application does use Insight, delete everything from the `examples` directory except for the `insight` subdirectory. (If your application uses the LIVE_* routines but not Insight itself, you can remove the entire `examples` directory.)

4. Remove the `external` directory.

5. Remove all `*.hlp`, `*.cnt`, `*.fts`, and `*.gid` files from the `help` directory. If your application does not use any of the widget applications for which help text files are included in the `widgets` subdirectory of the `help` directory, remove the `widgets` subdirectory. Remove the `copyrights` and `motd` subdirectories.

6. If your application does not produce PostScript output, remove the `ps` subdirectory of the `fonts` subdirectory of the `resource` directory. The file `hersh1.chr` in the `fonts` subdirectory is required if you use IDL's default fonts (in a plotting window, for example).

7. If your application does not use any of the map data supplied with IDL, remove the `maps` subdirectory of the `resource` directory.

8. If your application does not use color tables, remove the `colors` subdirectory of the `resource` directory.

9. Remove any `*.txt` files located anywhere in the remaining directories.

10. In the `idl50` directory, remove the file `idlde.exe` and insert `idlrt.exe`. (See "Runtime Executable Files on the IDL CD-ROM" on page 16 for the location of `idlrt.exe` for the platforms you support.) Remove all but the following files from the `idl50` directory:

    Windows 95
    `idl.ini, idl32.dll, idlrt.exe, ug3220.dll, wingl32.dll, wmesa32.dll`

    Windows NT (Intel Processors)
    `idl.ini, idl32.dll, idlrt.exe, ug3220.dll, wingl32.dll, wmesa32.dll`

    Windows NT (DEC Alpha Processors)
    `idl.ini, idl32.dll, idlrt.exe, ug3220.dll, wingl32.dll`

11. Alter the `idl.ini` file to contain the appropriate information for your application. See "The IDL.INI File" on page 19 for details on how to construct the proper initialization file.

12. Create `.SAV` files for any `.PRO` files from the IDL distribution that are used by your application, and place them in the `lib` directory. Alternatively, these routines can be compiled and saved in your application's main `.SAV` file.

13. Place your application's main `.SAV` file, along with any `.SAV` files containing variable data or additional routines, in the `hook` subdirectory of the `lib` directory in the distribution tree.

14. Your installation process must install the following files in the `SYSTEM` subdirectory of the `WINDOWS` directory (on Windows NT systems, the files should be installed in the `SYSTEM32` directory):

    Windows NT
    `ctl3d32.dll, odbc32.dll`

    Windows 95
    `ctl3d32.dll, glu32.dll, odbc32.dll, opengl32.dll`

    You will find these files in the `SYSTEM` and `SYSTEM32` directories of your development systems.

15. If your application uses IDL DataMiner functionality, see "Building a Windows Application that uses the DataMiner" on page A-2 for additional details on building your runtime distribution.

16. If you wish, create an installation script for your application and the runtime IDL distribution tree. Note that Windows NT requires that special software beyond the runtime application and IDL distribution tree be installed. See "IDL Runtime for Windows NT Requires the HASP NT Service" on page 20.

    There are a number of commercial applications available to help you build installers: these include EZ-Install for Windows, INSTALL Professional, and PC-Install for Windows. Research Systems has no connection with the companies that produce these applications, and does not make any claims as to the applications' suitability.

Licensing Issues

Runtime IDL uses the a combination of hardware copy protection devices and software license keys, as does standard IDL. (There are different hardware license keys for the runtime and developer versions of IDL.) This means that if you are distributing an application based on runtime IDL, you must also distribute both the hardware and software keys.

Normally, the first time the runtime IDL executable is run, the user is prompted to enter license information. Research Systems strongly recommends that you supply your runtime application "pre-licensed"—with the software key information already included in the `idl.ini` file you supply to your customer. Providing the license information in the `idl.ini` file means that you will not need to use the usual IDL licensing mechanism at all.

Note that even if you choose to have your customer license IDL the first time your application is run, you *must* supply the `idl.ini` file.

The IDL.INI File

Runtime IDL uses an initialization file—`idl.ini`—to store software licensing information and program defaults. As a runtime IDL developer, you will need to supply an `idl.ini` file that contains the proper information. The `idl.ini` file is read when IDL starts up for the first time and its entries are converted into Windows Registry entries automatically. (If registry entries already exist for an IDL installation in the same directory, a dialog will appear asking the user whether the existing entries should be overwritten.)

The following is an example `idl.ini` file, configured for runtime IDL:

```
[IDL 5.2]
HomeDir=.
SearchPath=+.\lib
HelpPath=+.\help
RuntimeFile=.\lib\hook\myapp.sav
RuntimeIcon=myicon.ico
InstallNum=314159
SiteNotice=Groovin' with IDL
Cookie=25-XALP9CAF
```

The fields have the following meanings:

[IDL 5.2]

The field heading specifies which version of IDL is in use. This heading should be the same as the heading used by the copy of IDL you use to create your runtime IDL application.

HomeDir

This field specifies the directory that contains the runtime IDL executable.

SearchPath

This field specifies which directories runtime IDL will search for `.SAV` files. The "+" symbol at the beginning of the string indicates that all subdirectories of the specified directory should be searched.

HelpPath

This field specifies which directories runtime IDL will search for hypertext help files. The "+" symbol at the beginning of the string indicates that all subdirectories of the specified directory should be searched.

RuntimeFile

This field should contain the name of the `.SAV` file to be restored automatically when runtime IDL starts. If this field is left blank and no `.SAV` file is specified on the command line, runtime IDL will attempt to restore a file named `runtime.sav` in the directory specified by the `HomeDir` field.

Note If you specify a `.SAV` file to be restored automatically, runtime IDL will ignore any `.SAV` file specified on the command line.

RuntimeIcon

This field should contain the name of the `.ICO` file containing your application's custom icon.

InstallNum, SiteNotice, Cookie

These three fields contain the license information for your runtime IDL application. Note that these fields may contain different information for each installation.

IDL Runtime for Windows NT Requires the HASP NT Service

In order to function properly under Windows NT, IDL requires that the HASP NT service version 1.4 be installed. As a runtime IDL developer, you must supply the HASP NT service to your own users and ensure that it is properly installed.

Your runtime IDL distribution contains an application (`hinstall.exe`) that will install or remove the HASP NT service. Use this application to install the service on your users' Windows NT systems. To do so, invoke `hinstall.exe` with the `/i` switch:

```
hinstall /i
```

as part of the installation process for your runtime IDL application. Note that `hinstall.exe` will prompt the user to re-boot the computer after installation of the HASP service.

To remove the HASP service, invoke `hinstall.exe` with the `/r` switch.

Using Runtime Applications

The runtime IDL application (`idlrt.exe`) restores a specified `.SAV` file (or a `.SAV` file specified in the `idl.ini` file, or `runtime.sav`, if no file is specified), and calls the procedure `MAIN`. When `MAIN` returns, IDL exits.

To use the runtime IDL application, specify the appropriate `.SAV` file on the `RuntimeFile` line of the `idl.ini` file. Then alter the "Command Line" field of the Program Manager's "Program Item Properties" dialog to refer to the runtime IDL executable. The Command Line should look something like:

```
c:\myapp\idlrt.exe
```

assuming that your runtime IDL application is located in the directory `c:\myapp`. Alternatively, you can specify which `.SAV` file to restore on the command line itself.

```
c:\myapp\idlrt.exe myfile.sav
```

assuming the runtime IDL distribution tree contains the `.SAV` file `myfile.sav` at the top level, and is located in the directory `c:\myapp`.

Chapter 4

IDL Runtime for Macintosh

Runtime IDL differs from standard IDL in the following ways:

- It will not accept commands interactively. The standard IDL for Macintosh user interface is displayed, but the command input window is not present.
- It will not compile `.PRO` files. Note that this means that runtime IDL will not compile `.PRO` files found in the directories specified by the !PATH system variable in order to find unknown routines at runtime. Runtime IDL *will* restore `.SAV` files found in the directories specified by !PATH.
- Instead of presenting the user with an interactive prompt, runtime IDL starts execution by calling a procedure called `MAIN`, which is assumed to be in the `.SAV` file specified at startup. If no `.SAV` file is specified, runtime IDL looks for a file named `runtime.sav` in the current directory. When `MAIN` returns, runtime IDL ends the session.

On the Macintosh, runtime IDL uses a copy of the standard IDL application that has had a resource altered to specify the runtime IDL feature. You will need to alter a copy of your IDL executable to create the runtime executable for your application.

Creating a Runtime Executable

To create a runtime version of IDL for Macintosh, you will need to alter a resource in the resource fork of an IDL executable file licensed with a runtime license. Resources are added using a resource file editor such as ResEdit. If you do not have a resource file editor, you can obtain ResEdit free from most online services or by calling the Apple Programmers and Developers Association (APDA) at 1-800-282-2732.

Note This procedure embeds the name of your runtime application in the IDL executable file. This allows you to double-click on the IDL icon to run your runtime application. If you do not alter the resource as described below, IDL will attempt to restore a file named `runtime.sav` in the `hook` subfolder of the `lib` subfolder in the runtime IDL distribution tree. This will result in error messages when your customer double-clicks on the IDL icon.

Altering the Resource

The following instructions assume you are using ResEdit to add the necessary resource. If you are using a different editor, the exact actions may be slightly different.

Complete the following steps. *Be sure to make your changes to a copy of IDL that has not yet been licensed.*

1. Double-click on the unlicensed IDL icon to start IDL. Click “License”, and enter your runtime license key information into the License dialog. When IDL runs, you will receive two error messages (a RESTORE error and a “Can’t find procedure” error); you can safely ignore these messages.
2. Quit IDL.
3. Open the IDL application using ResEdit.
4. Open the STR# resources by double-clicking on the STR# icon.
5. Double-click on STR# ID 134. (Figure 4-2 shows the STR # ID in the title of the window.)

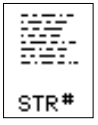

Figure 4-1: The STR# icon

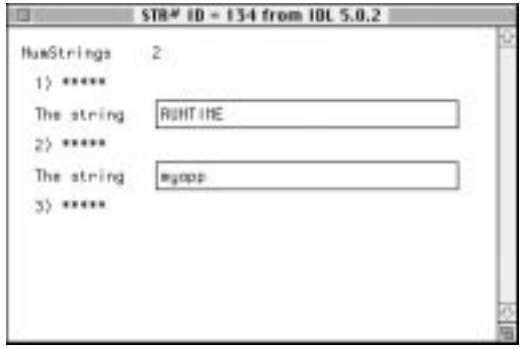

Figure 4-2: The ResEdit resource string dialog.

6. Note that the text field next to the second string is empty. Enter the name of your application's main `.SAV` file *without the `.SAV` extension*. In Figure 4-2, we have entered the name `'myapp'` as an example.

Note The number of strings in the STR # 134 resource depends on the type of IDL license you are using. You may see more than two text fields—for example, if your license includes IDL DataMiner functionality, you will see a field that contains the string `'idl_dm'` in addition to the two shown above.

7. Choose "Save" from the File menu.
8. Quit ResEdit.

Creating a Runtime Save File

If you want your users to see only your application when it runs, or to be able to double-click on the icon for the `.SAV` file that contains your `MAIN` program to start your runtime application, you much change the file type of the `.SAV` file. If you do not change the file type code of your `.SAV` file, your application will still run, but the IDL Development Environment interface will also appear on screen, with the Command Input window desensitized.

Normally, IDL `.SAV` files have the file type "`Sidl`". To create a double-clickable runtime `.SAV` file, use ResEdit to change the file type to "`SdlA`". To change the file type, do the following:

1. Open your `.SAV` file using ResEdit.
2. Select "Get Info ..." from the File menu.
3. Change the "Type" field to "SdlA".

4. Choose “Save” from the File menu.
5. Quit ResEdit.

   Repeat these steps for any `.SAV` files included in your application.

Building a Distribution Tree for your Application

You will need to build a special copy of the full IDL distribution, containing support for only your application. Do this by removing unused files from the full distribution and by adding files used by your application. The easiest way to create a distribution tree is to begin with a copy of the standard IDL distribution (the `IDL 5.2` folder, usually located in the folder `rsi`) and do the following:

1. Remove all `.PRO` files from the `lib` folder and its subfolders. `.PRO` files are ignored in runtime mode.
2. If your application uses Insight functionality or any of the LIVE_* routines, include the file `insight.sav` from the `hook` subfolder of the `lib` folder . Otherwise, remove `insight.sav` and `demo.sav` from the `hook` folder.
3. If your application does not use Insight, remove the `examples` folder. If your application does use Insight, delete everything from the `examples` folder except for the `insight` subfolder. (If your application uses the LIVE_* routines but not Insight itself, you can remove the entire `examples` folder.)
4. Remove the `external` folder.
5. Remove all `*.hlp` and `*.cnt` files from the `help` folder. If your application does not use any of the widget applications for which help text files are included in the `widgets` subfolder of the `help` folder, remove the `widgets` subfolder. Remove the `copyrights` and `motd` subfolders.
6. If your application does not produce PostScript output, remove the `ps` folder from the `fonts` folder in the `resource` folder. The file `hersh1.chr` in the `fonts` folder is required if you use IDL’s default fonts (in a plotting window, for example).
7. If your application does not use any of the map data supplied with IDL, remove the `maps` folder from the `resource` folder.
8. If your application does not use color tables, remove the `colors` folder from the `resource` folder.
9. Remove any `*.txt` files located anywhere in the remaining directories.
10. Substitute a runtime executable (altered as described above) for the IDL for Macintosh executable.

11. Create .SAV files for any .PRO files from the IDL distribution that are used by your application, and place them in the lib directory. Alternatively, these routines can be compiled and saved in your application's main .SAV file.

12. Finally, place your application's main .SAV file, along with any .SAV files containing variable data or additional routines, in the hook subfolder of the lib folder in the distribution tree. You may wish to create an alias to your .SAV file if you wish to have your customers drag the .SAV file icon directly onto the IDL icon. Alter the file type as described in "Creating a Runtime Save File" on page 25. See "Using Runtime Applications" on page 28 for more information on starting your runtime application.

13. If your application does not use IDL DataMiner functionality, remove the ODBC folder in the of the RSI folder. If your application does use DataMiner functionality, see "Building a Macintosh Application that uses the DataMiner" on page A-4 for additional details on building your runtime distribution.

14. If you wish, create an installation program for your application and the runtime IDL distribution tree.

    There are a number of commercial applications available to help you build installers: these include Developer VISE from MindVision Software, Stuffit and Stuffit Installer Maker from Aladdin Systems, and DragInstall from Ray Sauers Associates, Inc. Research Systems has no connection with the companies that produce these applications, and does not make any claims as to the applications' suitability.

Licensing Issues

Runtime IDL uses the a combination of hardware copy protection devices and software license keys, as does standard IDL. (There are different hardware license keys for the runtime and developer versions of IDL.) This means that if you are distributing an application based on runtime IDL, you must also distribute both the hardware and software keys.

Normally, the first time the runtime IDL executable is run, the user is prompted to enter license information. Research Systems strongly recommends that you supply your runtime application "pre-licensed" by running the application and entering the software key information in advance. You will find information on licensing IDL in the *IDL for Macintosh Installation Guide*.

Using Runtime Applications

There are three ways to start runtime IDL for Macintosh:

- Double-click on the runtime IDL icon itself. Runtime IDL will look for the file name specified in the STR # 134 resource (as described in “Creating a Runtime Executable” on page 24) in the `hook` subfolder of the `lib` folder of the IDL runtime distribution, and attempt to restore the file. If it does not find the file, IDL will exit with an error message. This is the preferred method for launching an IDL runtime application.
- Drag the `.SAV` file that contains the `MAIN` program onto the runtime IDL icon. This method is also quite robust.
- Double-click on the `.SAV` file that contains the `MAIN` program. This is the least-desirable method for starting a runtime application, because if more than one IDL executable (runtime or fully interactive) exists on a particular Macintosh, it is not certain that double-clicking on the `.SAV` file will launch the correct executable. This may become a problem if:
  - You, as the runtime IDL developer, have both runtime and non-runtime IDL executables on your machine. In this situation, simply avoid double-clicking on the `.SAV` file—drag it to the proper executable instead.
  - The runtime IDL application end-user has other copies of IDL installed. Since it may not be possible to know in advance whether the end-user has other copies of IDL installed, it is up to you as a runtime IDL application developer to inform your end-users that they should either:

  1. Limit themselves to a single IDL executable. This may not be feasible if the end-user has an older version of IDL that cannot be upgraded to the same version as that used to produce the runtime IDL application.
  2. Explicitly drag the runtime application’s `.SAV` file to the proper runtime IDL executable. This will work in all cases.
  3. Double-click on the runtime IDL executable icon to start the application. Note that this method requires that the name of the `.SAV` file be specified in the STR # 134 resource of the IDL executable, or (if no name is specified in the string resource) that the `.SAV` file be named `runtime.sav`. In either case, the `.SAV` file must be located in the `hook` subfolder of the `lib` folder in the runtime IDL distribution.

Chapter 5

IDL Runtime for Unix and VMS

Under the Unix and VMS operating systems, the same IDL executable is used for both standard and runtime IDL. A copy of IDL used in runtime mode differs from standard IDL in the following ways:

- It checks out a FlexLM feature named `idl_rt` rather than the standard `idl`.
- It will not accept commands interactively.
- It will not compile `.PRO` files. Note that this means that runtime IDL will not compile `.PRO` files found in the directories specified by the !PATH system variable in order to find unknown routines at runtime.
  Runtime IDL *will* restore `.SAV` files found in the directories specified by !PATH.
- Instead of presenting the user with an interactive prompt, runtime IDL starts execution by calling a procedure called `MAIN`, which is assumed to be in the `.SAV` file specified at startup. If no `.SAV` file is specified, runtime IDL looks for a file

named `runtime.sav` in the current directory. When `MAIN` returns, runtime IDL ends the session.

Building a Distribution Tree for your Application

You will need to build a special copy of the full IDL distribution, containing support for only your application. Do this by removing unused files from the full distribution and by adding files used by your application. The easiest way to create a distribution tree is to begin with a copy of the standard IDL distribution (the `idl` directory, usually located in `/usr/local/rsi` under Unix or in `SYS$SYSDEVICE:[IDL]` under VMS) and do the following:

1. Remove all `.PRO` files from the `lib` directory and its subdirectories. `.PRO` files are ignored in runtime mode.
2. If your application uses Insight functionality or any of the LIVE_* routines, include the file `insight.sav` from the `hook` subdirectory of the `lib` directory. Otherwise, remove `insight.sav` and `demo.sav` from the `hook` directory.
3. Remove the `hyperhelp` and `idlhelp` files from the `bin.<platform>` subdirectory of the bin directory, where <*platform*> is the name of the Unix platform for which you are creating a runtime IDL distribution tree.
4. If your application does not use Insight, remove the `examples` directory. If your application does use Insight, delete everything from the `examples` directory except for the `insight` subdirectory. (If your application uses the LIVE_* routines but not Insight itself, you can remove the entire `examples` directory.)
5. Remove the `external` directory.
6. Remove all `*.hlp`, `*.cnt`, and `*.fts` files from the `help` directory. If your application does not use any of the widget applications for which help text files are included in the `widgets` subdirectory of the `help` directory, remove the `widgets` subdirectory. Remove the `copyrights` and `motd` subdirectories.
7. If your application does not produce PostScript output, remove the `ps` subdirectory of the `fonts` subdirectory of the `resource` directory. If your application does not use Object Graphics, remove the `fs` subdirectory of the `fonts` subdirectory of the `resource` directory. The file `hersh1.chr` in the `fonts` subdirectory is required if you use IDL's default fonts (in a plotting window, for example).
8. If your application does not use any of the map data supplied with IDL, remove the `maps` subdirectory of the `resource` directory.
9. If your application does not use color tables, remove the `colors` subdirectory of the `resource` directory.

10. If your application does not allow printing, remove the following from the `xprinter` subdirectory of the `resource` directory:
    - the `ppds` subdirectory
    - the `tfm` subdirectory of the `fontmetrics` subdirectory
11. If your application does not use IDL DataMiner functionality, remove the `dm` subdirectory of the `resource` directory. If your application does use DataMiner functionality, see "Building a Unix Application that uses the DataMiner" on page A-5 for additional details on building your runtime distribution.
12. Remove any `*.txt` files located anywhere in the remaining directories.
13. Make sure you remove the `license.dat` file that belongs with your standard IDL distribution from the main IDL directory when building a distribution tree for your application. You will need to supply a user-specific `license.dat` file for each host your application will run on.
14. Create `.SAV` files for any `.PRO` files from the IDL distribution that are used by your application, and place them in the `lib` directory. Alternatively, these routines can be compiled and saved in your application's main `.SAV` file.
15. Place your application's main `.SAV` file, along with any `.SAV` files containing variable data or additional routines, in the `hook` subdirectory of the `lib` directory in the distribution tree.
16. If you wish, create an installation script for your application and the runtime IDL distribution tree.

Licensing Issues

Runtime IDL uses the same network license manager as standard IDL. This means that if you are distributing an application based on runtime IDL, you must also distribute the appropriate license files to your own users. You should be familiar with the information on installing and managing network licenses contained in the *IDL for Unix* and *IDL for VMS* installation guides.

Using Runtime Applications

When IDL is invoked in runtime mode, it first checks out a license with the feature name "idl_rt". IDL then restores the specified `.SAV` file (or `runtime.sav`, if no file is specified), and calls the procedure `MAIN`. When `MAIN` returns, IDL exits.

To use the runtime version of IDL, run the IDL program with the runtime flag:

Unix

The following command will start runtime IDL and attempt to restore a file named `runtime.sav`, located in the current directory:

```
% idl -rt
```

The following command with start runtime IDL and attempt to restore a file named `myproduct.sav`, located in the directory `/usr/local/rsi/idl/lib/hook`:

```
% idl -rt=/usr/local/rsi/idl/lib/hook/myproduct.sav
```

If the `.SAV` file is located in the current directory, the full path name need not be supplied.

VMS

The following command will start runtime IDL and attempt to restore a file named `RUNTIME.SAV`, located in the current directory:

```
$ IDL/RUNTIME
```

The following command with start runtime IDL and attempt to restore a file named `MYPRODUCT.SAV`, located in the directory `IDL_DIR:[000000]`:

```
$ IDL/RUNTIME=IDL_DIR:[000000]MYPRODUCT.SAV
```

If the `.SAV` file is located in the current directory, the full path name need not be supplied.

Suppressing Startup Messages

You can suppress IDL's normal startup message by including the *quiet flag* when calling IDL. For Unix, add the flag `-quiet` to the `idl -rt` command. For VMS, add the modifier `/QUIET` to the `IDL/RUNTIME` command.

Appendix A

Runtime Applications and the IDL DataMiner

The IDL DataMiner is an Open Database Connectivity (ODBC) interface that allows IDL users to access and manipulate information from a variety of database management systems. Research Systems developed the IDL DataMiner so that IDL users can have all the connectivity advantages of ODBC without having to understand the intricacies of ODBC or SQL (Structured Query Language).

Note The IDL DataMiner option is purchased separately from the core of IDL, and requires a special license. Contact your Research Systems sales representative for details.

The IDL DataMiner makes use of special libraries and support files to provide ODBC functionality. In order to use DataMiner functions, these libraries and support files must be included in the IDL distribution used by your application. This appendix explains how to include the appropriate support files along with your runtime application. Before you begin building a runtime application that uses the DataMiner, you should be familiar with the installation and setup of the

DataMiner subsystem on your development platform. Consult the *IDL DataMiner Guide* for information about the DataMiner.

DataMiner Setup

The IDL DataMiner must be specifically configured for each system on which it is installed. It is your responsibility as a developer to make sure that your customer provides the proper database network locations and site-specific configuration parameters.

Building a Windows Application that uses the DataMiner

In addition to the steps listed in "Building a Distribution Tree for your Application" on page 16, you will need to make sure the following components of the IDL DataMiner ODBC subsystem are installed along with your runtime application.

IDL for Windows uses a group of dynamic link libraries (dlls) to implement ODBC functionality. You will need to ensure that the proper dlls are installed in the `SYSTEM` subdirectory of the `WINDOWS` directory (on Windows NT systems, the files should be installed in the `SYSTEM32` directory):

Windows 95

```
drivset.hlp, ds16gt.dll, ds32gt.dll, mfc30.dll, msvcrt20.dll,
odbc16gt.dll, odbc32gt.dll, odbcad32.exe, odbccp32.cpl,
odbccr32.dll, odbcinst.cnt, odbcinst.hlp, odbcint.dll,
vkorac32.dll, vkss32.dll, vksyb32.dll, vsdrvm32.dll, vsinstsp.sql,
vsiproc.sql, vsiproc4.sql, vsorac.hlp, vsss.hlp, vssyb.hlp,
vvifx532.dll, vving632.dll
```

Windows NT (Intel Processors)

```
drivset.hlp, ds16gt.dll, ds32gt.dll, mfc30.dll, msvcrt20.dll,
odbc16gt.dll, odbc32gt.dll, odbcad32.exe, odbccp32.cpl,
odbccr32.dll, odbcinst.cnt, odbcinst.hlp, odbcint.dll,
vkorac32.dll, vkss32.dll, vksyb32.dll, vsdrvm32.dll, vsinstsp.sql,
vsiproc.sql, vsiproc4.sql, vsorac.hlp, vsss.hlp, vssyb.hlp,
vvifx532.dll, vving632.dll
```

Windows NT (DEC Alpha Processors)

```
ds16gt.dll, ds32gt.dll, mfc30.dll, odbc16gt.dll, odbc32gt.dll,
odbcad32.exe, odbccp32.cpl, odbccr32.dll
```

You will find these files in the `SYSTEM` and `SYSTEM32` directories of your development systems.

Creating an ODBCINST.INI File

Finally, you will need to install a file named `odbcinst.ini` in the `WINDOWS` directory on your customer's system. (If the `odbcinst.ini` file already exists on your customer's system, you will need to modify the file to include information on the Visigenic ODBC drivers supplied with IDL.)

Note You will need to create different `odbcinst.ini` files for each Windows platform.

To create the file, do the following:

1. Open the `odbcinst.ini` file on your development system and make a copy to be installed on the customer's system.
2. Make sure that the following drivers are listed in the `[ODBC 32 bit Drivers]` section:

   ```
   Visigenic VAR Oracle7 (32 bit)=Installed
   Visigenic Sybase DBLib (32 bit)=Installed
   Visigenic Sybase SQL Server 10 (32 bit)=Installed
   ```

   (Other drivers may be listed if the `odbcinst.ini` file already exists.)
3. Make sure that the following drivers are listed in the `[ODBC Drivers]` section:

   ```
   Visigenic Informix5=Installed
   Visigenic Ingres64=Installed
   Visigenic Oracle7=Installed
   Visigenic Sybase DBLib=Installed
   Visigenic Sybase SQL Server 10=Installed
   ```

   (Other drivers may be listed if the `odbcinst.ini` file already exists.)
4. Include all of the sections that contain the word "Visigenic". For example, on a Windows NT machine, the following sections might be included:

   ```
   [Visigenic Informix5]
   Driver=C:\WINNT\System32\vvifmx5.dll
   Setup=C:\WINNT\System32\vvifmx5.dll
   [Visigenic Ingres64]
   Driver=C:\WINNT\System32\vvingr64.dll
   Setup=C:\WINNT\System32\vvingr64.dll
   [Visigenic Oracle7]
   Driver=C:\WINNT\System32\vvorac.dll
   ```

```
Setup=C:\WINNT\System32\vvorac.dll
[Visigenic Sybase DBLib]
Driver=C:\WINNT\System32\vvss.dll
```

Note You will need to alter the path information shown to match the configuration of your customer's system.

Building a Macintosh Application that uses the DataMiner

In addition to the steps listed in "Building a Distribution Tree for your Application" on page 26, you will need to make sure the following components of the IDL DataMiner ODBC subsystem are installed along with your runtime application.

IDL supports two ODBC drivers: the Visigenic Oracle ODBC driver and the Microsoft SQL Server ODBC driver. When you install IDL on your development platform, system extension files for both drivers are installed in your System folder. You will need to make sure that the appropriate system extension is installed in the system folder of your customer's macintosh when your runtime application is installed.

You can find copies of the system extension files in the `ODBC` folder inside the `RSI` folder on the IDL CD-ROM. You will need to install the following system extension files in your customer's System folder:

- ODBC Configuration Manager PPC
- ODBC Driver Manager PPC
- ODBC Cursor Library PPC
- ODBC Setup PPC (a control panel)

In addition, you will need to install either the Visigenic ODBC drivers, the Microsoft SQL drivers, or both, in your customer's System folder. The system extension files for these drivers are located in the `Oracle 7` and `SQL Server` folders in the `ODBC` folder. The Visigenic drivers are:

- VSI Oracle Driver PPC
- VIS Oracle Setup PPC

The Microsoft Drivers are:

- ODBC SQL Server ADSP Netlib PPC
- ODBC SQL Server Setup PPC
- ODBC SQL Server Driver PPC
- ODBC SQL Server TCP Netlib PPC

Once your runtime application is installed, you will need to walk your customer through the process of setting up any network database connections needed for your application to function.

Building a Unix Application that uses the DataMiner

In addition to the steps listed in "Building a Distribution Tree for your Application" on page 30, you will need to make sure the following components of the IDL DataMiner ODBC subsystem are installed along with your runtime application.

On Unix systems, some portions of the ODBC system installation process require access to the database system itself, and thus must be performed after installation. Your customers (or an installation script that you create) will have to execute the following instructions to finish the DataMiner installation process on a Unix system.

Once IDL is installed on the Unix system, do the following:

1. Log on as a user who has access to the IDL directory (referenced by the environment variable $IDL_DIR). By default, this directory is `/usr/local/rsi/idl`.
2. Change to the `dm` subdirectory of your platform-specific `bin` directory:

   `$IDL_DIR/bin/bin.<OS Name>/dm`

   where <*OS Name*> is the name of your operating system.
3. Execute the `build_drivers` script:

   ```
   % ./build_drivers
   ```

Note The `build_drivers` script attempts to perform post-installation procedures on *all* ODBC drivers supplied with IDL. You can safely ignore any error messages received for ODBC drivers not present on your system.

Note The `build_drivers` script is generated at IDL install time and is only valid for the location where IDL was originally installed. If you have moved the IDL directory since installation, you will need to edit the paths in this script.

Once you have run the `build_drivers` script, copy the `odbc.ini` file to `.odbc.ini` in your home directory:

```
% cp $IDL_DIR/resource/dm/<OS Name>/odbc.ini ~/.odbc.ini
```

The `odbc.ini` file is used by the ODBC system to determine what ODBC drivers are installed. You might have to edit this file to take into account any driver-specific information fields. See Chapter 5 of the *IDL DataMiner Guide* for details.