

Guide to the Build Kit

*Part 4 Volume 16 of the SEDRIS
Technology Documentation Set*



1 July 2011

The principal sponsoring agency for the SEDRIS project is the DoD Modeling and Simulation Coordination Office (M&S CO), 1901 North Beauregard Street, Suite 500, Alexandria, VA 22311.

DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited.

Table of Contents

SECTION	PAGE
1 INTRODUCTION	1
1.1 Prerequisites.....	1
1.2 For Additional Information on SEDRIS	2
1.3 Document Conventions and Notations	3
2 FIRST STEPS AND QUICK OVERVIEW	4
2.1 Overview of SEDRIS Software	4
2.2 Supported Platforms.....	4
2.3 Downloading the SEDRIS SDK.....	5
2.4 Installing the SEDRIS SDK.....	5
2.5 What's in the SEDRIS SDK.....	6
2.6 Building the Libraries and Applications.....	7
2.7 What's Next.....	9
3 RUNNING THE STANDARD APPLICATIONS	10
3.1 Shared Library Issues.....	10
3.2 Sample Runs	11
4 CREATING A SEDRIS APPLICATION	14
4.1 Integrating With the Build Environment	14
4.2 Adding Source Files.....	16
4.3 Building and Testing.....	19
4.4 Distributing Your Application	19
5 ADDITIONAL DOCUMENTATION FOR DEVELOPERS	21
5.1 For SEDRIS Data Consumers.....	21
5.2 For SEDRIS Data Producers.....	21
6 WHERE TO GO FROM HERE	22
A VERSIONING ISSUES.....	23
A.1 File Format API Version Compatibility	23
A.2 Application Library Dependencies	23
B FREQUENTLY ASKED QUESTIONS (FAQ)	24
B.1 How do I get the release back to its pristine start condition?	24
B.2 Why isn't the Model Viewer application compiled successfully?	24
B.3 My applications compile fine, but they don't run. What is happening?.....	24
B.4 How can I create SEDRIS transmittals?	24
B.5 In Unix, I need to specify some libraries in my application's makefile	24
B.6 In Unix, I can't get the build environment to find OpenGL (or Mesa) in my system! ..	24
B.7 How do I get my application (e.g., Model Viewer) to find the GLUT libraries?.....	25
B.8 In Win32, I compiled the SEDRIS SDK using the static libraries workspace	25

List of Figures

FIGURE		PAGE
FIGURE 2-1.	SEDRIS SDK directory structure.....	6
FIGURE 3-1.	<i>Depth</i> program output	12
FIGURE 3-2.	<i>Model Viewer</i> screen shots.....	13
FIGURE 3-3.	<i>SEDRIS Transmittal Browser</i> screen shot.....	13
FIGURE 4-1.	The <code>list_models.c</code> source code file	17
FIGURE 4-2.	The <code>list_models.cpp</code> source code file	18
FIGURE 4-3.	Output from the <i>List Models</i> Application	19

List of Tables

TABLE		PAGE
TABLE 1-1.	Document Conventions and Notations	3

Section 1

1 INTRODUCTION

The purpose of this document is to provide software programmers new to the SEDRIS development environment with a concise description of the steps involved in creating, building, and testing SEDRIS applications.

This document is composed of several sections describing the different aspects and requirements in building SEDRIS software:

- Section 2: *First Steps and Quick Overview* – details the platforms supported, how to download the SEDRIS SDK from the SEDRIS web site (www.sedris.org), and how to compile the SDK.
- Section 3: *Running the Standard Applications* – explains the requirements on environment and data sources for running the SEDRIS applications included in the SEDRIS SDK.
- Section 4: *Creating a SEDRIS Application* – details the process of creating a new application to be compiled using the SEDRIS libraries and build environment.
- Section 5: *Additional Documentation for Developers* – describes additional documentation resources for SEDRIS developers.

This document also includes two appendices describing other aspects of the SEDRIS development process.

- Appendix A: *Versioning Issues* – discusses issues concerning management of different versions of the SEDRIS software and library dependencies.
- Appendix B: *Frequently Asked Questions (FAQ)* – a compilation of common problems and questions related to the SEDRIS build environment and the development of SEDRIS applications.

1.1 Prerequisites

It is assumed that the reader is already familiar with appropriate portions of Part 4: *Technical Reference Set* of the SEDRIS Technology Documentation Set (*see* Section 1.2) prior to reading this document.

1.2 For Additional Information on SEDRIS

This document is part of a larger set of overview and technical documents on SEDRIS, the SEDRIS Technology Documentation Set. The SEDRIS Technology Documentation Set describes the why, what, and how of SEDRIS.

The SEDRIS Technology Documentation Set, listed below, is available at the SEDRIS web site (www.sedris.org). An overview of the contents of each "part" of the documentation set is provided in Part 1. A detailed description of the individual "volumes" contained in Part 4 is provided in Volume 1.

- Part 1: *Introduction to SEDRIS and the Technology Documentation Set*
- Part 2: *SEDRIS and The Synthetic Environment Domain*
- Part 3: *SEDRIS Basics*
- Part 4: *Technical Reference Set*
 - Volume 1: *Technical Overview*
 - Volume 2: *The SEDRIS Data Representation Model*
 - Volume 3: *Examples of Using the Data Representation Model*
 - Volume 4: *Topology Technical Guide*
 - Volume 5: *Control Link Technical Guide*
 - Volume 6: *Data Tables Technical Guide*
 - Volume 8: *Images and Color Models Technical Guide*
 - Volume 9: *Attribute Inheritance and Context Technical Guide*
 - Volume 10: *Environmental Data Coding Specification (EDCS) Reference Manual*
 - Volume 11: *Spatial Reference Model (SRM) Reference Manual*
 - Volume 12: *Application Program Interface Overview*
 - Volume 13: *How to Extract Data from SEDRIS Transmittals*
 - Volume 14: *How to Produce SEDRIS Transmittals*
 - Volume 15: *SEDRIS Transmittal Format Description*
 - Volume 16: *Guide to the Build Kit*
 - Volume 17: *SEDRIS Reference Manual*
 - Volume 18: *Reference Implementation Listings*
- Part 5: *Tools and Utilities User's Guide Set*
 - Volume 1: *Transmittal Browser User's Guide*
 - Volume 2: *Checker User's Guide*
 - Volume 3: *Depth User's Guide*
 - Volume 4: *Feature Viewer User's Guide*
 - Volume 5: *Model Viewer User's Guide*
 - Volume 6: *Netscape Plug-In User's Guide*
 - Volume 7: *Ocean Profile User's Guide*
 - Volume 8: *SEE-IT User's Guide*

- Volume 9: *Side-by-Side Viewer User's Guide*
- Volume 10: *Wind Map User's Guide*
- Volume 11: *API Implementations and Format Conversions User's Guide*

- Part 6: *Procedures and Processes Manual*

1.3 Document Conventions and Notations

Throughout the SEDRIS Technology Documentation Set, special conventions and notations are implemented in the written text of documents to distinguish between and add emphasis to various computer-related or SEDRIS-specific terms. Table 1-1 defines these conventions and notations.

TABLE 1-1. Document Conventions and Notations

CONVENTION	EXAMPLE	DEFINITION
Reverse video	Open	window buttons/selections/choices/ <i>etc.</i>
<Key or Class>	<Shift>	keys on the keyboard
	<Colour Data>	Data Representation Model class names* (<i>e.g.</i> , Colour_Data = <Colour Data>)
Courier font	src/lib/api_impl/	directory, file, transmittal, library names
	gmake env	command line input
	BUILD_MODE	variable names, coding examples
	"false"	variable values
<i>italics</i>	<i>www.sedris.org</i>	Internet addresses
	<i>gmake</i>	software, window, mode, function, option, <i>etc.</i> names
	Part 1: <i>Intro to ...</i>	document/section titles
	<i>not always</i>	for emphasis
"quotes"	"generic"	highlight or call attention to

* Underscore characters (`_`) are *not always* replaced by angle brackets (`< >`), depending upon where data class names appear. Angle brackets appear within paragraph text (to avoid word wrap), whereas underscore characters are used in lists or other stand-alone text (for clarity) and are used in coding examples.

Section 2

2 FIRST STEPS AND QUICK OVERVIEW

This section describes the SEDRIS supported platforms, how to download and build SEDRIS software, and possible courses of action for SEDRIS developers.

2.1 Overview of SEDRIS Software

There are three parts to developing and using SEDRIS software:

- Transmittals – the data that will be consumed or produced by an application. Sample SEDRIS Transmittal Format (STF) files are included in the SEDRIS SDK releases. Additional sample data is available from the SEDRIS Data web site (data.sedris.org).
- Libraries – the SEDRIS libraries that implement the functionality needed to access, interpret, and manipulate SEDRIS data. The SEDRIS SDK includes support for reading and writing STF transmittals. You can obtain the SEDRIS SDK in both source code or compiled binaries.
- Applications – the applications that actually consume or produce SEDRIS data. The SEDRIS SDK includes several applications that consume SEDRIS data, such as *Depth*, *Syntax Checker*, and *Model Viewer*, along with other applications that produce SEDRIS data, such as *STF Test* and *ITR Test*. Additional tools and utilities are available from the SEDRIS Tools web site (tools.sedris.org) and from SEDRIS Associates and Partners.

The remainder of this section is an exploration of the SEDRIS SDK (also referred as “the SDK”), including how to download, install, and compile the SDK, along with a detailed description of the SDK contents including its directory structure.

2.2 Supported Platforms

SEDRIS is supported on a variety of platforms, including the following:

- Windows with Microsoft Visual C++ 6.0, .NET 2003, or 2005
- Linux version 2.4 and higher with GNU compiler
- Silicon Graphics, Inc. (SGI), Irix 6.5 (n32 only)
- Sun Microsystems, SunOS 5.6 and 5.7 with Forte compiler

If you require SEDRIS on other platforms, send e-mail to se-scrccb@sedris.org with information on the platform you would like to see SEDRIS supported. If you have tested the SEDRIS SDK on other platforms not listed above or you would like to help in porting SEDRIS to additional platforms, please let us know.

2.3 Downloading the SEDRIS SDK

All SEDRIS software should be downloaded from the SEDRIS web site (www.sedris.org). The SEDRIS SDK can be downloaded in both source code and pre-compiled binaries. The complete SEDRIS source code is packaged in a single file called `sedris_c_sdk_4.1.4.tgz` for Unix systems or `sedris_c_sdk_4.1.4.zip` for Win32 systems. The file names for the binary versions are named similarly but with a description of the platform that was used to compile the libraries. In addition, there are C++ API releases available with similarly named package files.

2.4 Installing the SEDRIS SDK

If you are installing the SEDRIS SDK in a Unix system, the software needs to be extracted from the downloaded file using the *gunzip* and *tar* programs. You can verify these programs are available by just attempting to execute those commands. Alternatively you can use the GNU tar program *gtar* for uncompressing and extracting all files at once. From a shell window, perform the following steps.

- Select a directory where you would like the SEDRIS software to be extracted, for example in `/usr/local/SEDRIS`, and create it if necessary. For example:

```
mkdir /usr/local/SEDRIS
cd /usr/local/SEDRIS
```

- Now extract the contents of the file. For the standard *gunzip* and *tar* programs, execute the following:

```
gunzip -c <path>/sedris_c_sdk_4.1.4.tgz | tar xf -
```

or, if you have the GNU tar program (*gtar*), execute the following:

```
gtar -xzf <path>/sedris_c_sdk_4.1.4.tgz
```

where `<path>` is the directory where the `sedris_c_sdk_4.1.4.tgz` file was downloaded.

The above commands will create a new directory with a name that matches the version of the SEDRIS SDK downloaded. For example, if you downloaded version 4.1.4, a directory named `sedris_c_sdk_4.1.4` will be created. This directory will hereafter be referred to as the

<root> directory. All directory references from now on are assumed to be relative to this directory. For example, if you extracted the SEDRIS SDK under the /usr/local/SEDRIS/sedris_c_sdk_4.1.4 directory, then a reference to the src directory is actually a reference to <root>/src, which translates to the /usr/local/SEDRIS/sedris_c_sdk_4.1.4/src directory.

If you are installing the SEDRIS SDK in a Win32 system, then you need to extract the contents of the downloaded file to an appropriate directory using an unzipping utility, such as WinZip, PKZIP, or 7zip.

For detailed documentation on requirements and other issues related to your version of the SEDRIS SDK, refer to the documentation located in the <root>/docs directory by opening the index.htm file with a web browser.

2.5 What's in the SEDRIS SDK

The SEDRIS SDK includes a complete set of libraries and utilities for developing SEDRIS applications that can read and write SEDRIS Transmittal Format (STF) files (a.k.a., STF transmittals), in addition to the EDCS SDK and SRM SDK. Figure 2-1 shows the directories that are present in the SEDRIS SDK source release. If you downloaded a pre-compiled binary release, the directory tree structure will be somewhat different (e.g., the src directory will not be available, and a bin, lib, and include directories will be present).

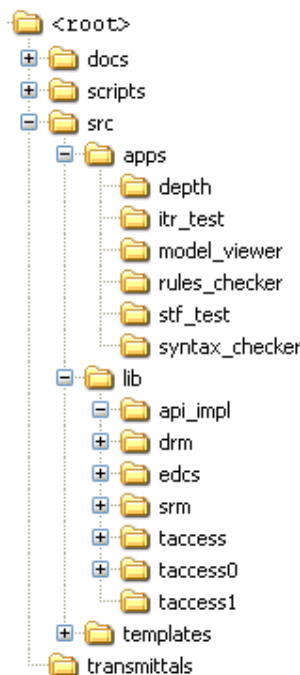


FIGURE 2-1. SEDRIS SDK directory structure

The `docs` directory contains documentation such as instructions for building the software, reference material for the technology components, application documentation, and migration documents.

The `scripts` directory contains Perl scripts that can help you migrate your SEDRIS software from a previous release to the new one. You can ignore this directory if you don't have any software to migrate.

The `src` directory contains all the source code for libraries and applications. The libraries for all the SEDRIS technology components are located under the `src/lib` directory. The library component that can read and write STF transmittals is located under the `src/lib/api_impl/stf_api` directory, and directories below it. SEDRIS applications are located under the `src/apps` directory, where you will find other directories named by the application they contain. For example, there are directories for the *Depth*, *Model Viewer*, *Rules Checker*, and *Syntax Checker* applications, with matching directory names such as `depth`, `model_viewer`, `rules_checker`, and so on. There are some slight differences in this directory structure between the C and C++ API releases since some applications do not appear in both releases.

The `transmittals` directory contains sample STF transmittals. You can use the `test.stf` transmittal in this directory, along with the standard applications supplied with the SEDRIS SDK, to verify that your installation is working properly. More information on doing this is provided in later sections of this document.

2.6 Building the Libraries and Applications

The SEDRIS software will compile and work "out-of-the-box" on all supported systems. However, read the following sections carefully to verify that your system has all the necessary applications and libraries, such as different versions of the *make* utility and compiler versions.

The *Model Viewer* included with this release relies on OpenGL and GLUT. You will need the following libraries installed on your system to successfully compile this application:

- [OpenGL](#) (a 3D graphics rendering API) version 1.1 and above, or [Mesa](#) (a viable replacement for OpenGL).
- [GLUT](#), a toolkit that simplifies the development of cross platform OpenGL applications, version 3.7 and above.

The SDK build environment for Unix requires the use of the GNU *make* utility (the *gmake* program) version 3.78.1 or higher; **older versions will not work**. GNU *make* is freely available under the GNU General Public License and may be downloaded via anonymous FTP from several [mirror sites](#). You can verify your availability of *make* and its version by executing the command `make -v`. If you get a "command not found" error, you will need to install it. You will also need the OpenGL and GLUT libraries to build the *Model Viewer* tool. There are binary

distributions of these libraries available for most operating systems, so follow the instructions provided in those distributions. Typically, this will involve installing the headers under your compiler's "include/GL" directory (e.g., "/usr/include/GL"), libraries under your linker's "lib" directory (e.g., "/usr/lib"), and the runtime libraries under a directory accessible by the `LD_LIBRARY_PATH` variable (e.g., "/usr/lib"). A **make** command in the `<root>` directory will recursively go through the SDK directory structure and build all libraries and applications provided with the distribution using the default settings.

The SDK build environment for Win32 is based on Microsoft Visual C++ 6.0 and Visual C++ .NET 2003 project files. Hence, a default installation of one of these compilers is sufficient to successfully build all the SDK libraries and applications. The only exception is the *Model Viewer* tool, which requires that the OpenGL and GLUT libraries be installed. The OpenGL headers and libraries should have been installed by default when you installed Visual C++, and your Windows system should also have all the OpenGL runtime libraries you will need. For the GLUT files you can get pre-compiled binary distributions that include the headers for compiling, the libraries for linking, and the DLLs for runtime. Install the header files in the "include" directory of the compiler and the libraries into the "lib" directory. In Visual C++ 6.0, these are "VC98\include\GL" and "VC98\lib" of the compiler's installation directory. The runtime libraries "glut.dll" and/or "glut32.dll" in your Windows "system" directory (typically "C:\Windows\System" in Windows 98/Me/XP and "C:\Winnt\System32" in Windows NT/2000). Follow these steps to build all the SDK libraries and applications:

1. From Windows Explorer, locate and open (i.e., double click) the `win32_headers.bat` file in the `<root>` directory of your installation. This step creates the top level `include` directory and copies to this directory all the header files used to access the technology components.
2. If you want to compile the SDK using dynamic linking (i.e., linking against DLLs), open the `vcpp_dynamic.dsw` project workspace file using Visual C++ 6.0 (or open the `vcpp_dynamic.sln` project file for Visual C++ .NET). For static linking, open the `vcpp_static.dsw` file instead.
3. From the Visual C++ IDE, select "all_sdk" as your "Active Project" and "Release" or "Debug" for your "Active Configuration". For example, to compile in "Release" mode, choose "Build→Set Active Configuration..." and select the "all_sdk - Win32 Release" entry.
4. From the "Build" menu, choose "Rebuild All".

Notes for Win32:

- If you will be switching between the static and dynamic project workspaces, make sure to use the "Rebuild All" command so that the previous compilation files are deleted.
- You can build individual applications by making them the current "Active Project" ("Project→Set Active Project" menu), and choosing "Rebuild All".

The time to build the SEDRIS SDK will vary, depending on your system characteristics, typically lasting less than 10 minutes. If you are having problems compiling, refer to Appendix B: *Frequently Asked Questions (FAQ)*, and the release documentation for more information.

The building process will be complete when the prompt re-appears on the screen and no errors are displayed. If no errors are shown, you are ready to try some of the sample applications included in the SEDRIS SDK and to create your own SEDRIS applications. If any errors or warnings are shown during compilation please send them to se-coders@sedris.org along with a description of your build environment so that we can troubleshoot their cause. If only a few warning messages are shown it is likely that you can safely ignore them (but you can still email them to us for further review).

For more detailed information on build settings and commands, please refer to the SEDRIS SDK documentation. For help, comments, and bug reports please send email to help@sedris.org. If you are a SEDRIS Associate or Partner, please use se-coders@sedris.org.

2.7 What's Next

If you would like more information on running the SEDRIS SDK applications, read through Section 3: *Running the Standard Applications*. If you want to create a new SEDRIS application, read Section 4: *Creating a SEDRIS Application*. Other documents of interest to developers are described in Section 5: *Additional Documentation for Developers*.

Section 3

3 RUNNING THE STANDARD APPLICATIONS

This section describes how to run the SEDRIS applications included in the SEDRIS SDK. These applications depend on several environment settings that need to be set appropriately. The rest of this section describes these requirements and the output expected from the applications when run against the sample transmittals in the SDK `transmittals` directory.

3.1 Shared Library Issues

There are several factors that may cause an application to fail to run successfully, such as a failure to find a shared library, an incompatible STF file version, a file read error, or running out of memory. There may also be problems accessing the window manager for your particular system, such as having an inappropriate setup for an X-Window client. This document cannot attempt to solve all possible problems, but a few of the most common ones are discussed in the next few paragraphs.

The default settings in the SEDRIS SDK build the libraries and applications using shared libraries. Hence, the system dependent "standard search paths" are used to locate the libraries when an application is run. Both Unix and Win32 systems have similar procedures for finding these shared libraries, but each requires environment variables to be set appropriately. In order to run the included applications, you will need to add to your current path the paths to the application executables and the shared libraries.

In Unix systems, and if you downloaded and installed a source release, executing the command **make env** retrieves the names of the application and library directories. Look for the lines containing "EXE_DIR" and "LIB_DIR". For example, you can use the command **make env | grep _DIR** to list all the directories specified by the build environment. Alternatively, and also if you installed a binary release, you may execute the **ls** command in the `<root>/bin` and `<root>/lib` directories (and their subdirectories) to find where the applications or libraries reside. For example, in a Linux system, the path to the executable applications might be `<root>/bin/linux-2.4.7-10-i386/OPT`. You will need to add the applications directory to your `PATH` environment variable, and the libraries directory to your `LD_LIBRARY_PATH` directory. For example, to add the previous Linux executable applications path in a Bash shell, you can use the command:

```
PATH=$PATH:<root>/bin/linux-2.4.7-10-i386/OPT.
```

In Win32 systems, the executable applications reside in the `<root>\bin\Release` or `<root>\bin\Debug` depending on build mode. Similarly, the libraries reside in either

<root>\lib\Release or <root>\lib\Debug. From a DOS Prompt, add these directories to the PATH environment variable by using the command:

```
set PATH=<root>\bin\Release;<root>\lib\Release;%PATH%
```

or use "Debug" instead of "Release" if you compiled the SDK in Debug mode.

For more information on the capabilities of your system and whether you need to install additional libraries, consult with your system administrator and have them review the information in this section.

3.2 Sample Runs

The *Depth* program can be used to display the objects in an STF transmittal by printing them to the console in ASCII format. Thus, you can pipe the output of *Depth* to a file to examine its contents using a text editor. The *Depth* program can be executed starting at any object in a transmittal, which is quite useful for examining only portions of a transmittal. The following paragraphs describe the process of running *Depth* and examining its output.

To run *Depth* on the transmittal `test.stf` located in the `transmittals` directory, perform the following steps:

- From a shell or prompt window, change the current directory to the `transmittals` directory.
- Make sure you have set your environment variables appropriately as described in the previous section.
- Execute the command:

```
depth test.stf > depth_out.txt
```

The *Depth* program will now generate a textual description of all objects in the `test.stf` transmittal, saving the output to a file named `depth_out.txt`. After the program finishes executing, you can open the `depth_out.txt` file using a text editor and examine the contents of the transmittal. Figure 3-1 shows a sample output of the *Depth* program.

```

SEDRIS Transmittal test.stf Opened.
- [:0,0,0] Transmittal Root
  - [:0,0,1] Access
  - [:0,0,2] Citation
    - [:0,0,3] Absolute Time
    - [:0,0,4] Responsible Party
  - [:0,0,5] Data Quality
  - [:0,0,6] Description
  - [:0,0,7] Absolute Time
  - [:0,0,8] Transmittal Summary
  - [:0,0,9] Model Library
    - [:0,0,10] Model
      - [:0,0,11] Geometry Model
        - [:0,0,12] Union Of Primitive Geometry
          - [:0,0,13] Polygon
            - [:0,0,14] Inline Colour
              - [:0,0,15] Primitive Colour
                - [:0,0,16] Ambient Colour
                  - [:0,0,17] RGB Colour
                    - [:0,0,18] Diffuse Colour
                      - [:0,0,19] RGB Colour
            - [:0,0,20] Translucency
            - [:0,0,21] Presentation Domain
          - [:0,0,22] Vertex
            - [:0,0,23] LSR 3D Location
        . . . .
      - [:0,0,107] Vertex
        - [:0,0,108] LSR 3D Location
      - [:0,0,109] Geometry Model Instance for Test Model
        (associated to [:0,0,11] Geometry Model)
Maximum Tree Height = 10

      2      Absolute Time
      1      Access
      4      Ambient Colour
      1      Citation
. . . .
Total object count = 110
. . . .

```

FIGURE 3-1. *Depth* program output

There are many more applications that use SEDRIS technologies. Many of these are not included in the SEDRIS SDK, but are available from the SEDRIS Tools web site at <http://tools.sedris.org>. Some sample screen shots of these applications are shown next.

The *Model Viewer* application can display the 3D models found in a SEDRIS transmittal. Figure 3-2 shows a couple of images produced by this application.

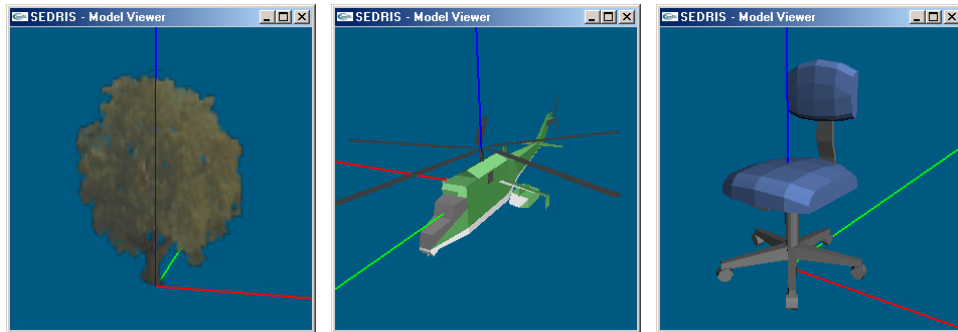


FIGURE 3-2. *Model Viewer* screen shots

SEDRIS Focus and SEDRIS Transmittal Browser applications (neither is included in the SEDRIS SDK) can be used to browse the hierarchy of a SEDRIS transmittal and display the fields of transmittal objects. Figure 3-3 shows a screen shot of the SEDRIS Focus application.

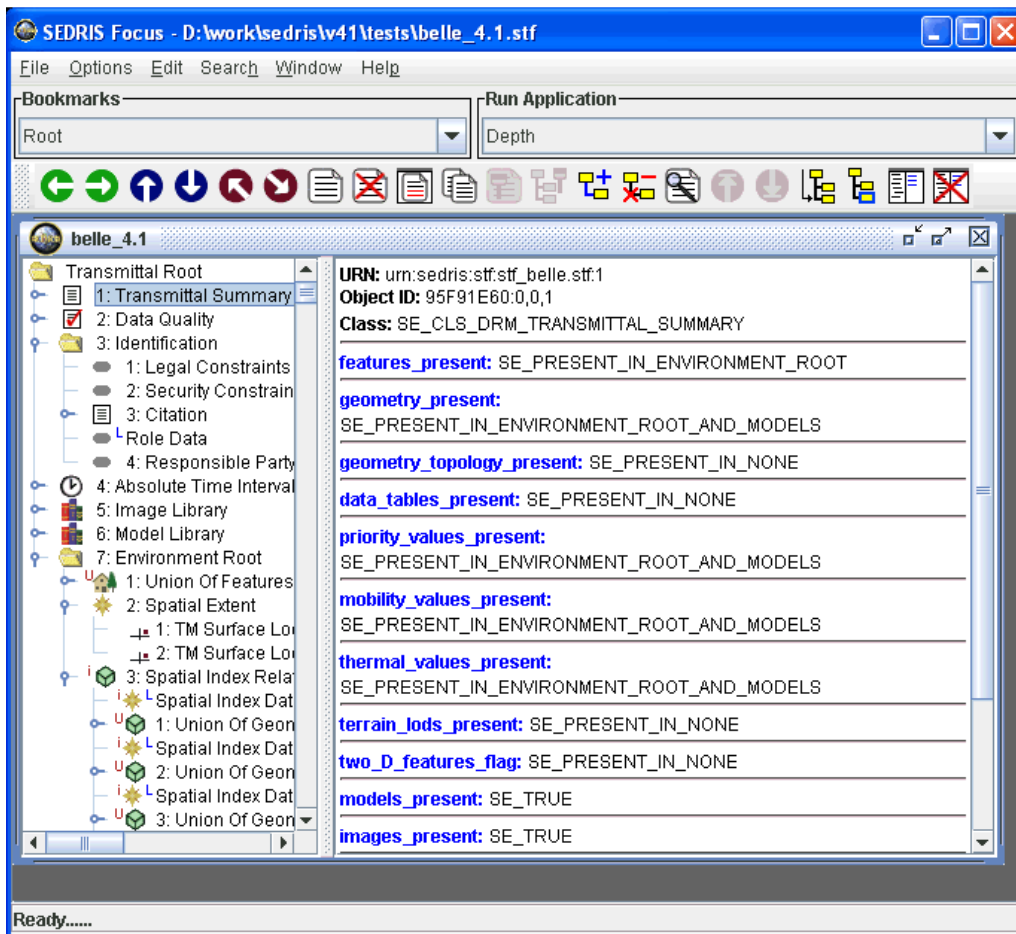


FIGURE 3-3. *SEDRIS Focus* screen shot

Section 4

4 CREATING A SEDRIS APPLICATION

This section describes steps that can be used to build an application integrated with the SEDRIS build environment. This is not intended to be a complete example of the SEDRIS API usage. For more in-depth documentation on the SEDRIS API usage, refer to Section 5: *Additional Documentation for Developers*.

The following sections describe the building of a simple application. The steps explored include where to put the application in the SEDRIS build environment directory structure, adding source code files, and building and testing the application executable. All of these steps are presented by creating "from scratch" an application, called *List Models*, that reads a transmittal using the SEDRIS API and prints all the names of the models contained in it. The developer could extend the completed application to enhance its functionality.

Due to changes in the build environment, most of the information in this section is valid only in Win32 systems. However, the sample application source code will compile and work on Unix systems.

The following sections assume that you have correctly installed and compiled the SEDRIS libraries as described in Section 2: *First Steps and Quick Overview*.

4.1 Integrating With the Build Environment

The initial question a first-time SEDRIS application developer would probably ask is: *"Where should I put my application?"* All applications provided by the SEDRIS SDK are located in the `src/apps` directory. There is no strict requirement to do so, but putting your application in this directory will make for an easier integration with the SEDRIS build environment. If you choose to place your application sources separate from the SEDRIS build environment, you will need to change the "include" and "libraries" options in your project to point to the corresponding SEDRIS SDK directories.

To compile your own applications from the Win32 build environment, use one of the provided applications as a model for your project files (the ".dsp" files), edit them as needed, insert them into the appropriate workspace (either "vcpp_dynamic.dsw" or "vcpp_static.dsw") by using the Visual C++ menu command "Project→Insert Project Into Workspace...", and set the project dependencies as done for the other applications. As an alternative, you can create the project from scratch directly from the SEDRIS SDK workspace. The following steps show you how to do this.

The following steps create an application that is linked dynamically against the SEDRIS SDK libraries. The application will be created in the "<root>/src/apps" directory using Microsoft Visual C++ 6.0. If you want to create an application in another directory, you will need to change the relative paths as appropriate.

1. Open the "vcpp_dynamic.dsw" file.
2. Select "Project→Add To Project→New...".
3. Choose the "Projects" tab in the "New" dialog, and select "Win32 Console Application".
4. In the "Location:" box, click the "..." button and navigate to the "<root>/src/apps" directory, and click the "OK" button.
5. In the "Project name:" box, enter "list_models".
6. Verify the "Add to current workspace" choice is selected, and click the "OK" button.
7. In the next dialog, leave the selection to "An empty project", click the "Finish" button, and the "OK" button in the next dialog.
8. Your new project is now in the workspace, and it should be "bolded" indicating it is the current project. If it is not, use the "Project→Set Active Project" menu to make it active.
9. Choose "Project→Add To Project→Files...", and add your source code files to it.
10. Choose "Project→Settings...", select the "list_models" project on the left side of the "Project Settings" dialog, and choose "Win32 Release" for the "Settings For:" drop down menu on the top left of the dialog.
11. In the "General" tab, enter ".././bin/Release" in the "Output files:" box.
12. In the "C/C++" tab, choose "Code Generation" in the "Category:" drop down menu, and choose "Multithreaded DLL" in the "Use run-time library:" drop down menu.
13. Now choose "Win32 Debug" for the "Settings For:" drop down menu.
14. In the "C/C++" tab, choose "Code Generation" in the "Category:" drop down menu, and choose "Debug Multithreaded DLL" in the "Use run-time library:" drop down menu.
15. In the "General" tab, enter ".././bin/Debug" in the "Output files:" box.
16. Now choose "All Configurations" for the "Settings For:" drop down menu.
17. In the "C/C++" tab, choose "Preprocessor" in the "Category:" drop down menu, and enter ".././include" in the "Additional include directories:" box.

18. Click the "OK" button in the "Project Settings" dialog.
19. Choose "Project→Dependencies...", and select "list_models" in the "Select project to modify:" drop down menu.
20. Click the box beside the "sedris" entry in the "Dependent on the following project(s):" scroll list so that a check mark is inside it, and click the "OK" button.
21. You should now be able to build your project by pressing the "F7" key or from the menu by choosing "Build→Build list_models.exe".
22. The executable will be located in the "<root>\bin\Release" or "<root>\bin\Debug" directory depending on your build mode.

You are now ready to add some source code files to your project.

4.2 Adding Source Files

The next step is to create a SEDRIS application by using the SEDRIS API. In this section we will create a source code file that uses the SEDRIS API to open a transmittal and print all of the names of the models in the transmittal.

If you are using the “C” API, create a file named `list_models.c` in the `list_models` directory. Use the code shown in Figure 4-1 for the contents of the file, and add this file to your project. If using the “C++” API, create a file named `list_models.cpp` using the contents shown in Figure 4-2.

The code shown is sufficient for opening any STF transmittal. The code performs some simple error checking to verify that the transmittal can indeed be opened and that a <Model Library> object exists. If a <Model Library> instance is available, the code retrieves all of the <Model> component instances and displays their names.

```

/* List Models - Build Kit document sample code - SEDRIS C API 4.1.x */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "se_read0.h"

void main ( int argc, char *argv[] )
{
    SE_Transmittal  trans;
    SE_Object       root_obj, model_lib_obj, model_obj;
    SE_Store        store;
    SE_URL          url;

    printf("SEDRIS - List Model Names Application\n\n");

    if ( argc < 2 ) {
        printf("Usage: list_models <transmittal_name>\n\n");
        exit(-1);
    } else if ( SE_SetURL(argv[1], &url) != SE_DRM_STAT_CODE_SUCCESS ) {
        printf("Error - Unable to set URL for %s!\n", argv[1]);
        exit(-1);
    } else if ( SE_OpenTransmittalByLocation(&url, SE_ENCODING_STF,
        SE_AM_READ_ONLY, &trans) != SE_RETCOD_SUCCESS ) {
        printf("Error - Unable to open SEDRIS Transmittal %s!\n", argv[1]);
        exit(-1);
    } else if ( SE_GetRootObject(trans, &root_obj) != SE_RETCOD_SUCCESS ) {
        printf("Error - Unable to get the root object\n");
        exit(-1);
    } else if ( SE_CreateStore(SE_ENCODING_STF, &store) != SE_RETCOD_SUCCESS ) {
        printf("Error - Unable to create fields store\n");
        exit(-1);
    }

    if ( SE_GetComponent(root_obj, SE_CLS_DRM_MODEL_LIBRARY,
        SE_FALSE, SE_FALSE, SE_ITRBEH_RESOLVE,
        &model_lib_obj, NULL ) == SE_RETCOD_SUCCESS )
    {
        SE_FIELDS_PTR model_attr;
        int n = 0;

        printf("\nReading models...\n");
        while ( SE_GetNthComponent(model_lib_obj, SE_CLS_DRM_MODEL,
            n+1, &model_obj, NULL ) == SE_RETCOD_SUCCESS )
        {
            n = n+1;

            if ( SE_GetFields(model_obj, store, &model_attr)
                == SE_RETCOD_SUCCESS )
            {
                if ( model_attr->u.Model.name.characters )
                    printf("Model %d = %s\n", n,
                        model_attr->u.Model.name.characters);
                else
                    printf("Model %d = NO NAME\n", n);
            }
            else
                printf("Model %d = Error getting fields!\n", n);
            SE_FreeObject(model_obj);
        }
        SE_FreeObject(model_lib_obj);
        printf("There were a total of %d models in this transmittal\n", n);
    }
    else
        printf("\nError - can't find Model Library!\n");

    SE_FreeObject(root_obj);
    SE_FreeStore(store);
    SE_FreeURL(&url, NULL);
    if ( SE_CloseTransmittal(trans) != SE_RETCOD_SUCCESS )
        printf("Error - failure closing transmittal!\n");

    printf("\nDone!\n");
}

```

FIGURE 4-1. The list_models.c source code file

```

/* List Models - Build Kit document sample code - SEDRIS C++ API 4.1.x */
#include <iostream>

#include "seWorkspace.h"
#include "seTransmittal.h"
#include "seIterator.h"
#include "seDRMAll.h"

using namespace sedris;
using namespace std;

int main( int argc, char *argv[] )
{
    cout << "SEDRIS - List Model Names Application\n\n";

    if ( argc < 2 ) {
        cout << "Usage: list_models <transmittal_name>\n\n";
        return -1;
    }

    try
    {
        seWorkspace wksp;
        seTransmittal xmtal;
        seDRMTransmittalRoot root_obj;
        seDRMModelLibrary model_lib_obj;

        wksp.openTransmittalByFile(argv[1], xmtal);
        xmtal.getRootObject(root_obj);

        if (root_obj.getComponent(model_lib_obj))
        {
            int n = 0;
            seIterator iter;
            seDRMModel model_obj;

            model_lib_obj.getComponentIterator(iter, SE_CLS_DRM_MODEL);

            cout << "\nReading models...\n";

            while ( iter.getNext(model_obj) )
            {
                ++n;

                if ( model_obj.get_name().characters )
                    cout << "Model " << n << " = " <<
                        model_obj.get_name().characters << endl;
                else
                    cout << "Model " << n << " = NO NAME\n";
            }

            cout << "There were a total of " << n <<
                " models in this transmittal\n";
        }
        else
            cout << "\nError - can't find Model Library!\n";

        cout << "\nDone!\n";
    }
    catch ( seException &e )
    {
        cerr << "Error - " << e.getWhat() << endl;
        return -1;
    }

    return 0;
}

```

FIGURE 4-2. The list_models.cpp source code file

4.3 Building and Testing

Building the *List Models* program is a straightforward task. Simply press the "F7" key or, from the Visual C++ menu, choose "Build→Build list_models.exe". This will compile and link the program sources against the SEDRIS API libraries. The object and executable files will be located in the same directory where all other SEDRIS applications are located. If you have set up your environment variables appropriately, you should be able to run your program as if you were running any of the other SEDRIS SDK applications.

You are now ready to test your new SEDRIS application. This application takes only one option, the name of the transmittal to open. From the `transmittals` directory, execute the command:

```
list_models test.stf
```

If your application fails to run with a message like "A required DLL file was not found," you will need to set the `PATH` environment variable to point to the directory under the `lib` directory where the SEDRIS libraries are. Check Section 2: *First Steps and Quick Overview* for more information on this. A sample output from a typical transmittal containing some models (not the "test.stf" transmittal) is shown in Figure 4-2.

```
SEDRIS - List Model Names Application

Reading models...
Model 1 = house1
Model 2 = house2
Model 3 = tree
Model 4 = bush
There were a total of 4 models in this transmittal

Done!
```

FIGURE 4-3. Sample output from the *List Models* Application

4.4 Distributing Your Application

The first choice developers need to make is whether they will distribute the application as binary, source code, or both.

Distributing the source code is probably the most flexible, but you will need to provide instructions on how to build your application. If your application depends on other libraries, such as OpenGL or Xlib, then you will need to explicitly state so in your documentation, and possibly provide developers with information on how to obtain those libraries. The *Model Viewer* application uses both OpenGL and GLUT, and so you may want to take a look at its

Makefile and project files to learn how to merge your application with the SEDRIS build environment.

Another option is to distribute the compiled and linked executable file. The build environment has the option of building dynamically- or statically-linked applications. A dynamically-linked application should be packaged with the shared libraries, whereas a statically-linked application doesn't need them. An advantage of distributing a dynamically-linked application is that an update to the SEDRIS libraries is as easy as replacing the previous libraries, but a statically-linked application will require re-linking. An advantage of distributing a statically-linked application is that the total size of the application will typically be smaller than the dynamically-linked application package, and that the environment path variable doesn't require to be set to the SEDRIS SDK `lib` directory.

Section 5

5 ADDITIONAL DOCUMENTATION FOR DEVELOPERS

This section describes the available documentation for the SEDRIS developer. The SEDRIS Technology Documentation Set (*see* Section 1.2) includes several documents that explain in detail various elements of the SEDRIS technologies.

SEDRIS distinguishes between two typical users of SEDRIS software: data consumers and data producers. Data consumers are those developers that write software using the SEDRIS API to read SEDRIS transmittals. Data producers, on the other hand, are developers who create SEDRIS transmittals using the write functions of the SEDRIS API. The following subsections address the documentation available for these two SEDRIS users.

5.1 For SEDRIS Data Consumers

If you want to learn more about how to use the SEDRIS API to read SEDRIS transmittals, read Part 4, Volume 1: *Technical Overview* for an overview of SEDRIS software. Then proceed to Part 4, Volume 12: *Application Program Interface Overview*, followed by Part 4, Volume 13: *How to Extract Data from SEDRIS Transmittals*. Finally, more in-depth information on the SEDRIS Data Representation Model can be found in Part 4, Volumes 2-11.

In addition to the documentation, exploring the SEDRIS SDK applications, such as *Depth* and *Model Viewer*, will provide you with a more real-application perspective on using the SEDRIS API.

5.2 For SEDRIS Data Producers

If you want to learn more about how to use the SEDRIS API for creating SEDRIS transmittals, read Part 4, Volume 14: *How to Produce SEDRIS Transmittals*. In addition, you will need a thorough familiarity with the SEDRIS Data Representation Model, which is provided in Part 4, Volumes 2-11. The SEDRIS SDK application *STF Test* and *ITR Test* are a good starting point to learn how to create SEDRIS transmittals.

Section 6

6 WHERE TO GO FROM HERE

After reading this document, it is recommended that the following documents within the SEDRIS Technology Documentation Set be reviewed, as appropriate.

- Part 3: *SEDRIS Basics* – for a detailed overview of SEDRIS.
- Part 4: *Technical Reference Set* – for the technical aspects of SEDRIS.
- Part 5: *Tools and Utilities User's Guide Set* – for instructions and guidance on the use of the SEDRIS tools and utilities.
- Part 6: *Procedures and Processes Manual* – for SEDRIS project procedures/processes.

Appendix A

A VERSIONING ISSUES

The SEDRIS SDK may have update releases from time to time. In addition, some applications depend on software libraries that may not be available in your system. The following are typical issues when dealing with SEDRIS and other libraries.

A.1 File Format API Version Compatibility

From time to time, there may be updates to the SEDRIS SDK releases. In general, update releases are labeled with the main release number and an appended update number in the form of “major.minor.update”, such as 4.1.4 where 4.1 is the main release version and 4 is the update number. For example, release 4.2 *may* have an update to complete the functionality of some API function or a bug fix, and that new release *may* be labeled 4.2.1. Unless explicitly stated, all releases with the same main release version number (that is, the major and minor part) can read and write transmittals produced by any of them.

A.2 Application Library Dependencies

The *Model Viewer* application uses OpenGL and GLUT for its graphical user interface. Hence, this application will only compile successfully in systems that have support for both OpenGL and GLUT. You need GLUT version 3.7 or higher in your system to be able to compile the *Model Viewer*. If you do not have OpenGL support in your system, you may want to see if Mesa (a free replacement for the OpenGL API) is available for your system at www.mesa3d.org.

Appendix B

B FREQUENTLY ASKED QUESTIONS (FAQ)

B.1 How do I get the release back to its pristine start condition?

In Unix systems, execute the `gmake distclean` command from the root installation directory. In Win32 systems, use the “Build→Clean” command from Visual C++. This will remove all the compiled directories along with their contents.

B.2 Why isn't the Model Viewer application compiled successfully?

The *Model Viewer* needs the GLUT libraries to compile. More information on GLUT, how to obtain it, and how to install it is available in the SEDRIS SDK Build Kit document.

B.3 My applications compile fine, but I can't run them. What is happening?

If your applications are built using shared libraries, you may need to set the `LD_LIBRARY_PATH` (for Unix) or `PATH` (for Win32) variables to point to the directory where the shared libraries are installed. The shared libraries are put by default under a system dependent directory in the `lib` directory.

B.4 How can I create SEDRIS transmittals?

The SEDRIS SDK includes functions used to create SEDRIS transmittals. The *STF Test* and *ITR Test* applications show how to create and add objects to SEDRIS transmittals. You can also use the SEDRIS Focus application to create simple transmittals.

B.5 In Unix, I need to specify some libraries in my application's makefile (or my API implementation's makefile) that the rest of SEDRIS doesn't need. How do I do that using the build environment?

Set an `EXTRA_LINKFLAGS` variable in your makefile to specify any extra library search paths you need for that makefile. Set a `LOCAL_LIBS` variable in your makefile to specify any extra libraries you need for that makefile.

B.6 In Unix, I can't get the build environment to find OpenGL (or Mesa) in my system!

This could be due to one of two causes.

- 1 Your application's makefile doesn't add the OpenGL paths to the search paths for include files and/or libraries. To fix this:

- a) add `$(GLINCL)` to the definition of your `LOCAL_INCLUDES` variable (e.g., `LOCAL_INCLUDES = blah` becomes `LOCAL_INCLUDES= blah $(GLINCL)`).
- b) add `$(GLLIBS)` to the definition of your `SYS_LIBS` variable (e.g., `SYS_LIBS = blah` becomes `SYS_LIBS= blah $(GLLIBS)`).

2 In the `templates` directory, the template corresponding to your compiler defines the macros `GLINCL`, `GLDEFS`, and `GLLIBS`:

- `GLINCL` specifies the search path for the OpenGL include directory
- `GLLIBS` specifies the search path for the OpenGL lib directory

If necessary, redefine these variables to point to the correct location for OpenGL on your system.

B.7 How do I get my application (e.g., Model Viewer) to find my installation of the GLUT libraries?

In Unix, add the following to your application's makefile:

- `$(GLUTINCL)` to your `LOCAL_INCLUDES`
- `$(GLUTLIBS)` to your `SYS_LIBS`

Also check that the template for your platform in the `templates` directory puts the correct directories in these variables.

For Win32 systems, see the SEDRIS SDK Build Kit document.

B.8 In Win32, I compiled the SEDRIS SDK using the static libraries workspace and everything worked fine. However when I switched to the dynamic libraries workspace, the applications failed to link properly. Why?

You need to choose “Rebuild All” from the Build menu, so that all the previously compiled files are re-compiled (both applications and libraries). All the source code needs to be compiled using the same workspace for the DLL exports to be setup appropriately.