Abstract

Data is worthless without knowing what the data represents, and you need metadata to efficiently manage large data sets. As computing power becomes cheaper and more data is derived, metadata becomes more important than ever.

Today researcher are setting more experimental scientific workflows than before. As a result a lot of steps leading to the implementation are skipped. The leading steps usually included documenting the work, which is not a central part of the more experimental approach. Since documenting is no longer a natural part of the scientific workflow, and the workflow might be changing a lot though its lifetime, many data products are lacking documentation.

Since the way the scientist work have changed, we feel the way they document their work need to change. Currently there is no metadata system that retrieves metadata directly from the scientific process without having the researcher having to change his code or in other ways manually set up the system to handle the workflow. This thesis suggest ways to automate the metadata retrieval, and shows how two of these techniques can be implemented. Automatic linage and metadata retrieval will help the researchers document the process a data product have gone though. My implementation shows how to retrieve linage and metadata by instrumenting Interactive Data Language scripts, and how to retrieve linage from shell script by looking at the system calls made by the executable.

The implementation discussed in this paper is intended to be a client for the Earth System Science Server, a metadata system for earth science data.

Acknowledgements

This thesis is the culmination of my Master of Technology in Computer Science studies at Norwegian University of Science and Technology. The thesis was written at the Environmental Information Lab, University of California, Santa Barbara. I would like to thank professor James Frew at University of Santa Barbara California and my teaching supervisor, professor Ingeborg Sølvberg, at the Norwegian University of Science and Technology for all the help and support I have received.

Contents

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

This thesis discusses different approaches to do effortless and unintrusive linage extraction on scientific processes. The results are intended to be used as a tool to document workflows for researchers who process digital data.

To my best knowledge there does not exist a system today that does this without forcing the users to make their workflow in a specific way, like wrapping their code in special scripts or manually inserting log statements into the code.

Although some metadata needs to be entered by a human, a lot of the effort in documenting the scientific work done on computers can be completely automated. A lot of information retrieval can be achieved by looking at how the computer program behaves or how it interacts with the rest of the system. As computing power is relatively cheap on todays marked we suggest sacrificing performance to relieve the user of having to input the metadata that the system is able to automatically retrieve.

An important part of the information we are trying to retrieve is linage. Linage is the history of the data set. Linage shows what data is used as source, and how the source data is processed to create the a new data set. To be able to construct the linage graph we need to know the dependencies between data and processes. Through this thesis I will discuss possible ways to, without effort, detect the interaction between programs and their environment to retrieve the linage. The result is a client which is part of a data management system referred to as Earth System Science Server.

It is important that the system does not interfere with the work of the user. If the user feels the presence of any sensor on his computer is interfering with his work it needs to be easy to completely deactivate the sensor. In this way the user can turn off the sensors on his computer when doing development and activate them when the algorithm is at a later stage.

1.1 Earth System Science Server (ES^3)

1.1.1 What is ES^3?

The Earth System Science Server is based on the Earth System Science Workbench (ESSW). ES^3 is a infrastructure to nonintrusive manage large amounts of data distributed over multiple locations. Today data is distributed from one center, and researchers who wish to publish processed data needs to submit it back to the central location. ES^3 enables the different data producers to publish their data through their data center. As long as the different data producers uses metadata in the same format and makes it available to a central location, it is possible to search through it and get a pointer to where the data can be found.

Metadata is an important part of a system that manages large sets of data products. If we implement a metadata standard, metadata and data sets can be shared easier between

different entities. By creating consistent metadata records and submitting it to the ES^3 the data consumers and suppliers are able to do the following:

- Share data sets between data centers
- Analyze the history of a dataset
- Data and process quality ensuring
- Search metadata to find products
- Easier identify a given data set
- Identify faulty data sets, and track the errors trough the processing steps

1.1.2 ES^3 technologies

As its name suggest the ES^3 has a client server architecture. The part of the server which interacts with the client is called the core. The core has two major roles. The first, is to receive information about data granules, the second is to present the information and make it available to a search interface. The core part of the ES^3 solution is currently under development and the details of this system are not clear at this point.

ES^3 uses the Alexandria Digital Library Middleware [1] to provide the search capability. The Middleware is a HTML based distributed digital library with collections of georeferenced materials. The Middleware provides an XML interface that provides clients with the ability to search the data collections. The ES^3 project has it's own ADL Middleware client, but the user can use another client if they desire.

ES^3 provides a constant namespace of the data granules though a HTTP redirector service we call the MODster. MODster is a distributed, decentralized inventory server for Earth science data granules. Data providers can tell the MODster server where their granules are located, and when a user is retrieving a granule from the MODster they get redirected to the real location. The key element that makes this work is that most granules have a well defined naming convention. By using the MODster we can use static links to the granules inside the ES^3 and in written papers. Also, if the user knows the name of a product, he or she can go directly to the MODster and find the data.

1.2 How does this thesis fit into ES^3

The result of the work behind this thesis is the foundation for a client which can submit linage and metadata to the ES^3 core. The data that is sent to the core is mainly collected without interaction with the user, but it is possible for the user to provide additional metadata by inserting statements into the source code.

The client is a generic linage and metadata collector extended to interact with ES^3. The client is designed without ES^3 specific modules up to the point where the data is submitted to the ES^3 core. This modular design allows a lot of freedom for changes to both the ES^3 core and the client without having to update the whole codebase.

Chapter 2

Related research

Currently there are no metadata system capable of automatically retrieving linage from scientific processes, without special preparation of the workflow. The Ufo [2] and Transparent result caching [4] projects shows how system tracing can be applied in application with some of the same properties as ES^3.

2.1 Ufo

The Ufo [2] project describes how to extend the functionality of a standard operating system entirely in the user space. The purpose of the Ufo is to make an extension of the local file system to include remote files accessed by e.g. ftp or http.

The authors discussed different ways to extend the operating system to get access to the file system. They are arguing that a extension in user level is favorable because it does not require the user to put an extra load on administrators, and it is more secure since it cannot introduce vulnerabilities which exposes the system.

Ufo is implemented using system call tracing. When a system call is issued, Ufo is able to pick up the call and change the parameters before it is sent to the kernel. When Ufo captures system calls to open a remote file, Ufo downloads the file and changes the parameter of the open command to the path of the newly downloaded local file. At this point the process can read and write to the file since it got a pointer to the local file. When the process closes the file handle, Ufo is responsible to write the changes back to the origin.

Ufo is implemented in two modules, the Ufo module and the Catcher. The Ufo module is the implementation of the remote file system and is responsible for the local cache. The cacher is the part of the system that is most interesting in regards to the ES^3 client. The Catcher is using the proc file system to intercept system calls, and informs the Ufo module about the call before it changes the parameters and sends the call down to the kernel.

Even though the goal of Ufo is different from ES^3, they show that it is possible to retrieve parts of the linage from arbitrary processes without any recompilation.

2.2 Transparent result caching

Transparent result caching [4] traps system calls to determine the dependencies among input files, development tools and output files by tracing the process system calls. They suggest storing linage locally on the computer, and use it to automatically update output files when a input file is changed. For example say the file foo.h is used to make the program foo. After compiling foo, you change the content of foo.h. When detecting the change of foo.h, TREC would automatically recompile foo.

The TREC project needs to collect a lot of the same same information ES^3 is going to collect. Although TREC does not record information about interprocess dependencies, it

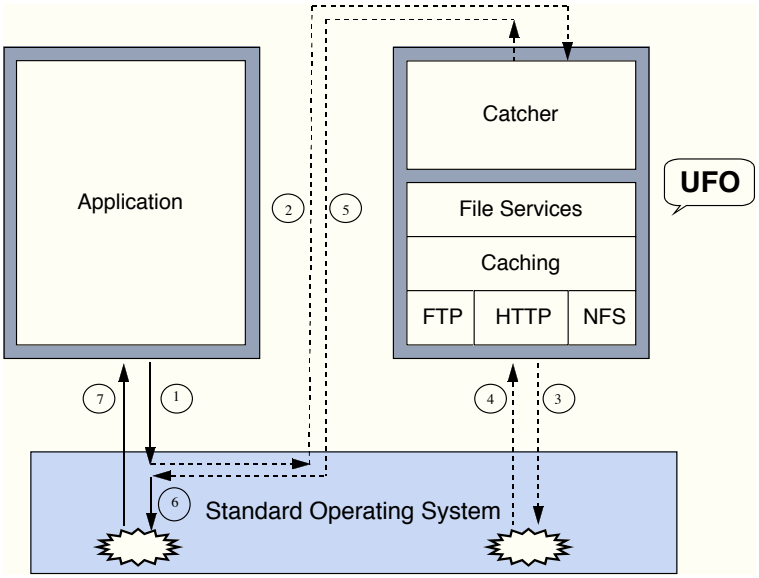

Figure 2.1: The UFo architecture

records dependencies between processes and files. Their work shows that it is possible to retrieve the linage and metadata from a process by tracing the system calls it makes.

Transparent result caching captures the dependency information by intercepting system calls using a native kernel tracing mechanisms. For each process, information about the command line arguments, environment variables, process parent, process children, files read and files written gets stored and made available to query.

TREC relies on the Solaris proc filesystem to intercept the system calls necessary to build the dependency information. The proc filesystem offers an interface to important kernel data structures that provide information about the state of a running kernel by use of special files. Most UNIX systems use a quite similar interface as the Solaris system. The system calls intercepted by TREC are: open, fork, fork1, creat, unlink, exec, execve, rename and exit. Command line arguments and environment variables are available trough the exec system calls. To reduce unnecessary overhead, they do not monitor read and write system calls. Instead they assume if a file is opened for writing that the process intends to write to the file, and if a file is opened for reading only the process reads from the file.

2.3 Earth System Science Workbench (ESSW)

ESSW [3] is a system designed to handle linage and metadata for objects in scientific workflows. The ESSW system needed more interaction than we wish to have in ES^3. Before linage from a workflow can be sent to the reservoir, the object in the workflow needs to be manually registered with the server. Workflows that is monitored by EESW has to make calls to a custom application programming interface that needs to be added by the maintainer of the workflow. The linage recored of an object can be queried though a web application.

ESSW addresses the same general problem we which to address with ES^3, but requires a lot of interaction and tuning before it is able to run. Also, a workflow will not be able to run under ESSW if the connection to the server is broken.

Chapter 3

Possible techniques

3.1 Instructing the source code

If our system is able to read and make changes to the source code before it is interpreted, we can change or insert code to make the program log the appropriate metadata.

Instruction can be done by inserting some simple logging statements before or after specific routine invocations. Some level of parsing is required to insert the code in the right place, retrieve the arguments and make sure that the inserted code does not change the outcome of the process. The code inserted and the original function call needs to be in the same code block to ensure that flow control statements does not make one executed and not the other.

Another way of achieving this is simply by changing the function call to a wrapper function under our control. The wrapper function can log the execution and return the value returned from the original function. This way we don't need to parse the arguments of the function call, and we don't need to worry about the call not being in the same block.

A preprocessor can be used to alter the source code. Some ways of invoking the preprocessor could make the process transparent, others could have the user explicitly invoke the preprocessor. By making the preprocessor a part of the compilation process would hide this step, and the user does not need to know about the preprocessor. All the source code containing operations we wish to log needs to be run through the preprocessor in order to enable the necessary logging.

It is reasonable to assume that the number of routines that initializes I/O is relatively low, probably in the order of a dozen or two. A preprocessor can have a list of the routines that we are interested in logging, and make changes to the code according to this list.

In the situation where the user directly enters the commands in a shell the notion of a preprocessor does not make sense. In this case it is possible to put a thin layer over the shell, which reads commands from the input and passes them over to the underlying shell. This way the layer on top of the shell can make changes to the commands before passing them through. A integration of the two techniques is possible if the users starts a shell and then calls a script to be executed in the shell. If the layer on top of the shell intercepts the call to execute the script, and invokes the preprocessor before the script is executed, the preprocessing would be transparent to the user.

3.2 Overriding routines

The idea is to override a routine with a wrapper routine. The wrapper routine would log the event before calling the real routine. Different environments offer different ways to overriding routine calls:

- Most object oriented languages offer a way of making subclasses which would override the super-classes functions.
- Changing the namespace: Some languages enable you to define your own namespaces. By changing this namespace to something we control we would be able to intercept the routine invocations.
- Linking to different versions of an external libraries. In many cases the functions we are interested in are a part of an external library. We can replace the external library with a library that is wrapped.
- By changing the order the environment looks for objects, functions or files. Most environments has some way of defining the search path. By inserting our routines ahead of the other routines in the search path they would be invoked.

3.3 Passive monitoring

Passive monitoring is to look at which commands are issued to the execution environment when a program is executed. This method is unique because we don't need to do any changes before the program is executed, since all hooks will reside in the execution environment.

This method is useful for information retrieval when the source code is not immediately accessible, and would be the only way if we don't have access to the source code. The downside is that we make it harder to record information that is internal to the program, such as the value of a given variable.

This idea can be implemented at the operating system level, or in the case of a virtual machine the interaction can take place at a higher level. It is likely that many environments already has support for this sort of logging. In a *nix environment the commands *strace* and *lstrace* give you system calls and library calls respectfully. The commands can either be issued at the time of execution or they can connect to a process already running. A simple daemon can be made to look for specific processes, and logs the system calls of those processes.

3.4 Interpreting the code

Most of the time when you look at a source code file you can easily establish which I/O operations are likely to take place during execution. When a computer is given the same code it certainly can tell which I/O operations it is doing while executing the code. Since this is true it should be possible to give the source code or even a pice of compiled code to an interpreter and it would be able to determine the arguments of certain function calls. The interpreter could process the code with far less resources than needed if the code really was executed, since the interpreter does not need to execute all statements in the code to retrieve the information desired.

Since the code in many cases will contain statements that will control the flow of execution like if-, while- and for-statements the flow may depend on complex computations. The complexity is reduced if the general execution flow does not depend on or is derived from external functions or data. It might be reasonable to assume that in many cases the I/O is not dependent on flow control statements, and therefore the interpreter does not need to be full-blown. Even with this assumption the interpreter has to be somewhat complex, and if the assumption is wrong the cost of making an interpreter would be high.

Chapter 4

The frameworks

4.1 Interactive Data Language (IDL)

IDL, the Interactive Data Language, is a commercial computing environment for data analysis and visualization made by Research Systems Inc. Supported operating systems include Windows, Linux and Mac OS X. Earth scientists use IDL because it provides a higher level of abstraction when handling image- and scientific data formats.

IDL has a graphical interface which provides the user with a large variety of tools, but I'm going to focus on the scripting interface because this is how automated tasks are completed.

Traditional interpreted languages does parsing, lexical analysis and execution on each statement before going to the next statement. IDL on the other hand uses a two-step process where the scripts are compiled and interpreted separately. Routines are compiled into a internal binary format when a command is issued. After the routine is compiled the interpreter executes the code. The interpreter is a simple stack based postfix language, in witch each instruction corresponds to a primitive of the IDL language.

The core of the system is the interpreter, which contains the stack. The stack consist of IDL_VPTR structures, which are pointers to IDL_VARIABLE structures. The IDL_VARIABLE structures are implementations of the variables in the IDL environment. During execution of a statement the interpreter pops pointers from the top of the stack and pushes the a pointer to the resulting IDL_VARIABLE back on the stack.

It is possible to invoke IDL functions from other programming languages and more importantly IDL has support for interacting with external programming languages like C/C++, Java, Fortran, COM and ActiveX. Some of the extensions run in the same memory space as IDL and could be a used to pull metadata out of IDL. The external modules made in C/C++ needs to be compiled for each computer architecture. It turns out that it is hard to access the information this way, and the memory structure might be dependent on the version of IDL.

When we make IDL call external modules we use the *CALL_EXTERNAL* function. When calling *CALL_EXTERNAL* IDL loads the external routine into the same memory space. This gives us potential a low level access to the entire system including the stack.

4.1.1 Instructing the source code

IDL has a simple programatic syntax and it is easier to preprocess than most programming languages. A preprocessor capable of changing function calls to wrapper functions does not need to contain a full lexical analyzer for the language, it can be achieved by a stream editor.

To be able to preprocess the scripts we need to intercept the source code before IDL interprets it. IDL has several ways you can invoke code. From the console, the commands

.RUN [filename], .COMPILE [filename] and *@[filename]* will load a the script file. Some of the commands will also execute the script. IDL can read scripts and routines from files by passing the filename as an argument when the IDL executable is called, and IDL can read scripts from standard input. If the filename is passed as an argument or the script is piped into IDL, the syntax changes slightly. I call this syntax the interactive syntax. When interactive syntax is used every procedure and function declaration has to start with the *.RUN* keyword. If the script is executed in any of the other ways the *.RUN* keyword can not be present.

The *.COMPILE* command explicitly compiles a routine to the internal binary format and loads it into the memory. To allow the user to load scripts within a IDL shell we can create a procedure which preprocess the code and offers the functionality of *.COMPILE*. I have called this function *ES3_COMPILE*. It is not possible to have *ES3_COMPILE* call *.COMPILE*, since IDL does not allow any command that starts with a dot to be invoked within a script. It is however possible to use *RESOLVE_ROUTINE* instead. The *RESOLVE_ROUTINE* procedure compiles user-written or library routines in a similar fashion. IDL looks for a file with the same name as the routine in IDL's search path, and loads the routine from the file when found. If the *COMPILE_FULL_FILE* option is given to the *RESOLVE_ROUTINE* procedure, it will compile all other routines inside the file.

Since *RESOLVE_ROUTINE* is looking for a routine, the script-file need to contain a procedure or function with the same name as the file it resides in. To enable *RESOLVE_ROUTINE* to load the file the preprocessor needs to have an option of creating a dummy procedure. The dummy procedure has no other function than to enable the file to be loaded dynamically from a script.

Since there is no ideal way of intercepting the code in the IDL environment if it is passed to IDL as an argument or through standard input, the preprocessor needs to be accessible outside IDL. If the script is normally piped into IDL, the script can now be piped though the preprocessor and then into IDL. This can ether be done explicitly by the user or by making a wrapper for the IDL executable. By having the preprocessor a standalone executable the user can also preprocess the code manually before executing it in a IDL environment which is not rigged in any way.

The preprocessor renames the built-in read and write function calls in the script to call the wrapper functions called ES3_*function name*. The wrapper functions need to be created by the preprocessor before the execution starts.

4.1.2 Overriding functions

The functions and procedures that come with IDL are distributed ether as binary libraries or IDL code. The lower level functionality and some complex routines are implemented in binary libraries, and the higher level functions are implemented in in IDL code. Most libraries that reads structured data from a file, for instance images and video, are distributed as IDL code. Although some functions like READ_TIFF and READ_PNG are distributed only as binaries.

When a routine is invoked IDL searches for a match in the following order:

1. System functions
2. Dynamically loadable modules
3. Compiled IDL functions or procedures
4. Files matching the name of the routine in the current directory
5. Files matching the name of the routine in directories specified by the internal environment variable *!PATH*

Since the higher level libraries is a part of the last item on the list it is easy to override these routines. This can be done by making routines with the same name as the routines we wish to override and placing them in a directory being searched earlier than the directory containing the original routine. This routine needs to have the full functionality of the original routine since we are not able to call the original routine while a routine with the same name is being executed. This is unfortunate since we could be put in a situation where the code needed to be updated for each new release of IDL. To only override the higher level routines is not enough since the higher level libraries does not cover all the ways a file can be accessed.

Routines that are part of dynamically loadable modules can not be overridden. The only possible way to change the behavior of routines in dynamically loadable modules we don't have the source code for, is to remove the original routine and replace it with our own implementation. Even if we did this there is a chance that the binary libraries that use the routine we replaced would be statically linked to the original library and would ether use the original version or break when executed.

When the higher level libraries opens a file, they use the *OPEN* command. If we override this command, all files that are used would be traced. Sadly *OPEN* is hard to override since it is a system routine and, IDL does not provide any way of override system routines. Also, it is unlikely that the binary libraries use this routine, it is more likely they use standard C/C++ or fortran libraries. Therefore overriding this routine would not be a complete solution.

4.1.3 Passive monitoring

There is no specific support for passive monitoring in IDL. The only way to do this is to trace the system calls issued by the IDL executable at the operating system level. By tracing the system calls made by IDL, all file operations would be recorded. It is however harder to look at the different parameters of the IDL scripts.

4.1.4 Interpreting the code

There is no good publicly available language definition for IDL, which makes the IDL language syntax best defined by the behavior of the latest version of the IDL interpreter. Making a fully blown IDL interpreter would involve a lot of work, and only minor parts could be reused for other programming languages.

4.2 Bash

Bash is the GNU implementation of the original Unix shell. Bash is short for Bourne-Again Shell. The name is a tribute to Steven R. Bourne, the creator of the original shell for Unix-compatible operating systems. There is a vide verity of shells out there, and most of them are open source. Bash is the default shell in most modern operating system except Windows, although you are able to run bash on windows. It is reasonable to assume bash is one of the most commonly used shells.

The purpose of shells are to provide a wrapper around the execution of executables. The simplest shell would read one line either from a script or the console and issue an execute command to the underlaying system. Shells usually forks/clones a child process before they send a execute command, and waits until the child process is finished.

Even if the main purpose of a shell is to issue execute external commands most modern shells provide other functionality to the user. Modern shell functionality includes debugging, subroutines, flow control, string manipulation and file pattern matching.

One feature that is extensively used in shell script are UNIX pipes. A UNIX pipe is a way to channel the input and output of processes to other processes or files. One of the most

common way to crate a pipe in a shell is to use the pipe character (|). The pipe character allows you to redirect the output from one process to into another process. You can also create files on a UNIX based system that are pipes, called named pipes. Named pipes are not regular files, but can be treated as files inside the program. In fact the program does not need to distinguish between regular files and named pipes.

The syntax of a bash script can get fairly complex when piping and other build-in functionally is used. In the scope of this thesis the syntax of bash is too complex for me to write a lexical analyzer.

4.2.1 Instructing the source code

In a shell environment the number of commands capable of initializing I/O is high. Almost all standard commands can read or write to a file, and the shell itself can be made to do I/O. In a shell script you would often find one line containing multiple commands connected together with pipes. To successfully insert logging statements at a line like this you need to call the logging function between the execution of each command. In bash there is a build-in function that does that, called trap. If you use the trap function with the debug flag you can execute a logging function for each command executed by the shell. The major drawback of using trap is that in the most common version of bash, no information is given about the command about to be executed.

Even if the commands executed and it's parameters where known we would not be able to keep track of all the I/O preformed by the script. Since each command is a standalone executable, programs will do I/O operations that are not possible to detect by looking at the environment the program is running in.

4.2.2 Override functions

Override executables can be done in a shell environment by inserting a directory containing an executable in the environment variable *PATH* before the directory of the original executable occurs. To ensure that user does not override our commands by changing the *PATH* variable, the trap function can be used to do the necessary changes before execution of the command.

Bash is flexible enough to create a generic command that can override all commands that and will do all the necessary logging before executing the parent command. The full path of the command being overridden can can be determined by the command name, parameters and the search path, which are available to the command through *$PATH*, *$0* and *$@*. All that needs to be done to override a command is to make a link, with the name of the command being overridden, to the generic override command. The folder containing the link should be placed before the folder of the original command in *PATH*

4.2.3 Passive monitoring

Bash debugger

Recent versions of bash has a more sophisticated debugger built in. Currently bash 2 is the most widespread version, and includes only a simple execution trace. If you issue the command '*set -x*', the commands will print as they are executed.

For example if you open a bash shell and type the following:

```
$ set -x
$ cat < bar | grep foo
```

This will be printed to stdout:

```
+ cat
```

```
+ grep foo
```

As you can see from the output above we get the trace of commands executed and their parameters, however some information is lost. We are not able to determine that cat is reading the file bar and the output is directed to grep. In fact we are not able to determine any interaction the program had with the environment.

We are able to collect a lot of information by analyzing the data we get from a execution trace like this when the programs executed are kown commands. We are able to parse the parameters and determine what I/O each execution is doing. When it comes to custom made executables we are no longer able to determine the I/O by looking at the parameters. We could make assumptions that would work for many cases, but not for all.

Trace system calls

Strace is a system call tracer, a debugging tool which prints out a trace of all the system calls made by a process. Strace is an open source project that runs on the linux kernel and it is included in most linux distributions. There is no need to alter or recompile the program being traced, so interaction with the programmer can be kept at a minimal.

When most shells are executing a command they create a sub process where the pipes in and out of the program are set up before the *execve* system call is issued. Strace outputs all the information needed to determine what pipes goes in and out of a command and what files are opened by the command.

The effect of each system in the strace log are dependent on the context the call was issued. This means it is not possible to determine what a script does by looking at each line of the strace log individually. If we want to extract the information from the strace log we need a parser for the log that keeps state of the pipes and files that are open.

We are also able to determine what files are opened by the process after the system call to execute the executable has been issued. Therefore we are also able to record all I/O done by custom made programs.

By doing passive monitoring we are not able to insert our own commands to collect information in the context of the command we are monitoring at the time of execution. For example by using this approach by itself we are not able to determine if the files accessed by the a command existed before the command was invoked, because we only have after the fact information about the execution.

4.2.4 Interpreting the code

Interpreting the code would not be hard since bash is open source and we would be able to utilize a fair amount of the code from the original project. Flow control in bash is however usually determined by the output from other executables. We don't necessarily support interpretation of these executables, and we would need to execute them to learn the exit status to determine the flow of the program. The exit status might depend on external elements and there is no guarantee that the exit status is going to be the same every time we run an executable, therefore it is impossible to determine the exact execution path of a shell script. To have our program execute arbitrary executables is anyway not an option since we might make undesirable changes to files or the environment.

If we change an exciting shell to extract the metadata and linage from scripts executed, we might need to update the shell when a new version of the shell is released. This is something we wish to avoid since the resources might not be in place to support development on this project in the future.

	instructing	overriding	monitoring
IDL	Possible - low number of function of interest	Not possible to override core functions	No support in IDL, outside monitoring possible
Bash	Complex syntax makes it really hard to instruct	Easy - changing the *PATH* variable	Easily done by tracing system calls

Table 4.1: Applicability of the different techniques

4.2.5 Parallels

In concept bash differs a lot from IDL. Bash is a command language interpreter that invokes executables, and IDL on the other hand is a language for application development. The difference is that most commands in bash spawns a subprocess for another application, in IDL only a couple of special commands executes applications.

At first the difference might not seem significant, but when studying the different languages up close the difference become more apparent. Since bash is mostly used to invoke executables, the command names are dependent on the applications that exists on the system it runs on. For IDL the commands are pretty much identical for all platforms. The biggest difference is probably that in bash we are not as much interested in the processing done by the bash interpreter, as what the executables invoked by bash does. In IDL we are mainly interested what is done by the interpreter.

Because of the different properties of the languages I chose to use different approaches for the two. As shown in table 4.1, it is possible to determine the linage by looking at the system calls made by the IDL interpreter. By doing it this way you would loose valuable information that we are able to get from the IDL environment when we instruct the code. The preprocessing technique is not ideal for bash since we are not so much interested in what commands are issued, as what the command do.

Chapter 5

Implementation

5.1 Plugins

The plugins is the part of the system responsible for catching the information we are interested in from the users process. The plugins are typically language specific.

The plugins do not write the information retrieved from the users process to any file but passes the information on to the rest of the system through the logging module. It is not necessary for plugins to keep any state since the logging modules is accessed statically.

Each directory containing source code that is being traced has a subdirectory called .es3. The plugins can store information they need in assigned directories under the .es3 directory.

The plugins can be turned on and off by inserting and removing the name of the programming language used in the *ES3_ENABLE* environment variable. The *ES3_ENABLE* is a colon separated list of active plugins. The plugins will not interact with the processes if the plugin is not turned on. It is up to the plugin itself to determine if it is active or not, and make a decision what to do. In most cases the session might have to be started over again if you want to enable or disable plugins during execution.

5.1.1 IDL plugin

The IDL plugin uses preprocessing of the users scripts to retrieve the linage and metadata from the process. In IDL only a small number of routines available to that will initiate I/O operations or interact with the environment. This allows us to collect all the linage information by only intercepting calls to these routines. The preprocessor can be invoked manually by the user, or automatically whenever a script is executed in IDL with ES^3 enabled.

Preprocessor

Since IDL does not support overriding routines I use a preprocessor to change the routine calls in the users scripts. The routines calls are renamed so when executing the code wrapper routines are invoked instead of the original targets. The wrapper routine calls the logging module which makes a record of the execution before the corresponding IDL routine is called.

The wrapper routines can be created by using the ”-w” option to the es3_idlprec executable. The routines need only to be created once when installing ES^3 on a system. The wrapper routines need to be included in the IDL searchpath before executing scripts that are preprocessed. If desired the wrappers can be stored in one directory and included in the default path by a systems administrator, but no administrator access is required to run

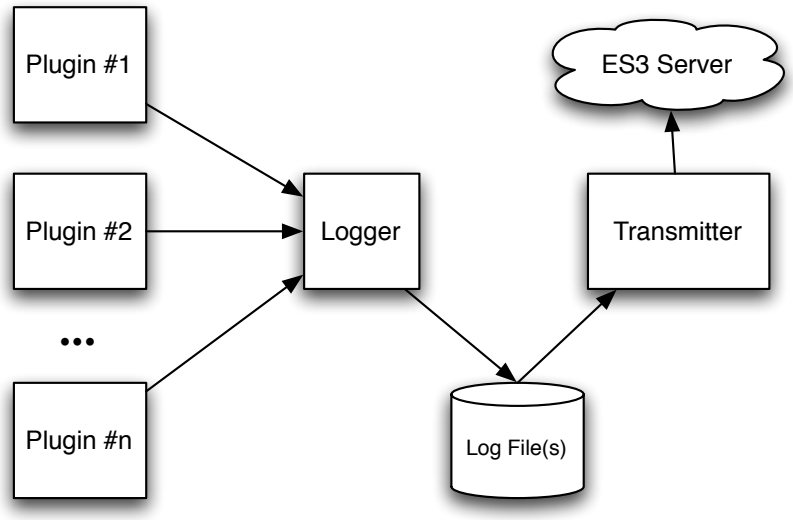

Figure 5.1: The ES^3 client architecture

the plugin. The names of the routines for this plugin all begin with "es3_", and will not interfere executions by users that does not have ES^3 enabled.

The logging module is wrapped by the procedure *ES3_LOGGER*. This procedure parses it's arguments, escapes the strings and invokes the es3_logger executable. The procedure was created to control the communication with the log module, and to simplify the wrapper routines.

The preprocessor is a naive stream editor, it does not do full lexical analysis of the scripts. It is probably possible to write a malicious script that uses a special syntax that will make the preprocessor break the script. This could be a problem if the IDL code contains strings with IDL code, and the strings continues over several lines. This is however unlikely to be a problem in real scripts, and can be avoided by making the preprocessor understand statements continuing over more than one line.

The preprocessor will ignore everything in a script following a line starting with the words *";ES3: IGNORE"*. The keyword is used to prevent the preprocessor to accidentally run on the wrapper scripts. A user can use this control word to exclude parts of his program. Improvements to the preprocessor will include adding more statements to control the behavior of the preprocessor.

The preprocessor will insert log statements at the beginning and end of each block and routine written by the user. The logger uses the term box to describe these borders. The use of boxes will allow us to keep track of the execution path of the script.

Automatic preprocessing

A user can explicitly preprocess the scripts, or the system can do it automatically. When a system has the ES^3 client installed and configured, the user can enable ES^3 tracking by adding "idl" to the environment variable *ES3_ENABLE* and crate the subfolders *.es3/idl/* in the folders containing source code ES^3 is going to instruct. If the user wants to disable ES^3, all that needs to be done is to remove "idl" from the *ES3_ENABLE* environment variable. If the variable *ES3_ENABLE* is not set or does not contain the string "idl" when you start IDL, ES^3 will not have any effect on the session.

When ES^3 is installed the IDL variable *!PATH* is set to include the wrappers ES^3 in-

Usage: es3_idlprec [options] [input-file]
If no input file is given the preprocessor will read from std-in.

Options are:

-h	Display a short help text and exit.
-o <output-file>	Place the output into <file>. If not output file is given the output is directed to std-out.
-d	Create a dummy procedure with the same name as the output file. When this option is set all the procedures defined in the file can be loaded into the IDL environment by using *RESOLVE_ROUTINE* and not having to use *.COMPILE*. *RESOLVE_ROUTINE* can be called from scripts and procedures, but *.COMPILE* only works in interactive mode.
-w <output-dir>	Create the wrapper routines .pro files in <output-dir> and exit. These wrapper routines needs to be in the IDL searchpath when preprocessed scripts are ran.
-l	Output a list all the routines in the input.
-b	Insert logging statements at the start of end of each function, procedure and block.
-p	Output only the difference between the original script and the preprocessed. This is used for debugging, and can be a way of ensuring the correct behavior of the preprocessor.

Table 5.1: Usage for the IDL preprocessor

Usage: es3_idlwrap [options] [directory]
If no directory is given, the current directory is wrapped.

Options are:

-h, –help	Display a small usage text
-c, –create	Create the .es3/idl subdirectorytree if it does not exist

Table 5.2: Usage for the IDL wrapper

ternal routines, and the procedure *ES3_STARTUP* is called at startup. If ES^3 is enabled for the session the startup procedure searches the *!PATH* variable for a directory containing the subdirectorytree .es3/idl/. When such a directory is found the startup script calls the *es3_idlwrap* executable and adds the .es3/idl/ directory to the *!PATH* variable ahead of the directory. The *es3_idlwrap* preprocesses all the files ending with ".pro" in the directory, and puts the processed files in the .es3/idl/ directory. It does not make any changes to the original file. This way, if a routine is called, the preprocessed version is used because it occurs before the original version in the searchpath.

If the subdirectorytree .es3/idl/ exists under the current directory it has to be treated as a special case since IDL will load any routine from files in the current directory before looking though the searchpath. The way this is solved is by running the *es3_idlwrap* executable on the current directory before loading all the routines outputted by *es3_idlwrap* into memory using the *RESOLVE_ROUTINE* procedure. *.COMPILE* is the command normally used to load routines into memory, but it cannot be called from within a script. Therefore we preprocess the files with the "-d" flag set. This will add a dummy routine to the preprocessed file which will allow for it to be loaded into memory by *RESOLVE_ROUTINE*.

The automatic preprocessing can be broken if the user changes the current directory or the *!PATH* variable after startup. This is why we cannot support these actions.

5.1.2 System call plugin

System calls are invocations of kernel routines issued by executables. To be able to access the system call we use strace. Strace is a program that comes with most linux distributions, and prints out a trace of all the system calls made by a another process. A program does not need to be recompiled before it can be traced, this allows us to use strace on binaries we don't have the source code for.

The system call plugin starts up strace in the background, and the output is piped into a system call postprocessor I've called es3_straceproc. Strace is set to attach to subprocesses of the process being traced, and the output is made more verbose to include all information needed to keep state of the processes traced. The postprocessor receives the system calls made, and makes the appropriate calls to the logger.

To improve performance es3_straceproc does not trace all system calls, but only the subset needed to keep sufficient state of the process. I've chosen not to trace all read and writes to a file, since this would reduce performance. I am making the assumption that if the process opens a file for reading, it reads from the file and if opens a file for writing, it writes to the file. A description of the system calls interpreted in can be found in table 5.4.

The *open* system call is not only issued when a data file is opened by the program, but for all files used by the program. If the process uses shared object, the shared object files are going to show up in the list of files used by the process. The client is going to submit all files referenced by the process back to the core, including any shared libraries. It is up to the server to determine if files should be excluded from the list.

The system call plugin is intended to use with bash scripts but there is no reason why it cannot be made more general and work with other programmatic languages. It is however designed with bash in mind, and might not work well with other languages.

The es3_sraceproc executable

The es3_straceproc executable is designed to read the output from strace on std-in and call the logger with the collected information.

The system calls that are outputted from strace could be broken over several lines if the call is put on hold. When the postprocessor receives a system call that is not finished, it stores it until the call is finished. The action done by the postprocessor when a system call is trapped is shown in table 5.3. The postprocessor stores information about the different processes in a hash table indexed by the process identifier.

Since both the logger and the strace postprocessor is written in perl, the source code of the logger is included in the postprocessor to improve performance. If the postprocessor is called as a standalone executable there is an overhead of a couple of seconds because the logger source code needs to be interpreted for each call. Further work would be to implement the logger with less overhead. When this is done the postprocessor can execute the logging executable without this delay.

5.2 Logger

5.2.1 The logger executable

The logger is a central part of the client. It connects the plugins and the transmitter together. All information that the plugins wishes to pass on the rest of the system needs to go though the logger. Having all the plugins go though the logger makes the system more flexible and easier to expand. If someone wants to write a new plugin, the interface to the rest of the system is defined by the interface of es3_log. There is no need for a author of a plugin to learn the details about the internal log format.

Having the logger a central part of the system we can put common functionality in the logger instead of having to implement the same functionality in each of the plugins. For

open	The filename and file descriptor is stored under the process identifier that opened the file
dup2	The filename or the pipe identifier of that the file descriptor points to is copied to the new file descriptor. If no matching file handle is found on the issuing process the closes ancestor with a matching file descriptor is used.
execve	The name of the executable, the arguments and the environment is stored under the process identifier issuing the system call. The open files for that process is also cleared since we only want to know about the files opened after execution, and not any library read prior to the execution of the command.
pipe	both the input and output end are registered as special file descriptors under the process issuing the system call.
clone	a new process is created in the hash and the parent process identifier is registered.
exit_group	The information about the process is sent off to the ES^3 client logging facillity.

Table 5.3: Actions made by the system trace postprocessor

open	opens a file and returns a file descriptor, a reference to the file in the table of open files for the current process.
dup2	duplicate an existing file descriptor
execve	loads a program from an ordinary, executable file into the current process, replacing the current program.
pipe	creates a pair of file descriptors, pointing to a pipe inode, and places them in the array pointed to by the argument
clone	creates a new process and continues execution, just like fork.
exit_group	the process exits the group.

Table 5.4: System calls interpreted by the system call plugin

example the plugins does not need to specify the process identifier if the process calling es3_log is the process being monitored. The logger will determine the process identifier of the caller and use it if an identifier is not specified.

The client log files are where all the information is stored before it is sent of to the ES^3 core. There is one log file for each session, and the log files are identified by the process identifier and the time it started. The logger is the only part of the system that is allowed to write to the client log files, and it is up to the logger to pick the right log file to write to. By haveing only the logger write to the log files it is easier to ensure that valid syntax is used in the log files. If this functionality was a part of the plugins it would be harder to keep bugs in the code from making invalid log files.

Due to a bug in the linux command process status (ps) the start time of a process is not reported consistently. The start time of a process might differ with a second from each time ps is ran, even if it is the same process being queried. I assume this is ether because ps is not able to detect the exact time, or because of a floating point error in the some implementations. The Berkeley Distribution found in Mac OS X does not seem to have this bug.

Since the process startup time is used in the filename of the log, by not taking the process status bug into consideration a log could be split up into more than one file. To work around this problem I've made the logger use logs with the same process identifier and startup time only two seconds away. Since the startup time is determined internally in the logger this workaround is completely invisible to the caller of the logger.

The logger is completely stateless with the exception of the startup time workaround. The logger does not read anything from the log, and is not aware about previous runs of the logger. It is up to the plugins to run the logger with the init flag to record general system information before other entries are made to the log. This could easily be changed to be automatic in the future, but the plugins should provide some additional information when it initializes the log.

The usage for es3_log is shown in table 5.5.

5.2.2 Internal log file

The internal log of the ES^3 client is stored in different files inside the /tmp/es3 directory. Each process has its own log file identified by the process identifier and the time the process started. We cannot only use the process identifier because they are reused, and are therefore not unique. The combination of startup time of a process and the time it started is however unique for a particular computer.

The client log is XML based, table 5.6 shows the allowed elements. Most of the attributes are optional, and some does not need to be specified by the plugin but rather determined by es3_logger.

The format of the log is not completely valid XML since there are more than one root element. If the log is encapsulated in a XML-elemet, the document would be valid. This is infract done by the transmitter before parsing the file. Each root-element is appended to the log with a single run of the logger, and no state is kept by the logger between the root elements.

5.3 Transmitter

The transmitter translates the internal log of the client to a format the ES^3 core understands. There is no dependency between the processing chains and the transmitter. Therefore the processing workflows are not dependent on having a connection to a server to be able to run. In earlier systems this was not the case and it usually led to the system being turned off, and newer turned on again. The transmitter can be invoked at regular interval to transmit the workflows stored on the host to the core.

Usage: es3_log [options]
Options are:

-h,–help	Display a short usage text.
–pid <pid>	The process identifier.
–pid <pid>	The identifier of the parent process.
–lpid <pid>	The process identifier used in the log file name.
–enter <name>	Enter block or routine <name>.
–leave <name>	Leave block or routine <name>.
–routine <name>	Routine <name> is called.
–enviroment <enviroment>	The enviroment in a colon separated list of: [(type)] <name> = "<value>"
–arguments	The arguments in a colon seperated list.
–time	The time the event occurred in the format yyyymmddTHHMMSSZ
–stime	The time the process started in the format yyyymmddTHHMMSSZ
–files	A colon separated list of files on the form: <filename> [, option1][, option2]... Options for files are: READ indicates that the function reads from the file WRITE indicated that the function writes to the file PIPE indicated that the file is not a regular file but a pipe
–language	The programming language used.
–init	Make and an entry containing system information in the log. The plugins should call es3_log with this option when it starts up a new session.
–print	Instead of writing the entry to the internal log files the entry gets printed to std-out. This option is used for debugging.

Table 5.5: Usage for the logger

Element name	Attraibutes	Children
init	**time** time the event was logged **pid** process identifier of the process being logged **stime** time the process started pstime time the parent process started ppid parent process id **language** programming language used user user is the program running under hostname host the process ran on	environment mount-points
exex	**time** time of execution **routine** name of the routine executed pid process id ppid parent process id	arguments enviroment io
enter	**region** name of the region entered	
leave	**region** name of the region left	
mount-points		mount*
enviroment		variable*
arguments		arument*
io		pipe* file*
mount	**share** path of the share mounted type what filesystem is used	**mount point** [Sting]
variable	**name** variable name type type of variable **value** value of variable	
argument		**argument's value** [Sting]
pipe	read process read from this pipe write process wrote to this pipe **id** identifier which is unique for the scope of the document	
file	read process read from this file write process wrote to this file	**file's path** [Sting]

Required attributes are in bold typeset
An asterisks denotes the element can occur zero or more times
Children are Elements except then other is defined by a type enclosed in brackets

Table 5.6: Tags used in the internal log

The output from the transmitter is a message to the ES^3 core. In a ES^3 message the data opbjects are represented as a file elements, the programs are represented as transformation elements and the elements are connected together with relation elements. The ES^3 message format is a work in progress, and it's highly likely to change from the format that is used in this paper. The part of the transmitter that transforms the objects into a language the ES^3 core understands is located inside a single method. This makes it easy to change the message format without doing any major changes to the transmitter.

The transmitter is written in java and uses Sun's API to parse the XML logs. The logs are not completely valid XML, they are missing a tag that encapsulates the entire log. This tag is added before the log is passed to the XML parser. When the parsing is done, objects are created that correspond to the different elements of the log.

When the logs are parsed the transmitter determines and denotes any relations between the workflows. This relations are determined by comparing the process identifiers. After the workflows are processed they are transmitted to the core. Since the ES^3 core is at this point not ready to receive any requests, the current implementation of the client does not transmit the messages, but writes the them to a file.

This page is intentionally left blank.

Chapter 6

Results

6.1 Achieving the goals

Our goals where to make a model for acquiring lineage from arbitrary earth science computational workflows, and implement the model for IDL and bash. The goals for the implementation was for it to:

- Require minimal interaction with the user after it is set up.
- Be easy to install and configure.
- Does not require administrative privileges to install or run
- Have a way to be reliable enabled and disabled.
- Be resilient to updates of software and configuration on the system it runs.

I've implemented the model for bash and IDL that automatically retrieves linage from processes and it does not require any interaction from the user to create a new or run a workflow. To demonstrate this I'm including the results of two example workflows, one written in IDL and one in bash. The client is easy to install, installation descriptions fits on less than a page, and does not require administrative privileges to install. The system is disabled if the environment variable *ES3_ENABLE* is not set when starting a session. The implementation is not optimized for performance since this was not one of our goals. Performance can be improved by making the logger a binary executable, ether by compiling the perl code or implementing it in a programming language with less overhead.

6.1.1 Resiliency to changes

It is important that the way metadata is retrieved is resilient to future changes in the software and the configuration of the host. We are hoping the client could be used with future releases of software it interacts with. If this is true the code does not need to be constantly updated, and the user does not have to wait for the latest upgrade of the client before he upgrades other parts of the system.

It is hard to determine if the client will run with future releases of dependent software since we cannot be sure of what changes are going to be made the future. The best we can do is to determine if it is likely for the parts of the software the plugins depend on to change in the future.

The bash plugin depends on the pattern of system calls issued by bash when it executes a command and the format of the output of strace. The way bash execute commands has not changed since bash first was released, and it is unlikely that this will change in the future. It might be that the system calls issued by bash differs from other operating system to another,

but it will only be a one time change to the plugin to make it run under a those operating systems. The current version is tested on Red Hat, Fedora and Gentoo. It is possible that the output of strace is going to change slightly. There is no note in the changelog of a change in output-format for the past 10 years. I don't think it is likely that we are going to see any changes that will effect the plugin any day soon.

The IDL plugin relies on the syntax used in the IDL scripts. The syntax of the IDL language is highly unlikely to change since it has been around a long time. However since the preprocessor used in the IDL plugin is a stream editor and does not do a complete lexical analysis of the script it is possible if the user uses a special syntax the preprocessor might not behave as intended. I can imagine there is a combination of quotation marks, comments and line breaks that will break the preprocessor. These error does not occur because the syntax of the language changes, but rather because the users uses an obscure syntax.

6.1.2 Examples

I have included two example workflows in this paper, one for bash and one for IDL. The source code and comments can be found in appendix E and D. The examples show that the system is able to retrieve the linage from these scripts without manually changing them in any way.

Bibliography

[1] Alexandria digital library project. http://www.alexandria.ucsb.edu/.

[2] Albert D. Alexandrov, Maximilian Ibel, Klaus E. Schauser, and Chris J. Scheiman. Ufo: a personal global file system based on user-level extensions to the operating system. *ACM Trans. Comput. Syst.*, 16(3):207–233, 1998.

[3] James Frew and Rajendra Bose. Earth system science workbench: A data management infrastructure for earth science products. In *SSDBM*, pages 180–189. IEEE Computer Society, 2001.

[4] Amin Vahdat and Tom Anderson. Transparent result caching. Technical report, Berkeley, CA, USA, 1997.

This page is intentionally left blank.

Appendix A

Abbreviations and Definitions

A.1 List of Definitions

Linage The history of sources and algorithms used to generate a piece of data. This data can be represented as a directed acyclic graph (DAG)

Metadata Data describing data. An example is a library catalog card, which contains data about the nature and location of a book: It is data about the data in the book referred to by the card.

(ADL) Middleware The digital library software developed by the Alexandria Digital Library project

Overriding (subroutines) Replacing the original subroutine with an alternative. In object oriented programming, is a language feature that allows a subclass to provide a specific implementation of a method that is already provided by one of its superclasses. The implementation in the subclass overrides (replaces) the implementation in the superclass.

Procedure A subroutine that does not return a value

Routine see: Subroutine

Subroutine A sequence of code which performs a specific task, as part of a larger program, and is grouped as one or more statement blocks. Subroutines can be ”called”, thus allowing programs to access the subroutine repeatedly without the subroutine’s code having been written more than once. A subroutine can ether be a function or a procedure.

Function A subroutine that returns a value.

A.2 List of Abbreviations

ADL Alexandria Digital Library - A a distributed digital library with collections of georeferenced materials.

BASH Bourne Again Shell

DAG Directed Acyclic Graph

EIL Environment Information Lab

ES^3 Earth System Science Server

ESSW Earth System Science Workbench

HTML Hypertext Markup Language - The document format language used on the World Wide Web

HTTP HyperText Transfer Protocol - A protocol used to transmit files over the World Wide Web

IDL Interactive Data Language

TREC Transparent result caching

UFO A Personal Global File System Based on User-Level Extensions to the Operating System

UCSB University of California, Santa Barabra

XML Extended Markup Language - Cleartext language for specifying structured data

Appendix B

Source Code

B.1 IDL plugin scripts

B.1.1 es3_idlprec

```
#!/usr/bin/perl
use Getopt::Std;

# the routines are represented as an array of four text stings:
# 0: the name of the routine
# 1: parametters and optional prametters
# 2: options - boolean options are marked with an leading /
# 3: arguments passed to ES3_LOG
#  (FILE=FILE and ROUTINE=<name> are implicit)
$ENV{'ES3_IGNORE'} = "es3_idlprec";
@functions = (
  ["READ_ASCII", "FILE", "COMMENT_SYMBOL, COUNT, DATA_START,
      DELIMITER, HEADER, MISSING_VALUE, NUM_RECORDS,
      RECORD_START, TEMPLATE, /VERBOSE", "/READ"],
  ["READ_BINARY", "FILE", "TEMPLATE, DATA_START, DATA_TYPE,
    DATA_DIMS, ENDIAN", "/READ"],
  ["READ_BMP", "FILE, RED, GREEN, BLUE, IHDR", "RGB", "/READ"],
  ["READ_DICOM", "FILE, RED, GREEN, BLUE", "IMAGE_INDEX",
    "/READ"],
  ["READ_IMAGE", "FILE, RED, GREEN, BLUE", "IMAGE_INDEX",
    "/READ"],
  ["READ_MRSID", "FILE", "LEVEL, SUB_RECT", "/READ"],
  ["READ_PNG", "FILE, RED, GREEN, BLUE", "/ORDER, /VERBOSE,
    /TRANSPARENT", "/READ"],
  ["READ_SPR", "FILE", "", "/READ"],
  ["READ_SYLK", "FILE", "/ARRAY, /COLMAJOR, NCOLS, NROWS,
    STARTCOL, STARTROW, /USEDOUBLES, /USELONGS", "/READ"],
  ["READ_TIFF", "FILE, RED, GREEN, BLUE",
    "CHANNELS, GEOTIFF, IMAGE_INDEX, INTERLEAVE,
      ORIENTATION, PLANARCONFIG, SUB_RECT, /UNSIGNED, /VERBOSE",
    "/READ"],
  ["READ_WAV", "FILE, RATE", "", "/READ"],
  ["READ_WAVE", "FILE, VARIABLES, NAMES, DIMENSIONS",
    "MESHNAMES", "/READ"],
  ["READ_XWD", "FILE, RED, GREEN, BLUE", "", "/READ"]
);

@procedures = (
  ["READ_JPEG", "FILE, IMAGE, COLORTABLE",
          "UNIT, BUFFER, COLORS, DITHER, /GRAYSCALE, /ORDER,
          TRUE, /TWO_PASS_QUANTIZE", "/READ"],
  ["READ_INTERFILE", "FILE, DATA", "", "/READ"],
  ["READ_PICT", "FILE, IMAGE, RED, GREEN, BLUE", "", "/READ"],
```

```
  ["READ_PPM", "FILE, IMAGE", "MAXVAL", "/READ"],
  ["READ_SRF", "FILE, IMAGE, RED, GREEN, BLUE", "", "/READ"],
  ["WRITE_BMP", "FILE, IMAGE, RED, GREEN, BLUE",
    "FOUR_BIT, Ihd, HEADER_DEFINE, RGB", "/WRITE"],
  ["WRITE_IMAGE", "FILE, FORMAT, DATA, RED, GREEN, BLUE",
    "APPEND, _EXTRA", "/WRITE"],
  ["WRITE_JPEG", "FILE, IMAGE", "UNIT, /ORDER,
    /PROGRESSIVE, QUALITY, TRUE", "/WRITE"],
  ["WRITE_NRIF", "FILE, IMAGE, RED, GREEN, BLUE", "", "/WRITE"],
  ["WRITE_PICT", "FILE, IMAGE, RED, GREEN, BLUE", "", "/WRITE"],
  ["WRITE_PNG", "FILE IMAGE, RED, GREEN, BLUE",
    "/VERBOSE, TRANSPARENT, /ORDER", "/WRITE"],
  ["WRITE_PPM", "FILE, IMAGE", "ASCII", "/WRITE"],
  ["WRITE_SRF", "FILE, IMAGE, RED, GREEN, BLUE",
    "WRITE_32, ORDER", "/WRITE"],
  ["WRITE_SYLK", "FILE, DATA", "STARTROW, STARTCOL", "/WRITE"],
  ["WRITE_TIFF", "FILE, IMAGE",
    "/APPEND, BITS_PER_SAMPLE, RED, GREEN, BLUE,
      COMPRESSION, GEOTIFF, /LONG, /SHORT, /FLOAT,
      ORIENTATION, PLANARCONFIG, /VERBOSE, XRESOL, YRESOL",
    "/WRITE"],
  ["WRITE_WAV", "FILE, DATA, RATE", "", "/WRITE"],
  ["WRITE_WAVE", "FILE, DATE", "BIN, NOMESHDEF, DATANAME,
    MESHNAME, VECTOR", "/WRITE"],
  ["READ_X11_BITMAP", "FILE, BITMAP, X, Y", "/EXPAND_TO_BYTES",
    "/READ"],
  ["OPENW", "UNIT, FILE",
    "/APPEND, /COMPRESS, BUFSIZE, /DELETE, ERROR,
      /F77_UNFORMATTED, /GET_LUN, /MORE, /NOEXPAND_PATH,
      /STDIO, /SWAP_ENDIAN, SWAP_IF_BIG_ENDIAN,
      /SWAP_IF_LITTLE_ENDIAN, /VAX_FLOAT, WIDTH, /XDR,
      /RAWIO", "/WRITE"],
  ["OPENR", "UNIT, FILE",
    "/APPEND, /COMPRESS, BUFSIZE, /DELETE, ERROR,
      /F77_UNFORMATTED, /GET_LUN, /MORE, /NOEXPAND_PATH,
      /STDIO, /SWAP_ENDIAN, SWAP_IF_BIG_ENDIAN,
      /SWAP_IF_LITTLE_ENDIAN, /VAX_FLOAT, WIDTH, /XDR,
      /RAWIO", "/READ"]
);


@routines = (@functions, @procedures);

#print the helptext
sub print_help{
  print STDERR "Usage: es3_idlprec [options] [input-file]
Options are:
  -h                    Display this usage text
  -o <output-file>      Place the output into <file>
  -d                    Create a dummy procedure with the " .
    "same name as the output file
  -w <output-dir>       Create the wrapper routines .pro files ".
    "in <output-dir> and exit
  -l                    List all the routines in the script
  -b                    Box it up
  -p                    Output only the difference
Example 1:
  % echo \"PRINT, 'HELLO WORLD'\" | es3_idlprec | idl

Example 2:
  % es3_idlprec -o es3_test.pro test.pro
  % idl
  IDL> .COMPILE es3_test.pro

Example 3:
  IDL> ES3_COMPILE test
  IDL> TEST, 'foo'
```

```
";
exit;
}


%options=();
getopts("ho:dw:lbp",\%options);

print_help if defined $options{h};

#create wrapper routines
if (defined $options{w}) {
  $dir = $options{w};
  $dir .= "/" if not ($dir =~ /\/$/);
  die("Output directory not found: '".$dir."'")
    if not(-e  $dir);

  for $i (0 ... $#routines) {
    $filename = $dir."es3_".lc($routines[$i][0]).".pro";
    open(file,">".$filename);
    $function = ($i <  @functions);
    print file ";ES3: IGNORE\n";
    print file $function?"FUNCTION":"PRO" ;
    print file " ES3_" . $routines[$i][0] . ", ";
    print file $routines[$i][1];
    @options = split(/,[ \t\n]*/, $routines[$i][2]);
    for $j (0 ... $#options) {
      $opt = $options[$j];
      $opt =~ s/^\///;
      print file ", ";
      print file "\$\n\t\t" if (($j+1)%1==0);
      print file $opt . "=" . $opt;
    }
    print file "\n\tCOMPILE_OPT HIDDEN\n";
    print file "\tES3_LOG, ROUTINE='" . $routines[$i][0] . "',";
    print file " FILE=FILE, ";
    print file $routines[$i][3] . "\n";
    if($function) {
      print file "\tRETURN, " . $routines[$i][0] . "(";
    } else {
      print file  "\t" . $routines[$i][0] . ", ";
    }
    print file  $routines[$i][1];
    for $j (0 ... $#options) {
      $opt = $options[$j];
      $opt = $opt . "=" . $opt
        if not $opt =~ s/\/(.*)/$1=KEYWORD_SET($1)/;
      print file ", ";
      print file "\$\n\t\t" if (($j+1)%1==0);
      print file $opt;
    }
    print file ")" if($function);
    print file "\nEND\n";
    close(file);
  }
  exit;
}

#set output stream
if (defined $options{o}) {
  $output_name = $options{o};
  $output_name =~ s/\.pro//i;
  $output_name =~ s/.*\///i;
  open(output,">".$options{o});
} else {
  open(output,'>-');
```

```
}

#set input stream
if ($ARGV[0]) {
  open(input, $ARGV[0])
} else {
  open(input, '-');
}

#make the dummy process
if(defined $options{d} and defined $output_name) {
  print output "PRO " . $output_name;
  print output "\n\tCOMPILE_OPT hidden\nEND\n\n";
}

#do the parsing

@blocks; # list of blocks/routine we have entered
sub enter{
  $block = $_[0];
  push(@blocks, $block);
  if (defined $options{b} and not $block =~ /<begin>/) {
    while($line =~ /\$[\t ]*(;.*)?$/ ){
#    while($line =~ /\$[\t ]*$/ ){
      $line .= <input>;
    }
    $line .= "HELP, NAMES=\"*\", OUTPUT=ES3_ENVIROMENT & ";
    $line .=  "ES3_LOG, ENTER=\"" . $block;
    $line .= "\", ENVIROMENT=ES3_ENVIROMENT\n";
  }
}
while ($line = <input>) {
  $original = $line;
  if($line =~ /^[ \t]*;[ \t]*ES3:[ \t]*(.*)$/i){
    $command = $1;
    if ($command =~ /ignore/i) {
      print output $line;
      while ($line = <input>) {
        print output $line;
      }
      break;
    }
  }
  next if($line =~ /[\t ]*HELP,.*ES3_LOG/i);
  for $i (0 ... $#functions) {
    my $function = $functions[$i][0];
    $line =~ s/([^_])($function[ \t]*\()/$1ES3_$2/i;
  }
  for $i (0 ... $#procedures) {
    my $procedure = $procedures[$i][0];
    $line =~
      s/(^(.*&)?[\t ]*([^_])?)($procedure[ \t]*,)/$1ES3_$4/i;
  }
  if ($line =~ /^(pro|function)[ \t]+([a-zA-Z0-9_:]+)/i) {
    enter($2);
    print STDERR $2 . "\n" if (defined $options{l});
  }
  if ($line =~ /^[ \t]*BEGIN[ \t]*(;.*)?$/i){
    enter("<script>") if @blocks == 0;
    enter("<begin>");
  }
  enter("<script>") if (not $line =~ /^[ \t]*(;.*)?$/
    and @blocks == 0);

  if ($line =~ /^[ \t]*END[ \t]*(;.*)?$/i){
    $block = pop(@blocks);
    if (defined $options{b} and not $block =~ /<begin>/){
```

```
      $line = "HELP, NAMES=\"*\", OUTPUT=ES3_ENVIROMENT & ";
      $line .= "ES3_LOG, LEAVE=\"";
      $line .= $block."\", ENVIROMENT=ES3_ENVIROMENT\n";
      $line .= "END ; " . $block . "\n";
    }
  }

  if (defined $options{p}) {
    if(not $line eq $original) {
       print output " - " . $original;
       print output " + " . $line;
    }
  } else {
   print output $line;
  }
}

close(input);
close(output);
```

B.1.2 es3_idlwrap

```
#!/bin/bash
export ES3_IGNORE=es3_idlwrap

prec="es3_idlprec"
if ! which $prec &> /dev/null;
then
  echo "  Error: Could not find $prec in PATH";
  exit
fi

help() {
echo "Usage: es3_idlwrap [options] [directory]"
echo "Options are:"
echo "  -h,--help          Display this usage text"
echo -n "  -c,--create        Create .es3 and .es3/idl"
echo " directories if they do not exist"

}

create=false
dir=`pwd`
while [ -n "$1" ]
do
  case "$1"
  in
    -h|--help)
      help
      exit 0
      ;;
    -c)
      create=true
      ;;
    -*)
      echo "  Unknown option: " $1
      ;;
    *)
      dir=$1
      ;;
  esac
  shift
done

if [ ! -d "$dir" ]
```

```
then
  echo "  Could not find dir: '$dir'"
  exit
fi

cd "$dir"
if [ ! -d ".es3/idl" ]
then
  if [ "$create" = "true" ]
  then
    mkdir ".es3"
    mkdir ".es3/idl"
  else
    echo "  Error: ES3 is not enabled for dir: '$dir'"
    echo "  You need to create the directory `pwd`/.es3/idl"
    exit
  fi
fi

ls -1 *.pro | while read pro
do
  if newer "$pro" ".es3/idl/$pro"
  then
    name=`echo $pro| sed 's/\(.*\)\.pro/\1/'`
    if ! grep -i -e \
    "^[ \t]*\(PRO\|FUNCTION\)[ \t]*$name\([^a-z]\|$\)" \
    $pro &> /dev/null
    then
      copts="-d"
    fi
    $prec $copts -b -o .es3/idl/$pro $pro
  fi
done
```

B.1.3 es3_startup.pro

```
;ES3: IGNORE
PRO ES3_STARTUP
  COMPILE_OPT hidden
  QUIET = !QUIET
  !QUIET=1
  IF STRMATCH(":"+GETENV("ES3_ENABLE")+":", "*:idl:*") THEN BEGIN
    PATH_DIRS=STRSPLIT(!PATH, ":", /EXTRACT)
    !PATH=""
    FOR I = 0, N_ELEMENTS(PATH_DIRS)-1 DO BEGIN
      ES3_DIR = PATH_DIRS[I] + "/.es3/idl/"
      IF (FILE_TEST(ES3_DIR, /DIRECTORY)) THEN BEGIN
        SPAWN, "ES3_IGNORE=idl_startup es3_idlwrap '" + PATH_DIRS[I]+"'"
        !PATH = !PATH + ":" + ES3_DIR
      ENDIF
      !PATH = !PATH + ":" + PATH_DIRS[I]
    ENDFOR
    IF (FILE_TEST(".es3/idl/", /DIRECTORY)) THEN BEGIN
      PRINT,"% Loading modules from current directory..."
      SPAWN, "ES3_IGNORE=idl_startup es3_idlwrap"
      CD, ".es3/idl/"
      CURRENT=FILE_SEARCH("*.pro")
      FOR I = 0, N_ELEMENTS(CURRENT)-1 DO BEGIN
        ROUTINE=STRMID(CURRENT[I],0, $
          STRPOS(CURRENT[I],".",/REVERSE_SEARCH))
        RESOLVE_ROUTINE, ROUTINE, /EITHER
      ENDFOR
      CD, "../.."
    ENDIF
    ES3_LOG, /INIT
```

```
    QUIET = !QUIET
    PRINT,"% ES3 snooping is enabled for this session"
  ENDIF
  !QUIET=0

END
```

B.1.4 es3_compile.pro

```
PRO ES3_COMPILE, MODULE
  FILE=file_search(MODULE, /FULLY_QUALIFY_PATH)
  IF (FILE) THEN BEGIN
    SPAWN, 'ES3_IGNORE=es3_compile idlprec.pl -d -o es3_tmp.pro -b '+FILE
    RESOLVE_ROUTINE, 'ES3_TMP', /COMPILE_FULL_FILE, /EITHER
    SPAWN, 'ES3_IGNORE=es3_compile rm es3_tmp.pro'
    PRINT,"% ES3 hooked module: " + MODULE
  ENDIF ELSE PRINT,"% Could not find module: " $
    STRUPCASE(MODULE)
END
```

B.1.5 es3_log.pro

```
;ES3: IGNORE

FUNCTION ES3_ESC, STRING
  RETURN=STRJOIN(STRSPLIT(STRING,"\", /EXTRACT, $
    /PRESERVE_NULL), "\\\\")
  RETURN=STRJOIN(STRSPLIT(RETURN,'"', /EXTRACT, $
    /PRESERVE_NULL), '\\\"')
  RETURN, RETURN
END

PRO ES3_LOG, ROUTINE=ROUTINE, FILE=FILE, READ=READ, $
  WRITE=WRITE, ENTER=ENTER, LEAVE=LEAVE, $
  ENVIROMENT=ENVIROMENT, INIT=INIT
  COMPILE_OPT HIDDEN
  ARG = ""
  IF KEYWORD_SET(FILE) THEN BEGIN
    ARG=ARG+"--files '"+FILE_SEARCH(FILE, /FULLY_QUALIFY_PATH)
    IF KEYWORD_SET(READ) THEN ARG=ARG+", READ"
    IF KEYWORD_SET(WRITE) THEN ARG=ARG+", WRITE "
      ARG = ARG+"' "
  ENDIF

  IF KEYWORD_SET(ROUTINE) THEN ARG=ARG+"--routine "+ROUTINE+" "
  IF KEYWORD_SET(ENTER) THEN ARG=ARG+"--enter "+ENTER+" "
  IF KEYWORD_SET(LEAVE) THEN ARG=ARG+"--leave "+LEAVE+" "
  IF KEYWORD_SET(ENVIROMENT) THEN BEGIN
    ARG += '--enviroment "'
    FOR I = 0, n_elements(ENVIROMENT)-1 DO BEGIN
      IF (NOT STRPOS(ENVIROMENT[I], "ES3_") EQ 0) THEN BEGIN
        SPLIT = STRSPLIT(ENVIROMENT[I], /EXTRACT)
        VALUE = STRTRIM(strmid(ENVIROMENT[I], $
          STRPOS(ENVIROMENT[I], "=")+2),1)
        IF STRPOS(VALUE, "'") EQ 0 THEN $
          VALUE = strmid(VALUE, 1, $
            STRPOS(VALUE, "'", /REVERSE_SEARCH)-1)
          ARG += "(" + SPLIT[1] + ") " + SPLIT[0] + "="
          ARG += '\"' + ES3_ESC(VALUE) + '\":'
        ENDIF
      ENDFOR
    ARG += '" '
    ENDIF
```

```
  IF KEYWORD_SET(INIT) THEN BEGIN
    SYSTEM = '!DIR=\"'+ES3_ESC(!DIR)+'\":'
    SYSTEM += '!PATH=\"'+ES3_ESC(!PATH)+'\":'
    SYSTEM += '!VERSION.ARCH=\"'+ES3_ESC(!VERSION.ARCH)+'\":'
    SYSTEM += '!VERSION.OS=\"'+ES3_ESC(!VERSION.OS)+'\":'
    SYSTEM += '!VERSION.OS_FAMILY=\"'
    SYSTEM +=   ES3_ESC(!VERSION.OS_FAMILY)+'\":'
    SYSTEM += '!VERSION.RELEASE=\"'
    SYSTEM +=   ES3_ESC(!VERSION.RELEASE)+'\":'
    SYSTEM += '!VERSION.BUILD_DATE=\"'
    SYSTEM +=   ES3_ESC(!VERSION.BUILD_DATE)+'\":'
    SYSTEM += '!VERSION.MEMORY_BITS=\"'
    SYSTEM +=   ES3_ESC(!VERSION.MEMORY_BITS)+'\":'
    SYSTEM += '!VERSION.FILE_OFFSET_BITS=\"'+ $
    ES3_ESC(!VERSION.FILE_OFFSET_BITS)+'\":'
    ARG='--init --language idl --enviroment "'+SYSTEM+'"'
    ENDIF
  SPAWN, 'ES3_IGNORE=idl_startup es3_log ' + ARG, OUT
  FOR I = 0, n_elements(OUT)-1 DO IF STRLEN(OUT[I]) NE 0 THEN $
    PRINT, OUT[I]
END
```

B.1.6 Wrapper scripts

es3_openr.pro

```
;ES3: IGNORE
PRO ES3_OPENR, UNIT, FILE, $
    APPEND=APPEND, $
    COMPRESS=COMPRESS, $
    BUFSIZE=BUFSIZE, $
    DELETE=DELETE, $
    ERROR=ERROR, $
    F77_UNFORMATTED=F77_UNFORMATTED, $
    GET_LUN=GET_LUN, $
    MORE=MORE, $
    NOEXPAND_PATH=NOEXPAND_PATH, $
    STDIO=STDIO, $
    SWAP_ENDIAN=SWAP_ENDIAN, $
    SWAP_IF_BIG_ENDIAN=SWAP_IF_BIG_ENDIAN, $
    SWAP_IF_LITTLE_ENDIAN=SWAP_IF_LITTLE_ENDIAN, $
    VAX_FLOAT=VAX_FLOAT, $
    WIDTH=WIDTH, $
    XDR=XDR, $
    RAWIO=RAWIO
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='OPENR', FILE=FILE, /READ
  OPENR, UNIT, FILE, $
    APPEND=KEYWORD_SET(APPEND), $
    COMPRESS=KEYWORD_SET(COMPRESS), $
    BUFSIZE=BUFSIZE, $
    DELETE=KEYWORD_SET(DELETE), $
    ERROR=ERROR, $
    F77_UNFORMATTED=KEYWORD_SET(F77_UNFORMATTED), $
    GET_LUN=KEYWORD_SET(GET_LUN), $
    MORE=KEYWORD_SET(MORE), $
    NOEXPAND_PATH=KEYWORD_SET(NOEXPAND_PATH), $
    STDIO=KEYWORD_SET(STDIO), $
    SWAP_ENDIAN=KEYWORD_SET(SWAP_ENDIAN), $
    SWAP_IF_BIG_ENDIAN=SWAP_IF_BIG_ENDIAN, $
    SWAP_IF_LITTLE_ENDIAN=KEYWORD_SET(SWAP_IF_LITTLE_ENDIAN), $
    VAX_FLOAT=KEYWORD_SET(VAX_FLOAT), $
    WIDTH=WIDTH, $
    XDR=KEYWORD_SET(XDR), $
    RAWIO=KEYWORD_SET(RAWIO)
```

```
END
```

es3_openw.pro

```
;ES3: IGNORE
PRO ES3_OPENW, UNIT, FILE, $
    APPEND=APPEND, $
    COMPRESS=COMPRESS, $
    BUFSIZE=BUFSIZE, $
    DELETE=DELETE, $
    ERROR=ERROR, $
    F77_UNFORMATTED=F77_UNFORMATTED, $
    GET_LUN=GET_LUN, $
    MORE=MORE, $
    NOEXPAND_PATH=NOEXPAND_PATH, $
    STDIO=STDIO, $
    SWAP_ENDIAN=SWAP_ENDIAN, $
    SWAP_IF_BIG_ENDIAN=SWAP_IF_BIG_ENDIAN, $
    SWAP_IF_LITTLE_ENDIAN=SWAP_IF_LITTLE_ENDIAN, $
    VAX_FLOAT=VAX_FLOAT, $
    WIDTH=WIDTH, $
    XDR=XDR, $
    RAWIO=RAWIO
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='OPENW', FILE=FILE, /WRITE
  OPENW, UNIT, FILE, $
    APPEND=KEYWORD_SET(APPEND), $
    COMPRESS=KEYWORD_SET(COMPRESS), $
    BUFSIZE=BUFSIZE, $
    DELETE=KEYWORD_SET(DELETE), $
    ERROR=ERROR, $
    F77_UNFORMATTED=KEYWORD_SET(F77_UNFORMATTED), $
    GET_LUN=KEYWORD_SET(GET_LUN), $
    MORE=KEYWORD_SET(MORE), $
    NOEXPAND_PATH=KEYWORD_SET(NOEXPAND_PATH), $
    STDIO=KEYWORD_SET(STDIO), $
    SWAP_ENDIAN=KEYWORD_SET(SWAP_ENDIAN), $
    SWAP_IF_BIG_ENDIAN=SWAP_IF_BIG_ENDIAN, $
    SWAP_IF_LITTLE_ENDIAN=KEYWORD_SET(SWAP_IF_LITTLE_ENDIAN), $
    VAX_FLOAT=KEYWORD_SET(VAX_FLOAT), $
    WIDTH=WIDTH, $
    XDR=KEYWORD_SET(XDR), $
    RAWIO=KEYWORD_SET(RAWIO)
END
```

es3_read_ascii.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_ASCII, FILE, $
    COMMENT_SYMBOL=COMMENT_SYMBOL, $
    COUNT=COUNT, $
    DATA_START=DATA_START, $
    DELIMITER=DELIMITER, $
    HEADER=HEADER, $
    MISSING_VALUE=MISSING_VALUE, $
    NUM_RECORDS=NUM_RECORDS, $
    RECORD_START=RECORD_START, $
    TEMPLATE=TEMPLATE, $
    VERBOSE=VERBOSE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_ASCII', FILE=FILE, /READ
  RETURN, READ_ASCII(FILE, $
    COMMENT_SYMBOL=COMMENT_SYMBOL, $
```

```
      COUNT=COUNT, $
      DATA_START=DATA_START, $
      DELIMITER=DELIMITER, $
      HEADER=HEADER, $
      MISSING_VALUE=MISSING_VALUE, $
      NUM_RECORDS=NUM_RECORDS, $
      RECORD_START=RECORD_START, $
      TEMPLATE=TEMPLATE, $
      VERBOSE=KEYWORD_SET(VERBOSE))
END
```

es3_read_binary.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_BINARY, FILE, $
      TEMPLATE=TEMPLATE, $
      DATA_START=DATA_START, $
      DATA_TYPE=DATA_TYPE, $
      DATA_DIMS=DATA_DIMS, $
      ENDIAN=ENDIAN
   COMPILE_OPT HIDDEN
   ES3_LOG, ROUTINE='READ_BINARY', FILE=FILE, /READ
   RETURN, READ_BINARY(FILE, $
      TEMPLATE=TEMPLATE, $
      DATA_START=DATA_START, $
      DATA_TYPE=DATA_TYPE, $
      DATA_DIMS=DATA_DIMS, $
      ENDIAN=ENDIAN)
END
```

es3_read_bmp.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_BMP, FILE, RED, GREEN, BLUE, IHDR, $
      RGB=RGB
   COMPILE_OPT HIDDEN
   ES3_LOG, ROUTINE='READ_BMP', FILE=FILE, /READ
   RETURN, READ_BMP(FILE, RED, GREEN, BLUE, IHDR, $
      RGB=RGB)
END
```

es3_read_dicom.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_DICOM, FILE, RED, GREEN, BLUE, $
      IMAGE_INDEX=IMAGE_INDEX
   COMPILE_OPT HIDDEN
   ES3_LOG, ROUTINE='READ_DICOM', FILE=FILE, /READ
   RETURN, READ_DICOM(FILE, RED, GREEN, BLUE, $
      IMAGE_INDEX=IMAGE_INDEX)
END
```

es3_read_image.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_IMAGE, FILE, RED, GREEN, BLUE, $
      IMAGE_INDEX=IMAGE_INDEX
   COMPILE_OPT HIDDEN
   ES3_LOG, ROUTINE='READ_IMAGE', FILE=FILE, /READ
```

```
  RETURN, READ_IMAGE(FILE, RED, GREEN, BLUE, $
    IMAGE_INDEX=IMAGE_INDEX)
END
```

es3_read_interfile.pro

```
;ES3: IGNORE
PRO ES3_READ_INTERFILE, FILE, DATA
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_INTERFILE', FILE=FILE, /READ
  READ_INTERFILE, FILE, DATA
END
```

es3_read_jpeg.pro

```
;ES3: IGNORE
PRO ES3_READ_JPEG, FILE, IMAGE, COLORTABLE, $
    UNIT=UNIT, $
    BUFFER=BUFFER, $
    COLORS=COLORS, $
    DITHER=DITHER, $
    GRAYSCALE=GRAYSCALE, $
    ORDER=ORDER, $
    TRUE=TRUE, $
    TWO_PASS_QUANTIZE=TWO_PASS_QUANTIZE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_JPEG', FILE=FILE, /READ
  READ_JPEG, FILE, IMAGE, COLORTABLE, $
    UNIT=UNIT, $
    BUFFER=BUFFER, $
    COLORS=COLORS, $
    DITHER=DITHER, $
    GRAYSCALE=KEYWORD_SET(GRAYSCALE), $
    ORDER=KEYWORD_SET(ORDER), $
    TRUE=TRUE, $
    TWO_PASS_QUANTIZE=KEYWORD_SET(TWO_PASS_QUANTIZE)
END
```

es3_read_mrsid.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_MRSID, FILE, $
    LEVEL=LEVEL, $
    SUB_RECT=SUB_RECT
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_MRSID', FILE=FILE, /READ
  RETURN, READ_MRSID(FILE, $
    LEVEL=LEVEL, $
    SUB_RECT=SUB_RECT)
END
```

es3_read_pict.pro

```
;ES3: IGNORE
PRO ES3_READ_PICT, FILE, IMAGE, RED, GREEN, BLUE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_PICT', FILE=FILE, /READ
  READ_PICT, FILE, IMAGE, RED, GREEN, BLUE
END
```

es3_read_png.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_PNG, FILE, RED, GREEN, BLUE, $
    ORDER=ORDER, $
    VERBOSE=VERBOSE, $
    TRANSPARENT=TRANSPARENT
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_PNG', FILE=FILE, /READ
  RETURN, READ_PNG(FILE, RED, GREEN, BLUE, $
    ORDER=KEYWORD_SET(ORDER), $
    VERBOSE=KEYWORD_SET(VERBOSE), $
    TRANSPARENT=KEYWORD_SET(TRANSPARENT))
END
```

es3_read_ppm.pro

```
;ES3: IGNORE
PRO ES3_READ_PPM, FILE, IMAGE, $
    MAXVAL=MAXVAL
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_PPM', FILE=FILE, /READ
  READ_PPM, FILE, IMAGE, $
    MAXVAL=MAXVAL
END
```

es3_read_spr.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_SPR, FILE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_SPR', FILE=FILE, /READ
  RETURN, READ_SPR(FILE)
END
```

es3_read_srf.pro

```
;ES3: IGNORE
PRO ES3_READ_SRF, FILE, IMAGE, RED, GREEN, BLUE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_SRF', FILE=FILE, /READ
  READ_SRF, FILE, IMAGE, RED, GREEN, BLUE
END
```

es3_read_sylk.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_SYLK, FILE, $
    ARRAY=ARRAY, $
    COLMAJOR=COLMAJOR, $
    NCOLS=NCOLS, $
    NROWS=NROWS, $
    STARTCOL=STARTCOL, $
    STARTROW=STARTROW, $
    USEDOUBLES=USEDOUBLES, $
    USELONGS=USELONGS
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_SYLK', FILE=FILE, /READ
```

```
  RETURN, READ_SYLK(FILE, $
    ARRAY=KEYWORD_SET(ARRAY), $
    COLMAJOR=KEYWORD_SET(COLMAJOR), $
    NCOLS=NCOLS, $
    NROWS=NROWS, $
    STARTCOL=STARTCOL, $
    STARTROW=STARTROW, $
    USEDOUBLES=KEYWORD_SET(USEDOUBLES), $
    USELONGS=KEYWORD_SET(USELONGS))
END
```

es3_read_tiff.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_TIFF, FILE, RED, GREEN, BLUE, $
    CHANNELS=CHANNELS, $
    GEOTIFF=GEOTIFF, $
    IMAGE_INDEX=IMAGE_INDEX, $
    INTERLEAVE=INTERLEAVE, $
    ORIENTATION=ORIENTATION, $
    PLANARCONFIG=PLANARCONFIG, $
    SUB_RECT=SUB_RECT, $
    UNSIGNED=UNSIGNED, $
    VERBOSE=VERBOSE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_TIFF', FILE=FILE, /READ
  RETURN, READ_TIFF(FILE, RED, GREEN, BLUE, $
    CHANNELS=CHANNELS, $
    GEOTIFF=GEOTIFF, $
    IMAGE_INDEX=IMAGE_INDEX, $
    INTERLEAVE=INTERLEAVE, $
    ORIENTATION=ORIENTATION, $
    PLANARCONFIG=PLANARCONFIG, $
    SUB_RECT=SUB_RECT, $
    UNSIGNED=KEYWORD_SET(UNSIGNED), $
    VERBOSE=KEYWORD_SET(VERBOSE))
END
```

es3_read_wav.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_WAV, FILE, RATE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_WAV', FILE=FILE, /READ
  RETURN, READ_WAV(FILE, RATE)
END
```

es3_read_wave.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_WAVE, FILE, VARIABLES, NAMES, DIMENSIONS, $
    MESHNAMES=MESHNAMES
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_WAVE', FILE=FILE, /READ
  RETURN, READ_WAVE(FILE, VARIABLES, NAMES, DIMENSIONS, $
    MESHNAMES=MESHNAMES)
END
```

es3_read_x11_bitmap.pro

```
;ES3: IGNORE
PRO ES3_READ_X11_BITMAP, FILE, BITMAP, X, Y, $
    EXPAND_TO_BYTES=EXPAND_TO_BYTES
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_X11_BITMAP', FILE=FILE, /READ
  READ_X11_BITMAP, FILE, BITMAP, X, Y, $
    EXPAND_TO_BYTES=KEYWORD_SET(EXPAND_TO_BYTES)
END
```

es3_read_xwd.pro

```
;ES3: IGNORE
FUNCTION ES3_READ_XWD, FILE, RED, GREEN, BLUE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='READ_XWD', FILE=FILE, /READ
  RETURN, READ_XWD(FILE, RED, GREEN, BLUE)
END
```

es3_write_bmp.pro

```
;ES3: IGNORE
PRO ES3_WRITE_BMP, FILE, IMAGE, RED, GREEN, BLUE, $
    FOUR_BIT=FOUR_BIT, $
    Ihd=Ihd, $
    HEADER_DEFINE=HEADER_DEFINE, $
    RGB=RGB
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_BMP', FILE=FILE, /WRITE
  WRITE_BMP, FILE, IMAGE, RED, GREEN, BLUE, $
    FOUR_BIT=FOUR_BIT, $
    Ihd=Ihd, $
    HEADER_DEFINE=HEADER_DEFINE, $
    RGB=RGB
END
```

es3_write_image.pro

```
;ES3: IGNORE
PRO ES3_WRITE_IMAGE, FILE, FORMAT, DATA, RED, GREEN, BLUE, $
    APPEND=APPEND, $
    _EXTRA=_EXTRA
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_IMAGE', FILE=FILE, /WRITE
  WRITE_IMAGE, FILE, FORMAT, DATA, RED, GREEN, BLUE, $
    APPEND=APPEND, $
    _EXTRA=_EXTRA
END
```

es3_write_jpeg.pro

```
;ES3: IGNORE
PRO ES3_WRITE_JPEG, FILE, IMAGE, $
    UNIT=UNIT, $
    ORDER=ORDER, $
    PROGRESSIVE=PROGRESSIVE, $
    QUALITY=QUALITY, $
```

```
    TRUE=TRUE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_JPEG', FILE=FILE, /WRITE
  WRITE_JPEG, FILE, IMAGE, $
    UNIT=UNIT, $
    ORDER=KEYWORD_SET(ORDER), $
    PROGRESSIVE=KEYWORD_SET(PROGRESSIVE), $
    QUALITY=QUALITY, $
    TRUE=TRUE
END
```

es3_write_nrif.pro

```
;ES3: IGNORE
PRO ES3_WRITE_NRIF, FILE, IMAGE, RED, GREEN, BLUE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_NRIF', FILE=FILE, /WRITE
  WRITE_NRIF, FILE, IMAGE, RED, GREEN, BLUE
END
```

es3_write_pict.pro

```
;ES3: IGNORE
PRO ES3_WRITE_PICT, FILE, IMAGE, RED, GREEN, BLUE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_PICT', FILE=FILE, /WRITE
  WRITE_PICT, FILE, IMAGE, RED, GREEN, BLUE
END
```

es3_write_png.pro

```
;ES3: IGNORE
PRO ES3_WRITE_PNG, FILE IMAGE, RED, GREEN, BLUE, $
    VERBOSE=VERBOSE, $
    TRANSPARENT=TRANSPARENT, $
    ORDER=ORDER
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_PNG', FILE=FILE, /WRITE
  WRITE_PNG, FILE IMAGE, RED, GREEN, BLUE, $
    VERBOSE=KEYWORD_SET(VERBOSE), $
    TRANSPARENT=TRANSPARENT, $
    ORDER=KEYWORD_SET(ORDER)
END
```

es3_write_ppm.pro

```
;ES3: IGNORE
PRO ES3_WRITE_PPM, FILE, IMAGE, $
    ASCII=ASCII
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_PPM', FILE=FILE, /WRITE
  WRITE_PPM, FILE, IMAGE, $
    ASCII=ASCII
END
```

es3_write_srf.pro

```
;ES3: IGNORE
PRO ES3_WRITE_SRF, FILE, IMAGE, RED, GREEN, BLUE, $
    WRITE_32=WRITE_32, $
    ORDER=ORDER
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_SRF', FILE=FILE, /WRITE
  WRITE_SRF, FILE, IMAGE, RED, GREEN, BLUE, $
    WRITE_32=WRITE_32, $
    ORDER=ORDER
END
```

es3_write_sylk.pro

```
;ES3: IGNORE
PRO ES3_WRITE_SYLK, FILE, DATA, $
    STARTROW=STARTROW, $
    STARTCOL=STARTCOL
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_SYLK', FILE=FILE, /WRITE
  WRITE_SYLK, FILE, DATA, $
    STARTROW=STARTROW, $
    STARTCOL=STARTCOL
END
```

es3_write_tiff.pro

```
;ES3: IGNORE
PRO ES3_WRITE_TIFF, FILE, IMAGE, $
    APPEND=APPEND, $
    BITS_PER_SAMPLE=BITS_PER_SAMPLE, $
    RED=RED, $
    GREEN=GREEN, $
    BLUE=BLUE, $
    COMPRESSION=COMPRESSION, $
    GEOTIFF=GEOTIFF, $
    LONG=LONG, $
    SHORT=SHORT, $
    FLOAT=FLOAT, $
    ORIENTATION=ORIENTATION, $
    PLANARCONFIG=PLANARCONFIG, $
    VERBOSE=VERBOSE, $
    XRESOL=XRESOL, $
    YRESOL=YRESOL
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_TIFF', FILE=FILE, /WRITE
  WRITE_TIFF, FILE, IMAGE, $
    APPEND=KEYWORD_SET(APPEND), $
    BITS_PER_SAMPLE=BITS_PER_SAMPLE, $
    RED=RED, $
    GREEN=GREEN, $
    BLUE=BLUE, $
    COMPRESSION=COMPRESSION, $
    GEOTIFF=GEOTIFF, $
    LONG=KEYWORD_SET(LONG), $
    SHORT=KEYWORD_SET(SHORT), $
    FLOAT=KEYWORD_SET(FLOAT), $
    ORIENTATION=ORIENTATION, $
    PLANARCONFIG=PLANARCONFIG, $
    VERBOSE=KEYWORD_SET(VERBOSE), $
    XRESOL=XRESOL, $
    YRESOL=YRESOL
```

```
END
```

es3_write_wav.pro

```
;ES3: IGNORE
PRO ES3_WRITE_WAV, FILE, DATA, RATE
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_WAV', FILE=FILE, /WRITE
  WRITE_WAV, FILE, DATA, RATE
END
```

es3_write_wave.pro

```
;ES3: IGNORE
PRO ES3_WRITE_WAVE, FILE, DATE, $
    BIN=BIN, $
    NOMESHDEF=NOMESHDEF, $
    DATANAME=DATANAME, $
    MESHNAME=MESHNAME, $
    VECTOR=VECTOR
  COMPILE_OPT HIDDEN
  ES3_LOG, ROUTINE='WRITE_WAVE', FILE=FILE, /WRITE
  WRITE_WAVE, FILE, DATE, $
    BIN=BIN, $
    NOMESHDEF=NOMESHDEF, $
    DATANAME=DATANAME, $
    MESHNAME=MESHNAME, $
    VECTOR=VECTOR
END
```

B.2 Bash plugin scripts

B.2.1 es3_straceproc

```
#!/usr/bin/perl
use Switch;

sub help{
  print STDERR "postproc is a parser for strace output\n";
  exit;
}
while ($arg=shift(@ARGV)){
  switch($arg) {
    case "-h" {
      help;
    }
    case "--help"{
      help;
    }
  }
}
#include es3_log
my $bin_dir = `which es3_log`;
$bin_dir =~ s/\/\//\//g;
$bin_dir =~ s/[^\/]*$//;

push(@INC, $bin_dir);
require "es3_log";
```

```
my %procs;

sub getFile :method {
  $pid = @_[0];
  $fid = @_[1];
  $rw = @_[2];
  if ($procs{$pid}{'file'}{$fid}){
    return $procs{$pid}{'file'}{$fid};
  } elsif ($procs{$pid}{'file'}{$rw.$fid}) {
    return "<".$procs{$pid}{'file'}{$rw.$fid}.">";
  } elsif ($procs{$pid}{'parent'}) {
    my $parent = $procs{$pid}{'parent'};
    return getFile($parent, $fid, $rw);
  }
  return '<std-in>' if ($fid == 0);
  return '<std-out>' if ($fid == 1);
  return '<std-err>' if ($fid == 2);
  return '<null>';
}

my $start_time = 0;
sub writeProc {
  my $pid=@_[0];
  my $ppid = $procs{$pid}{'parent'};
  if($pid and $procs{$pid}{'cmd'} and not $procs{$pid}{'ignore'}){
    if (not $procs{$ppid}{'init'}) {
      es3_log("--lpid", $ppid,"--init",
        "--language", "bash");#, "--stime", $start_time);
      $procs{$ppid}{'init'} = 1;
    }
    #print("$pid executed:".$procs{$pid}{'cmd'}."\n");
    my $file_arg = "";
    for $i (0 ... $#{$procs{$pid}{'io'}}){
      #print($procs{$pid}{'io'}[$i][0]." ");
      $file_arg .= join(",", @{$procs{$pid}{'io'}[$i]});
      $file_arg .= ":";
    }
    my @larg;
    $larg[0] = "--pid";
    $larg[1] = $pid;
    $larg[2] = "--ppid";
    $larg[3] = $procs{$pid}{'parent'};
    $larg[4] = "--lpid";
    $larg[5] = $procs{$pid}{'parent'}?$procs{$pid}{'parent'}:$pid;
    $larg[6] = "--files";
    $larg[7] = $file_arg;
    $larg[8] = "--routine";
    $larg[9] = $procs{$pid}{'cmd'};
    $larg[9] =~ s/^"(.*)"$/$1/;
    $larg[10] = "--arguments";
    $larg[11] =  join(", ", @{$procs{$pid}{'exec_arg'}});
    $larg[12] = "--time";
    $larg[13] = tsToStd($procs{$pid}{'execve-time'});
 #   $larg[14] = "--stime";
 #   $larg[15] = $start_time;
    $larg[16] = "--enviroment";
    my $env = "";
    while(($name, $value) = each(%{$procs{$pid}{'enviroment'}})) {
      $env .= $name . '="' . $value . '":';
    }
    $larg[17] = $env;
    es3_log(@larg);
    delete($procs{$pid});
  }
}

open(raw, ">/tmp/es3/raw");
```

```
open(input, '-');
my %unfinished;
my $pipes = 1;
while($line = <input>) {
  #putting the <unfinished> and <resumed> lines back together
  if ($line =~ /^(([0-9]+)[\t ]+.*)[\t ]+<unfinished[\t ]+\.\.\.>$/) {
    $unfinished{$2} = $1;
    next;
  }
  if ($line =~ s/^([0-9]+)[\t ]+[0-9:\.]+[\t ]+<\.\.\.[\t ]+[a-z_0-9]+(
    ){0}[\t ]+resumed>(.*)$/$unfinished{$1}$2/){
    delete($unfinished{$1});
  }
  print raw $line;

  if ($line =~ /^([0-9]+)[\t ]+([0-9:\.]+)[\t ]+([a-z_0-9]+)\((.*)\)(
    ){0}[ \t]*=[\t ]*([0-9\-]*)/) {
    my $pid = $1;
    my $ts = $2;
    my $call = $3;
    my $unsplit_arg = $4;
    my @arg = split(/, /,$4);
    my $ret = $5;
    if (not $main_pid) {
      $main_pid = $pid;
      $start_time = tsToStd($ts);
    }
#    if (not $procs{$pid}{'start_time'})
    switch($call) {
      case "open" {
        if($ret =~ /^[0-9]+$/) {
          my $pwd = ${$procs{$pid}{'enviroment'}}{"PWD"};
          my $fname = @arg[0];
          if ($pwd and not $fname =~ /^"\// ) {
            $fname =~ s/^"//;
            $fname = '"' . $pwd . '/' . $fname;
            $fname =~ s/\/[^\/]+\/\.\.\///\//g;
          }
          $procs{$pid}{'file'}{$ret} = $fname;
          @op = "READ" if($arg[1] =~ /O_RDONLY/);
          @op = "WRITE" if($arg[1] =~ /O_WRONLY/);
          push(@{$procs{$pid}{'io'}}, [$fname, @op]);
        }
      }
        #case "close"{
          #print("$pid close @arg[0]\n");
        #}
      case "dup2"{
        if ($arg[1] == 0) { #stdin
          $file = getFile($pid, $arg[0], 'r');
        } elsif($arg[1] == 1 or $arg[1] == 2){ #stdout/stderr
          $file = getFile($pid, $arg[0], 'w');
        } else {
          $file = getFile($pid, $arg[0]);
        }
          $procs{$pid}{'file'}{$arg[1]} = $file;
      }
      case "execve"{
        $procs{$pid}{'cmd'} = $arg[0];
        $procs{$pid}{'execve-time'} = $ts;
        my @exec_arg = ();
        if($unsplit_arg =~ /".*", \[(.*)\], \[(.*)\]/){
          @exec_arg = split(/, /, $1);
          delete (@exec_arg[0]);
          my $env_string = $2;
          my %enviroment;
```

```
          if($env_string) {
            #print("ENV:".$env_string."\n");
            my @env_parts = split(/, /, $env_string);
            my $last_part = "";
            foreach $part (@env_parts){
              if ($part =~ /[^\\](\\\\)*"/) {
                $valid_part = $last_part .
                  ($last_part?", ":"") . $part;
                if ($valid_part =~ /"([a-zA-Z_0-9]+)=(.*)"/) {
                  $enviroment{$1}= $2;
                  #print($1 ."=".$enviroment{$1}."'\n");
                }
                  $last_part = "";
              } else {
                $last_part = $last_part . ", " . $part;
              }
            }
            $procs{$pid}{'ignore'} = $enviroment{'ES3_IGNORE'};
          }
          %{$procs{$pid}{'enviroment'}} = %enviroment;
        }
        @{$procs{$pid}{'exec_arg'}} = @exec_arg;
        my @io = ();
        $io[0] = [getFile($pid, 0), "READ"];
        $io[1] = [getFile($pid, 1), "WRITE"];
        $io[2] = [getFile($pid, 2), "WRITE"];
        for $i (0...2){
          push(@{$io[$i]}, "PIPE")
            if($io[$i][0] =~ s/^<(.*)>$/$1/);
        }
        @{$procs{$pid}{'io'}} = @io;
      }
      case "pipe"{
        #0 is set up for reading, 1 is set up for writing
        $arg[0] =~ s/^\[//;
        $arg[1] =~ s/\]$//;
        delete $procs{$pid}{'file'}{"${arg[0]}"};
        delete $procs{$pid}{'file'}{"${arg[1]}"};
        $procs{$pid}{'file'}{"r${arg[0]}"} = $pipes;
        $procs{$pid}{'file'}{"w${arg[1]}"} = $pipes;
        $pipes++;
        #print("$pid pipe r:" . $arg[0] . " w:".$arg[1]."\n");
      }
      case "clone"{
        $procs{$ret}{'parent'} = $pid
      }
      case "exit_group"{
        writeProc($pid);
        exit if ($pid eq $main_pid);
      }
    }
  } elsif ($line =~ /^([0-9]+)[\t ]+([0-9:\.]+)[\t ]+---[\t ]+SIGCHLD.*/) {
    my $pid = $1;
    my $ts = $2;
    #print("one of " . $pid . " children exited\n");
  # For AMD processors
  } elsif ($line =~ /[\t 0-9]+exit_group\([^\)]+$/) {
#    exit;
  } else {
    print("ES3 could not parse: " . $line);
  }
}

for my $pid ( keys %procs ) {
  writeProc($pid);
}
```

```
close(input);
#close(bash);
#%h = %{$procs{'17884'}{'file'}};
#print($h{3}."\n");
```

B.2.2 es3_include.sh

```
if [ "$ES3_ENABLE" != "${ES3_ENABLE#*bash}" -a -z "$ES3_BASH_PLUGIN" ]; then
  export ES3_IGNORE=bash_startup
  export ES3_BASH_PLUGIN=running
  es3_trace() {
    if [ ! -d /tmp/es3/ ]; then
      mkdir /tmp/es3
      chmod 777 /tmp/es3
    fi
    pid=$1
    strace -v -s 1000 \
      -e trace=open,dup2,execve,pipe,clone,exit_group \
      -p $pid -f -F -ttt -o "| es3_straceproc" &> /tmp/es3/err
    chmod a+rw `find /tmp/es3/* -user $USER`

  }
  echo staring trace
  # start strace in the background
  es3_trace $$ &
  # wait for strace to start (no, it's not a good solution)
  sleep 0.1
  unset ES3_IGNORE
  unset ES3_BASH_PLUGIN
fi
```

B.3 Logger

```
#!/usr/bin/perl
use Switch;
use Sys::Hostname;


sub help {
  print STDERR "Usage: es3_log [options]
Options are:
  -h,--help                     Display this usage text
  --pid <pid>                   The pid of the process
  --ppid <pid>                  The pid of the parent process
  --lpid <pid>                  The pid the process should be logged under
  --stime <time>                The start time of the process
  --enter <name>                Enter block or function <name>
  --leave <name>                Leave block or function <name>
  --routine <name>              Routine <name> is called
  --enviroment <enviroment>     The enviroment in a colon " .
"seperated list of:
                                    [(type)] <name> = \"<value>\"
";
exit;
}

my %months = (
  "Jan" => 0,
  "Feb" => 1,
  "Mar" => 2,
  "Apr" => 3,
  "May" => 4,
```

```
  "Jun" => 5,
  "Jul" => 6,
  "Aug" => 7,
  "Sep" => 8,
  "Oct" => 9,
  "Nov" => 10,
  "Dec" => 11);

sub tsToStd :method {
  use Time::gmtime;
  if(@_[0]){
    my $ts = @_[0];
    $gm = gmtime($ts);
    my %d;
    $d{'Y'} =  $gm->year+1900;
    $d{'m'} =  $gm->mon;
    $d{'d'} =  $gm->mday;
    $d{'H'} =  $gm->hour;
    $d{'M'} =  $gm->min;
    $d{'S'} =  $gm->sec;
    for $key ('m', 'd', 'H', 'M', 'S') {
      $d{$key} = '0'.$d{$key} if ($d{$key}<10);
    }
    $d{'S'} .= $1 if($ts =~ /.*(\..*)/);

    return "$d{'Y'}$d{'m'}$d{'d'}T$d{'H'}$d{'M'}$d{'S'}Z";
  }
}

sub stdToTs :method {
  use Time::Local;
  if (@_[0]) {
    my $std = @_[0];
    if($std =~ /([0-9]{4})([0-9]{2})([0-9]{2})(
    ){0}T([0-9]{2})([0-9]{2})([0-9]{2})(\.[0-9]*)?Z/) {
      return timegm($6, $5, $4, $3, $2, $1);
    }
    0;
  }
}


sub psToStd :method {
  use Time::Local;
  if(@_[0]){
    my @part = split(/[: ]+/, @_[0]);
    return tsToStd(timelocal($part[5], $part[4], $part[3],
      $part[2], $months{$part[1]}, $part[6])) if ($part[5]);
  }
}

sub escapeXML :method {
  $line = @_[0];
  $line =~ s/</&lt;/g;   #<
  $line =~ s/>/&gt;/g;   #>
  $line =~ s/'/&apos;/g; #'
  $line =~ s/"/&quot;/g; #"
  return $line;
}

sub print_env{
  my @enviroment = @_;
  if(@enviroment){
    print log_file "  <enviroment>\n";
    for $i (0...$#enviroment){
      print log_file "    <variable ";
      print log_file 'type="'.escapeXML($enviroment[$i][0]) . '" '
```

```
        if $enviroment[$i][0];
      print log_file 'name="'.escapeXML($enviroment[$i][1]) . '" ';
      print log_file 'value="';
      print log_file escapeXML($enviroment[$i][2]) . '"';
      print log_file "/>\n";
    }
    print log_file "  </enviroment>\n";
  }
}

# usage: get_stime(<pid>, [default])
sub get_stime :method {
  if (@_[0]) {
    my $spid = @_[0];
    my $start_std;
    if (@_[1]) {
      $start_std = @_[1];
    } else {
      my $start_ps = `ps -p $spid -o "lstart=" --no-headers 2> /dev/null`;
      $start_std = psToStd($start_ps);
    }
    my $start_ts = stdToTs($start_std);

    #determine if there is a logfile with a simmular start
    #time and use that instead
    #this is to fix a precsission bug in the ps command
    open(ls, "ls -1 /tmp/es3/*-$spid 2> /dev/null |");
    while ($entry = <ls>) {
      if ($entry =~ /([0-9]{8}T[0-9]{6}(\.[0-9]*)?Z)-([0-9]+)/) {
        if (abs(stdToTs($1)-$start_ts)<3) {
          return $1;
        }
      }
    }
    close(ls);
    return $start_std;
  }
  0;
}

sub es3_log {
  my $init = 0;
  my $pid = 0;
  my $enter = 0;
  my $leave = 0;
  my $time = 0;
  my @files = ();
  my $language = 0;
  my @args = @_;
  my @enviroment = ();
  my $arguments = 0;

  while ($arg=shift(@args)){
    switch($arg) {
      case "-h" {
        help
      }
      case "--help"{
        help
      }
      case "--pid" {
        $pid=shift(@args);
      }
      case "--ppid" {
        $ppid=shift(@args);
      }
      case "--lpid" {
```

```
        $lpid=shift(@args);
      }
      case"--enter" {
        $enter=shift(@args);
      }
      case"--leave" {
        $leave=shift(@args);
      }
      case"--routine" {
        $routine=shift(@args);
      }
      case"--arguments" {
        $arguments=shift(@args);
      }
      case"--time" {
        $time=shift(@args);
      }
      case"--stime" {
        $stime=shift(@args);
      }
      case"--files" {
        foreach $part (split(/:/, shift(@args))){
          push(@files, [split(/,[ \t]*/, $part)]);
          ${files[$#files]}[0] =~ s/"(.*)"/$1/;
        }
      }
      case"--language" {
        $language=shift(@args);
      }
      case"--init" {
        $init = 1;
      }
      case"--print" {
        $print = 1;
      }
      case"--enviroment" {
        @all = reverse split(/:/, shift(@args));
        while($part = pop(@all)){
          while(((not $part =~ /[^\\](\\\\)*"$/)
            or $part =~ /^[^=]*=[ \t]*"$/) and @all > 0) {
            $missing = pop(@all);
            $part .= ':'.$missing;
          }
          next if not
            $part =~ /(\((.*)\))?[ \t]*(.*)[ \t]*=[ \t]*"(.*)"/;
          push(@enviroment,[$2, $3, $4]);
          for $i (0 ... 2) {
            $enviroment[@enviroment-1][$i] =~ s/\\"/"/g;
            $enviroment[@enviroment-1][$i] =~ s/\\\\/\\/g;
          }
        }
      }
    }
  }
  $pid = getppid() if not $pid;
  if (not $ppid) {
    $ppid = `ps -p $pid -o "ppid=" --no-headers 2> /dev/null`;
    $ppid =~ s/[^0-9]//g;
  }
  $stime = get_stime($pid, $stime);
  $pstime = get_stime($ppid) if ($ppid);

  $time=tsToStd(time()) if not $time;
  $user = getlogin() if not $user;
  $user = $ENV{'USER'} if not $user;

  if($print){
```

```
    open(log_file, '>-');
  } else {
    $log_file_name = "/tmp/es3/".$stime."-";
    if ($lpid) {
      $log_file_name .= $lpid;
    } else {
      $log_file_name .= $pid;
    }
    open(log_file, ">>". $log_file_name);
  }

  if($enter) {
    print log_file  '<enter region="'.escapeXML($enter).'"';
    if(@enviroment) {
      print log_file ">\n";
      print_env(@enviroment);
      print log_file "</enter>\n";
    } else {
      print log_file "/>";
    }
  }
  if($leave) {
    print log_file  '<leave region="'.escapeXML($leave).'"';
    if(@enviroment) {
      print log_file ">\n";
      print_env(@enviroment);
      print log_file "</leave>\n";
    } else {
      print log_file "/>";
    }
  }
  if($init) {
    print log_file '<init';
    print log_file  ' time="'.$time.'"' if($time);
    print log_file  ' pid="'.$pid.'"' if($pid);
    printf log_file  ' stime="%s"', $stime if ($stime);
    printf log_file  ' pstime="%s"', $pstime if ($pstime);
    print log_file  ' ppid="'.$ppid.'"' if($ppid);
    print log_file  ' language="'.escapeXML($language).'"'
      if($language);
    print log_file  ' user="' . escapeXML($user) . '"' if($user);
    print log_file  ' hostname="' . escapeXML(hostname()) . '"';
    print log_file ">\n";
    print_env(@enviroment);
    open(mount, "mount|");
    $mounts = 0;
    while($line = <mount>){
      if($line =~ /(.*) on (.*) type ((smbfs)|(nfs)) \(.*\)/) {
        print log_file "  <mount-points>\n" if not $mounts;
        $mounts=1;
        printf(log_file
          "    <mount share=\"%s\" type=\"%s\">%s</mount>\n",
        escapeXML($1), escapeXML($5), escapeXML($2));
      }
    }
    close(mount);
    print log_file "  </mount-points>\n" if $mounts;
    print log_file "</init>\n";
  } elsif($routine) {
    print log_file "<exec";
    print log_file  ' time="'.$time.'"' if($time);
    print log_file  ' routine="'.escapeXML($routine).'"'
      if ($routine);
    if ($lpid) {
      print log_file  ' pid="'.$pid.'"';
      print log_file  ' ppid="'.$ppid.'"' if ($ppid);
    }
```

```
    if(@enviroment or @files or $arguments) {
      print log_file ">\n";
      if($arguments){
        my @arr = split(/,[ ]?/, $arguments);
        print log_file "  <arguments>\n";
        my $rest = "";
        foreach $arg (@arr) {
          $arg = $rest . $arg;
          if  (not $arg =~ s/^\"(.*[^\\](\\\\)?)\"$/$1/ ) {
            $rest = $arg;
          } else {
            print log_file "    <argument>";
            print log_file escapeXML($arg);
            print log_file "</argument>\n";
          }
        }
        print log_file "  </arguments>\n";
      }
      print_env(@enviroment);

      if(@files){
        print log_file ("  <io>\n");
        for  $i (0 ... $#files) {
          my $read = 0;
          my $write = 0;
          my $pipe = 0;
          for $j (1 ... $#{$files[$i]}){
            $read = 1 if ($files[$i][$j] eq "READ");
            $write = 1 if ($files[$i][$j] eq "WRITE");
            $pipe = 1 if ($files[$i][$j] eq "PIPE");
          }
          print log_file ("    <".($pipe?'pipe':'file'));
          print log_file (' read="true"') if ($read);
          print log_file (' write="true"') if ($write);
          print log_file ($pipe?' id="":">");
          print log_file (escapeXML($files[$i][0]));
          print log_file (($pipe?'"/>':'</file>')."\n");
        }
        print log_file ("  </io>\n");
      }

      print log_file "</exec>\n";
    } else {
      print log_file "/>\n";
    }
  }

  close(log_file);
  unlink("/tmp/es3/last");
  symlink($log_file_name, "/tmp/es3/last");
  chmod(0666, "/tmp/es3/last");
}

unless ( -d "/tmp/es3" ){
  mkdir "/tmp/es3";
  chmod 0777, "/tmp/es3";
}

$ENV{'ES3_IGNORE'}="true";
es3_log(@ARGV) if(@ARGV);
1;
```

B.4 Transmitter

B.4.1 ExecElement.java

```
package edu.ucsb.eil.es3.client;

import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;

import java.util.ArrayList;
import java.util.Vector;

/**
 * User: haavar
 * Date: Apr 13, 2005
 * Time: 5:02:13 PM
 */
public class ExecElement extends LogEntry {
  Element source;
  FileElement[] files = null;
  PipeElement[] pipes = null;
  private String region;

  public ExecElement(Element source) {
    this.source = source;

    Element elements[] = getSubElements(source, "io");
    ArrayList fileList = new ArrayList();
    ArrayList pipeList = new ArrayList();

    for (int i = 0; i < elements.length; i++) {
      if (elements[i].getTagName().equals("pipe"))
        pipeList.add(new PipeElement(elements[i], this));
      else if (elements[i].getTagName().equals("file"))
        fileList.add(FileElement.getFileElement(elements[i]));

    }
    files = new FileElement[fileList.size()];
    fileList.toArray(files);
    pipes = new PipeElement[pipeList.size()];
    pipeList.toArray(pipes);

  }

  public String[] getArguments() {
    NodeList argumentNodes = source.getElementsByTagName(
        "arguments");
    if (argumentNodes.getLength() > 0 &&
        argumentNodes.item(0).getNodeType() ==
        Node.ELEMENT_NODE) {
      NodeList arguments =
          ((Element) argumentNodes.item(0)).getChildNodes();
      ArrayList argumentList = new ArrayList();
      for (int i = 0; i < arguments.getLength(); i++) {
        Node subNode = arguments.item(i);
        if (subNode.getNodeType() == Node.ELEMENT_NODE)
          argumentList.add(
              subNode.getFirstChild().getNodeValue());
      }
      String[] ret = new String[argumentList.size()];
      argumentList.toArray(ret);
      return ret;
    }
    return new String[]{};
  }
```

```
  public String getRoutine() {
    return source.getAttribute("routine");
  }

  public int getPid() {
    try {
      return Integer.parseInt(source.getAttribute("pid"));
    } catch (Exception e) {
      return -1;
    }
  }

  public String getTime() {
    return source.getAttribute("time");


  }

  public PipeElement[] getPipes() {
    return pipes;
  }

  public FileElement[] getFiles() {
    return files;
  }

  public ExecElement[] getInputs() {
    Vector inputs = new Vector(3);
    for (int i = 0; i < pipes.length; i++) {
      ExecElement[] subSet = pipes[i].getInputs();
      for (int j = 0; j < subSet.length; j++)
        inputs.add(subSet[j]);
    }
    ExecElement[] ret = new ExecElement[inputs.size()];
    inputs.copyInto(ret);
    return ret;
  }

  public ExecElement[] getOutputs() {
    Vector outputs = new Vector(3);
    for (int i = 0; i < pipes.length; i++) {
      ExecElement[] subSet = pipes[i].getOutputs();
      for (int j = 0; j < subSet.length; j++)
        outputs.add(subSet[j]);
    }
    ExecElement[] ret = new ExecElement[outputs.size()];
    outputs.copyInto(ret);
    return ret;
  }


  public short getType() {
    return EXEC_ELEMENT;
  }

  public String getRegion() {
    return region;
  }

  public void setRegion(String region) {
    this.region = region;
  }
}
```

B.4.2 FileElement.java

```
package edu.ucsb.eil.es3.client;

import org.w3c.dom.Element;
import org.w3c.dom.Node;

import java.util.Hashtable;

/**
 * User: haavar
 * Date: Apr 13, 2005
 * Time: 10:20:10 PM
 */
public class FileElement {
  private static Hashtable all = new Hashtable();
  private Element source;

  private FileElement(Element source) {
    this.source = source;
  }

  public static FileElement getFileElement(Element source) {
    FileElement tmp = new FileElement(source);
    if (all.containsKey(tmp.getName()))
      return (FileElement) all.get(tmp.getName());
    else {
      all.put(tmp.getName(), tmp);
      return tmp;
    }
  }

  public String getName() {
    Node nameN =  source.getFirstChild();

    if(nameN!=null) return nameN.getNodeValue();
    else return "";
  }

  public boolean getRead() {
    return Boolean.valueOf(source.getAttribute("read"))
        .booleanValue();
  }

  public boolean getWrite() {
    return Boolean.valueOf(source.getAttribute("write"))
        .booleanValue();
  }
}
```

B.4.3 InitElement.java

```
package edu.ucsb.eil.es3.client;

import org.w3c.dom.Element;

/**
 * User: haavar
 * Date: Apr 13, 2005
 * Time: 5:02:04 PM
 */
public class InitElement extends LogEntry {
  Element source;

  public InitElement(Element source) {
    this.source = source;
  }
```

```
  public int getPid() {

    String pid = source.getAttribute("pid");
    if (pid != null) {
      try {
        return Integer.parseInt(pid);
      } catch (NumberFormatException e) {
      }
    }
    return -1;
  }

  public String getLanguage() {
    return source.getAttribute("language");
  }

  public String getUser() {
    return source.getAttribute("user");
  }

  public String getHostname() {
    return source.getAttribute("hostname");
  }

  public short getType() {
    return INIT_ELEMENT;
  }
}
```

B.4.4 LogEntry.java

```
package edu.ucsb.eil.es3.client;

import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;

import java.util.ArrayList;

/**
 * User: haavar
 * Date: Apr 13, 2005
 * Time: 5:32:15 PM
 */
public abstract class LogEntry {
  public static final short INIT_ELEMENT = 1;
  public static final short EXEC_ELEMENT = 2;
  public static final short BOARDER_ELEMENT = 3;

  private boolean processed = false;

  public Variable[] getEnviroment() {
    return null;
  }

  //getTime()
  protected static Element[] getSubElements(Element source,
                                            String tagName) {
    NodeList argumentNodes = source.getElementsByTagName(
        tagName);
    if (argumentNodes.getLength() > 0 &&
        argumentNodes.item(0).getNodeType() ==
        Node.ELEMENT_NODE) {
      NodeList arguments =
```

```
          ((Element) argumentNodes.item(0)).getChildNodes();
      ArrayList argumentList = new ArrayList();
      for (int i = 0; i < arguments.getLength(); i++) {
        Node subNode = arguments.item(i);
        if (subNode.getNodeType() == Node.ELEMENT_NODE)
          argumentList.add(subNode);
      }
      Element[] ret = new Element[argumentList.size()];
      argumentList.toArray(ret);
      return ret;
    }
    return null;
  }

  protected boolean isTrue(String bool) {
    if (bool == null)
      return false;
    return (bool.equals("1") || bool.equalsIgnoreCase("true"));
  }

  public abstract short getType();

  public boolean isProcessed() {
    return processed;
  }

  public void setProcessed() {
    processed = true;
  }

}
```

B.4.5 LogInputStream.java

```
package edu.ucsb.eil.es3.client;

import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.File;

/**
 * User: Haavar Valeur
 * Date: Apr 13, 2005
 * Time: 3:18:38 PM
 * This class adds one line to the start and one line to the
 * end of a filestream This is used to put the <es3-client-log>
 * tag around the client log so it can be parsed by a xml
 * parser
 */
public class LogInputStream extends FileInputStream {
  private byte[] head = "<es3-client-log>".getBytes();
  private byte[] tail = "</es3-client-log>".getBytes();
  private int pos = 0;

  LogInputStream(String fileName)
      throws FileNotFoundException {
    super(fileName);
  }

  LogInputStream(File file)
      throws FileNotFoundException {
    super(file);
  }
```

```
  public int available() throws IOException {
    return head.length + tail.length - pos +
        super.available();
  }

  public int read(byte[] b, int off, int len)
      throws IOException {
    int read = 0;
    int rlen = 0;
    rlen = Math.min(len, head.length - pos);
    if (rlen > 0) {
      System.arraycopy(head, pos, b, off, rlen);
      read += rlen;
      pos += rlen;
    }
    rlen = Math.min(len - read, super.available());
    if (rlen > 0) {
      super.read(b, off + read, rlen);
      read += rlen;
    }
    rlen =
        Math.min(len - read, head.length + tail.length - pos);
    if (rlen > 0) {
      System.arraycopy(tail, pos - head.length, b, off + read,
          rlen);
      read += rlen;
      pos += rlen;
    }
    return read > 0 ? read : -1;
  }

  public int read(byte[] b) throws IOException {
    return read(b, 0, b.length);

  }

  public int read() throws IOException {
    byte[] ret = new byte[1];
    read(ret, 0, 1);
    return ret[0];

  }

  public boolean markSupported() {
    return false;
  }


}
```

B.4.6 PipeElement.java

```
package edu.ucsb.eil.es3.client;

import org.w3c.dom.Element;

import java.util.Hashtable;
import java.util.Vector;

/**
 * User: haavar
 * Date: Apr 13, 2005
 * Time: 8:51:34 PM
 */
public class PipeElement {
```

```
private static Hashtable pipes = new Hashtable();
private static Vector counterParts;
private Element source;
private String name;
private ExecElement father;

public PipeElement(Element source, ExecElement father) {
  this.father = father;
  this.source = source;
  name = source.getAttribute("id");


  if (!isStandardPipe()) {
    //Vector pipes;
    if (pipes.containsKey(getName())) {
      counterParts = (Vector) (pipes.get(getName()));
    } else {
      counterParts = new Vector(3);
      pipes.put(this, getName());
    }
    counterParts.add(this);
  }
}


public boolean isStandardPipe() {
  return (getName().startsWith("<std-"));
}

private String getName() {
  return name;
}

private boolean getRead() {
  return Boolean.valueOf(source.getAttribute("read"))
      .booleanValue();
}

private boolean getWrite() {
  return Boolean.valueOf(source.getAttribute("write"))
      .booleanValue();
}

private ExecElement[] getIO(boolean in) {
  Vector valid = new Vector();
  for (int i = 0; i < counterParts.size(); i++) {
    PipeElement candidate = (PipeElement) counterParts.get(
        i);
    if (candidate.getExec() != father &&
        ((in && candidate.getWrite()) ||
        (!in && candidate.getRead())))
      valid.add(candidate.getExec());

  }
  ExecElement[] ret = new ExecElement[valid.size()];
  valid.copyInto(ret);
  return ret;

}

private ExecElement getExec() {
  return father;
}

public ExecElement[] getOutputs() {
  return getIO(false);
}
```

```
  public ExecElement[] getInputs() {
    return getIO(true);
  }

}
```

B.4.7 Transmitter.java

```
package edu.ucsb.eil.es3.client;

import java.io.*;
import java.util.ArrayList;
import java.sql.Timestamp;


/**
 * User: haavar
 * Date: Apr 13, 2005
 * Time: 3:05:23 PM
 */
public class Transmitter {
  private static final String TMP_DIR = "/tmp/es3/";
  private static ArrayList logNames = new ArrayList();

  public Transmitter() {
    File tmpDir = new File(TMP_DIR);
    File[] logFiles = tmpDir.listFiles(
        new LogFilenameFilter());
    for (int i = 0; i < logFiles.length; i++)
      addLogFileName(logFiles[i].getName());

    Log[] logs = new Log[logFiles.length];
    try {
      PrintStream out = new PrintStream(
          new FileOutputStream(
              "/tmp/es3/es3_message.xml"));
      for (int i = 0; i < logs.length; i++) {
        try {
          logs[i] = new Log(logFiles[i]);
          WorkflowMessage workflow = logs[i].getWorkflow();
          workflow.writeXML(out);
        } catch (FileNotFoundException e) {
          e.printStackTrace();
        }
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
  }

  public static void addLogFileName(String fileName) {
    logNames.add(fileName);
  }

  /**
   * Look for exciting logs matching the starup time and pid
   * For now it used the filename, but the pid and stime
   * is no longer used at a filename this function needs to
   * match those with the contents of the logfile
   *
   * @param stdTime Startup time of the process
   * @param pid     The process id
   * @return the filename of the logfile
   */
```

```
  public static String getLogID(String stdTime, int pid) {
    long ts = stdToTimeStamp(stdTime);
    for (int i = 0; i < logNames.size(); i++) {
      String fileName = (String) logNames.get(i);
      if (fileName.endsWith("Z-" + pid)) {
        long fileTS = stdToTimeStamp(fileName.substring(0,
            fileName.lastIndexOf("Z-") + 1));
        if (Math.abs(fileTS - ts) < 10*1000) return fileName;
      }
    }
    return null;
  }

  public static long stdToTimeStamp(String stdTime) {
    //yyyy-mm-dd hh:mm:ss.fffffffff
    if (!stdTime.matches("^[0-9]{8}T[0-9]{6}(\\.[0-9]+)?Z$"))
      throw new IllegalArgumentException(
          "Could not parse time: '" + stdTime + "'");
    String Y = stdTime.substring(0, 4);
    String M = stdTime.substring(4, 6);
    String D = stdTime.substring(6, 8);
    String h = stdTime.substring(9, 11);
    String m = stdTime.substring(11, 13);
    String s = stdTime.substring(13, stdTime.length() - 1);
    String time;
    if (s.lastIndexOf(".") < 0) s += ".";
    s += "000000000";
    s = s.substring(0, s.indexOf(".") + 9);
    time = Y + "-" + M + "-" + D + " " + h + ":" + m + ":" +
        s;

    return Timestamp.valueOf(time).getTime();
  }

  public static void main(String arg[]) {
    //System.out.println(
    //    stdToTimeStamp("20050508T050836.00012Z"));
    new Transmitter();
  }

  private class LogFilenameFilter implements FilenameFilter {

    public boolean accept(File dir, String fileName) {
      return fileName.matches(
          "^[0-9]{8}T[0-9]{6}(\\.[0-9]+)?Z-[0-9]+$");
    }
  }
}
```

B.4.8 Variable.java

```
package edu.ucsb.eil.es3.client;

import org.w3c.dom.Element;

/**
 * User: haavar
 * Date: Apr 13, 2005
 * Time: 5:08:37 PM
 */
public class Variable {


  String value;
  String type;
```

```
  String name;

  public Variable(Element source) {
    value = source.getAttribute("value");
    type = source.getAttribute("type");
    name = source.getAttribute("name");
  }

  public String getValue() {
    return value;
  }

  public String getType() {
    return type;
  }

  public String getName() {
    return name;
  }
}
```

B.4.9 WorkflowMessage.java

```
package edu.ucsb.eil.es3.client;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.PrintStream;
import java.util.*;

/**
 * User: haavar
 * Date: Jun 8, 2005
 * Time: 1:53:57 PM
 */
public class WorkflowMessage {
  ArrayList files = new ArrayList();
  ArrayList transformations = new ArrayList();
  Hashtable relations = new Hashtable();
  String localID;
  public WorkflowMessage(String localID) {
    this.localID = localID;
  }

  private String addObject(FileElement file) {
    if (!files.contains(file))
      files.add(file);
    return "F-" + files.indexOf(file);
  }

  public String addObject(ExecElement transf) {
    if (!transformations.contains(transf))
      transformations.add(transf);
    return "T-" + transformations.indexOf(transf);
```

```
}

public void addRelation(FileElement file,
                        ExecElement transf) {
  Relation rel = new Relation(addObject(file), addObject(transf));
  relations.put(rel, rel);
}

public void addRelation(ExecElement transf1,
                        ExecElement transf2) {
  Relation rel = new Relation(addObject(transf1), addObject(transf2));

  relations.put(rel, rel);
}

public void addRelation(ExecElement transf,
                        FileElement file) {
  Relation rel = new Relation(addObject(transf), addObject(file));

  relations.put(rel, rel);
}

public void writeXML(PrintStream out) {
  try {
    DocumentBuilder builder =
        DocumentBuilderFactory.newInstance()
        .newDocumentBuilder();
    Document doc = builder.newDocument();
    Element workflowE = doc.createElement("workflow");
    doc.appendChild(workflowE);
    workflowE.setAttribute("localID", localID);

    for (Iterator it = relations.values().iterator();
         it.hasNext();) {
      Element relationE = doc.createElement("relation");
      workflowE.appendChild(relationE);
      Relation relation = (Relation)it.next();
      relationE.setAttribute("to", relation.to);
      relationE.setAttribute("from", relation.from);
    }
    for (int i = 0; i < files.size(); i++) {
      FileElement file = (FileElement) files.get(i);
      Element fileE = doc.createElement("file");
      workflowE.appendChild(fileE);
      fileE.setAttribute("name", file.getName());
      fileE.setAttribute("id", addObject(file));
    }
    for (int i = 0; i < transformations.size(); i++) {
      ExecElement exec = (ExecElement) transformations.get(
          i);
      Element transfE = doc.createElement("transformation");
      workflowE.appendChild(transfE);
      transfE.setAttribute("routine", exec.getRoutine());
      transfE.setAttribute("id", addObject(exec));
      int pid = exec.getPid();
      String time = exec.getTime();
      if (pid > 0 && time != null) {
        String ref = Transmitter.getLogID(time, pid);
        if (ref != null)
          transfE.setAttribute("workflow", ref);
      }

      String[] argumetnts = exec.getArguments();
      if (argumetnts.length > 0) {
        Element argsE = doc.createElement("arguments");
        transfE.appendChild(argsE);
```

```
            for (int j = 0; j < argumetnts.length; j++) {
              Element argE = doc.createElement("argument");
              argsE.appendChild(argE);
              argE.appendChild(
                  doc.createTextNode(argumetnts[j]));
            }
          }

        }
        Transformer transformer =
            TransformerFactory.newInstance()
            .newTransformer();
        transformer.setOutputProperty(OutputKeys.INDENT, "yes");
        transformer.transform(new DOMSource(doc),
            new StreamResult(out));

      } catch (ParserConfigurationException e) {
        e.printStackTrace();
      } catch (TransformerException e) {
        e.printStackTrace();
      }

    }

    private class Relation{
      String from;
      String to;
      Relation(String from, String to){
        this.from = from;
        this.to = to;

      }

      public String toString(){
        return to+"-"+from;
      }
    }

}
```

B.4.10 XMLParser.java

```
package edu.ucsb.eil.es3.common;

import org.w3c.dom.Document;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import java.io.IOException;
import java.io.InputStream;

/**
 * User: Haavar Valeur
 * Date: Sep 3, 2004
 * Time: 3:23:18 PM
 */
public class XMLParser {
    private DocumentBuilder builder = null;

    public XMLParser() {
        DocumentBuilderFactory domFactory =
            DocumentBuilderFactory.newInstance();
```

```
        domFactory.setIgnoringElementContentWhitespace(true);
        domFactory.setIgnoringComments(true);
        domFactory.setNamespaceAware(true);

        try {
            builder = domFactory.newDocumentBuilder();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        }
    }



    public Document parse(InputStream in) {
        return parse(new InputSource(in));
    }

    public Document parse(java.io.Reader xmlRequest) {
        return parse(new InputSource(xmlRequest));
    }

    public Document parse(InputSource input) {
        if (input == null) return null;

        Document doc = null;

        try {
            doc = builder.parse(input);
        } catch (SAXException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        return doc;
    }


}
```

This page is intentionally left blank.

Appendix C

Short installation instructions

General instalations steps

- Unpack the client package where you want it. The client directory you unpacked is from now on referenced as **ES3C_HOME**.
- Add the 'bin' folder inside **ES3C_HOME** to your executable path. This can be done by adding it to the *$PATH* enviroment variable in your .profile, .bash_profile or .bashrc files depending on your preferences or system setup.

Logs entries are made in /tmp/es3/<process-start-time>-<pid> and /tmp/es3/last will be a symlink to the latest log.
$ES3_ENABLE may contain multiple languages separated by colon. To disable plugins unset the *$ES3_ENABLE* variable or remove the desired plugin from the variable and restart the session.

IDL plugin

- Add the 'idl' folder inside **ES3C_HOME** to the IDL-path. If you already are setting your *$IDL_PATH* environment variable in some of your startup scripts you can add the folder to the list. If you don't have *$IDL_PATH* set, setting it would delete the default idl searchpath. In the latter case it is easier to add the line "!PATH = '**ES3C_HOME**/idl:' + !PATH" where **ES3C_HOME** is the client directory to the startup script made in the next step.
- Make idl run the command 'es3_startup' on startup. To make idl run a command at startup we use a startupscirpt. A startup-script is a script that is pointed to by the environment variable *$IDL_STARTUP*. If you have a startup-script file you are using then you add the command 'es3_startup' to the last line of your script. If you don't have a starup-script file you create a text file with one line: 'es3_startup' and set the *$IDL_STARTUP* enviroment variable to point at the file.

To enable the bash plugin you need to add ":idl" to the environment variable *$ES3_ENABLE*.

Bash plugin:

- To install the bash plugin you need to source the file **ES3C_HOME**/bash/es3_include.sh in a starupscript that is ran every time you start bash. I've done this by adding the line ". **ES3C_HOME**/bash/es3_include.sh" where **ES3C_HOME** is the home of the client to my .bashrc file. Make sure it runs the script every time you start bash, not only when you start a new login session.

To enable the bash plugin you need to add ":bash" to the environment variable *$ES3_ENABLE*.

This page is intentionally left blank.

Appendix D

IDL example workflow

To demonstrate the the use of the IDL plugin I've chosen to use a IDL script written by Tom Painter. The script is called modscag_cleance and is used to clean up values in pictures that is a part of one of his workflows.

Figure D.1 shows the overall linage of the script. This script reads and writes to 5 files. The fact that the script reads and writes from the same file is not a problem for the client, but it makes it harder to interpret the behavior of the program. There is only one transformation because there was only a single procedure call in the example code.

D.1 Source code

D.1.1 Original source code

This is the source code before it has been processed.

```
pro modscag_cleanse,prefix=prefix,ns=ns,nl=nl

; modscag_cleanse
;
; IDL program to clean up under and overflow of MODSCAG run plus
; using snow cover to remove unnecessary grain size estimates.
;
; Input
;   prefix = prefix for all of the MODSCAG output filenames
;   ns = number of samples
;   nl = number of lines
;
; Output
;   rewrite of the MODSCAG files
;
; t.h.painter
; 1.19.2005
;

  ; open snow file
  openr,1,string(prefix,'snow.pic')
  snow=fltarr(ns,nl)
  readu,1,snow
  close,1

  ; cleanse snow file
        if (min(snow) lt 0.0) then snow(where(snow lt 0.1))=0.0
  snow(where(snow lt 0.1))=0.0
  if (max(snow) gt 1.0) then snow(where(snow gt 1.0))=1.0

  ; output snow file
  openw,10,string(prefix,'snow.pic')
```

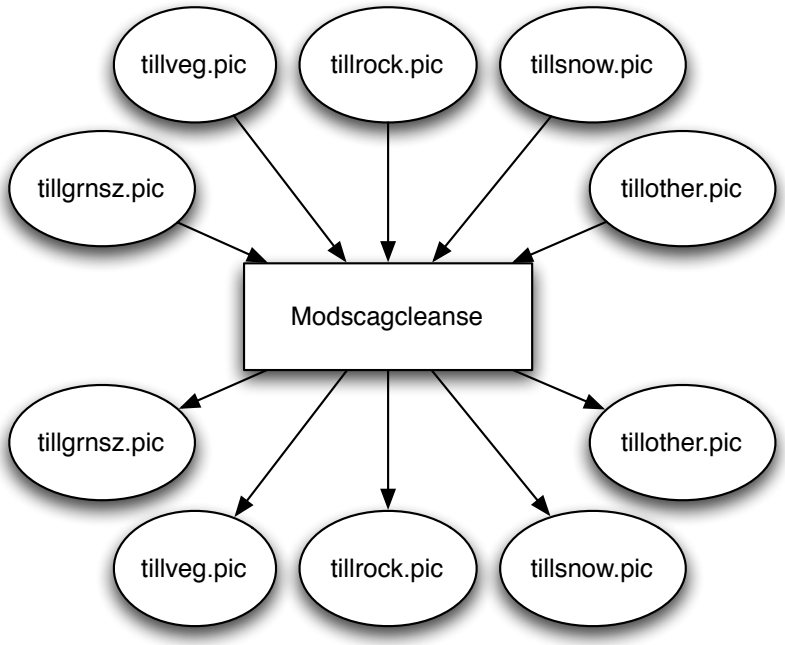

Figure D.1: A DAG representing the information flow of modscag_cleance

```
writeu,10,snow
close,10

; open grain size file
      openr,1,string(prefix,'grnsz.pic')
grnsz=fltarr(ns,nl)
readu,1,grnsz
close,1

; cleanse grain size file
grnsz(where(snow lt 0.2))=0.0

; output grain size file
openw,10,string(prefix,'grnsz.pic')
writeu,10,grnsz
close,10

; open vegetation file
openr,1,string(prefix,'veg.pic')
veg=fltarr(ns,nl)
readu,1,veg
close,1

; cleanse vegetation file
if (min(veg) lt 0.0) then veg(where(veg lt 0.0))=0.0
if (max(veg) gt 1.0) then veg(where(veg gt 1.0))=1.0

; output vegetation file
openw,10,string(prefix,'veg.pic')
writeu,10,veg
close,10

      ; open rock file
      openr,1,string(prefix,'rock.pic')
```

```
        rock=fltarr(ns,nl)
        readu,1,rock
        close,1

        ; cleanse rock file
        if (min(rock) lt 0.0) then rock(where(rock lt 0.0))=0.0
        if (max(rock) gt 1.0) then rock(where(rock gt 1.0))=1.0

        ; output rock file
        openw,10,string(prefix,'rock.pic')
        writeu,10,rock
        close,10

        ; open other file
        openr,1,string(prefix,'other.pic')
        other=fltarr(ns,nl)
        readu,1,other
        close,1

        ; cleanse other file
        if (min(other) lt 0.0) then other(where(other lt 0.0))=0.0
        if (max(other) gt 1.0) then other(where(other gt 1.0))=1.0

        ; output other file
        openw,10,string(prefix,'other.pic')
        writeu,10,other
        close,10

  end
```

D.1.2 Preprocessed source code

Below is the source code after it has gone through the preprocessing stage. In this example the procedures openr and openw, which opens files for reading and writing, have been changed to the wrapper functions. The code for the wrapper functions can be found in section B.1.6. The third and the third last line of the script is a call to the logger to record the execution of the modscag_clense procedure.

```
;edit
pro modscag_cleanse,prefix=prefix,ns=ns,nl=nl
HELP, NAMES="*", OUTPUT=ES3_ENVIROMENT & ES3_LOG, $
  ENTER="modscag_cleanse", ENVIROMENT=ES3_ENVIROMENT

; modscag_cleanse
;
; IDL program to clean up under and overflow of MODSCAG run plus
; using snow cover to remove unnecessary grain size estimates.
;
; Input
;   prefix = prefix for all of the MODSCAG output filenames
;   ns = number of samples
;   nl = number of lines
;
; Output
;   rewrite of the MODSCAG files
;
; t.h.painter
; 1.19.2005
;

  ; open snow file
  ES3_openr,1,string(prefix,'snow.pic')
  snow=fltarr(ns,nl)
  readu,1,snow
  close,1
```

```
; cleanse snow file
      if (min(snow) lt 0.0) then snow(where(snow lt 0.1))=0.0
snow(where(snow lt 0.1))=0.0
if (max(snow) gt 1.0) then snow(where(snow gt 1.0))=1.0

; output snow file
ES3_openw,10,string(prefix,'snow.pic')
writeu,10,snow
close,10

; open grain size file
      ES3_openr,1,string(prefix,'grnsz.pic')
grnsz=fltarr(ns,nl)
readu,1,grnsz
close,1

; cleanse grain size file
grnsz(where(snow lt 0.2))=0.0

; output grain size file
ES3_openw,10,string(prefix,'grnsz.pic')
writeu,10,grnsz
close,10

; open vegetation file
ES3_openr,1,string(prefix,'veg.pic')
veg=fltarr(ns,nl)
readu,1,veg
close,1

; cleanse vegetation file
if (min(veg) lt 0.0) then veg(where(veg lt 0.0))=0.0
if (max(veg) gt 1.0) then veg(where(veg gt 1.0))=1.0

; output vegetation file
ES3_openw,10,string(prefix,'veg.pic')
writeu,10,veg
close,10

      ; open rock file
      ES3_openr,1,string(prefix,'rock.pic')
      rock=fltarr(ns,nl)
      readu,1,rock
      close,1

      ; cleanse rock file
      if (min(rock) lt 0.0) then rock(where(rock lt 0.0))=0.0
      if (max(rock) gt 1.0) then rock(where(rock gt 1.0))=1.0

      ; output rock file
      ES3_openw,10,string(prefix,'rock.pic')
      writeu,10,rock
      close,10

      ; open other file
      ES3_openr,1,string(prefix,'other.pic')
      other=fltarr(ns,nl)
      readu,1,other
      close,1

      ; cleanse other file
      if (min(other) lt 0.0) then other(where(other lt 0.0))=0.0
      if (max(other) gt 1.0) then other(where(other gt 1.0))=1.0

      ; output other file
      ES3_openw,10,string(prefix,'other.pic')
```

```
        writeu,10,other
        close,10

HELP, NAMES="*", OUTPUT=ES3_ENVIROMENT & ES3_LOG, LEAVE="modscag_cleanse", $
  ENVIROMENT=ES3_ENVIROMENT
END ; modscag_cleanse
```

D.2 Client internal log file

All the linage information was extracted from the script and can be found as exec elements in the log file included below. Additional information about the computer and the parameters is included at the start and bottom of the file.

```
<init time="20050522T234606Z" pid="31002" stime="20050522T234604Z"
    pstime="20050522T234256Z" ppid="30920" language="idl" user="haavar"
    hostname="spitting-duck.bren.ucsb.edu">
  <enviroment>
    <variable name="!DIR" value="/home/rsi/idl_6.1"/>
    <variable name="!PATH" value="/home/haavar/probulator//idl:
      /home/haavar/probulator//idl/wrappers:/home/rsi/idl_6.1/lib/hook:
      /home/rsi/idl_6.1/lib/macros:
      /home/rsi/idl_6.1/lib/obsolete:
      /home/rsi/idl_6.1/lib/utilities:
      /home/rsi/idl_6.1/lib/itools/components:
      /home/rsi/idl_6.1/lib/itools/framework:
      /home/rsi/idl_6.1/lib/itools/ui_widgets:
      /home/rsi/idl_6.1/lib/itools:
      /home/rsi/idl_6.1/lib/wavelet/source:
      /home/rsi/idl_6.1/lib/wavelet/data:
      /home/rsi/idl_6.1/lib:
      /home/rsi/idl_6.1/examples/data:
      /home/rsi/idl_6.1/examples/data_access/sdf:
      /home/rsi/idl_6.1/examples/data_access:
      /home/rsi/idl_6.1/examples/doc/dicom:
      /home/rsi/idl_6.1/examples/doc/itools:
      /home/rsi/idl_6.1/examples/doc:
      /home/rsi/idl_6.1/examples/HP_TIFF:
      /home/rsi/idl_6.1/examples/imsl:
      /home/rsi/idl_6.1/examples/misc:
      /home/rsi/idl_6.1/examples/project:
      /home/rsi/idl_6.1/examples/visual/utility:
      /home/rsi/idl_6.1/examples/visual:
      /home/rsi/idl_6.1/examples/widgets/wexmast:
      /home/rsi/idl_6.1/examples/widgets:
      /home/rsi/idl_6.1/examples/demo/demodata:
      /home/rsi/idl_6.1/examples/demo/demosrc:
      /home/rsi/idl_6.1/examples/demo/demoslideshows/slideshowsrc:
      /home/rsi/idl_6.1/examples/demo:
      /home/rsi/idl_6.1/examples:
      /home/haavar/painters example"/>
    <variable name="!VERSION.ARCH" value="x86"/>
    <variable name="!VERSION.OS" value="linux"/>
    <variable name="!VERSION.OS_FAMILY" value="unix"/>
    <variable name="!VERSION.RELEASE" value="6.1"/>
    <variable name="!VERSION.BUILD_DATE" value="Jul 14 2004"/>
    <variable name="!VERSION.MEMORY_BITS" value="       32"/>
    <variable name="!VERSION.FILE_OFFSET_BITS" value="       64"/>
  </enviroment>
  <mount-points>
    <mount share="dab15:/ed15/rsi" type="nfs">/home/rsi</mount>
    <mount share="dab15:/ed15/user69/haavar" type="nfs">/home/haavar</mount>
  </mount-points>
</init>
<enter region="modscag_cleanse">
  <enviroment>
```

```
    <variable type="UNDEFINED" name="GRNSZ" value="&lt;Undefined&gt;"/>
    <variable type="INT" name="NL" value="2"/>
    <variable type="INT" name="NS" value="2"/>
    <variable type="UNDEFINED" name="OTHER" value="&lt;Undefined&gt;"/>
    <variable type="STRING" name="PREFIX" value="data/till"/>
    <variable type="UNDEFINED" name="ROCK" value="&lt;Undefined&gt;"/>
    <variable type="UNDEFINED" name="SNOW" value="&lt;Undefined&gt;"/>
    <variable type="UNDEFINED" name="VEG" value="&lt;Undefined&gt;"/>
  </enviroment>
</enter>
<exec time="20050522T234610Z" routine="OPENR">
  <io>
    <file read="true">/home/haavar/painter/data/tillsnow.pic</file>
  </io>
</exec>
<exec time="20050522T234610Z" routine="OPENW">
  <io>
    <file>/home/haavar/painter/data/tillsnow.pic</file>
  </io>
</exec>
<exec time="20050522T234611Z" routine="OPENR">
  <io>
    <file read="true">/home/haavar/painter/data/tillgrnsz.pic</file>
  </io>
</exec>
<exec time="20050522T234612Z" routine="OPENW">
  <io>
    <file>/home/haavar/painter/data/tillgrnsz.pic</file>
  </io>
</exec>
<exec time="20050522T234613Z" routine="OPENR">
  <io>
    <file read="true">/home/haavar/painter/data/tillveg.pic</file>
  </io>
</exec>
<exec time="20050522T234613Z" routine="OPENW">
  <io>
    <file>/home/haavar/painter/data/tillveg.pic</file>
  </io>
</exec>
<exec time="20050522T234614Z" routine="OPENR">
  <io>
    <file read="true">/home/haavar/painter/data/tillrock.pic</file>
  </io>
</exec>
<exec time="20050522T234614Z" routine="OPENW">
  <io>
    <file>/home/haavar/painter/data/tillrock.pic</file>
  </io>
</exec>
<exec time="20050522T234615Z" routine="OPENR">
  <io>
    <file read="true">/home/haavar/painter/data/tillother.pic</file>
  </io>
</exec>
<exec time="20050522T234616Z" routine="OPENW">
  <io>
    <file>/home/haavar/painter/data/tillother.pic</file>
  </io>
</exec>
<leave region="modscag_cleanse">
  <enviroment>
    <variable type="FLOAT" name="GRNSZ" value="Array[2, 2]"/>
    <variable type="INT" name="NL" value="2"/>
    <variable type="INT" name="NS" value="2"/>
    <variable type="FLOAT" name="OTHER" value="Array[2, 2]"/>
    <variable type="STRING" name="PREFIX" value="data/till"/>
```

```
    <variable type="FLOAT" name="ROCK" value="Array[2, 2]"/>
    <variable type="FLOAT" name="SNOW" value="Array[2, 2]"/>
    <variable type="FLOAT" name="VEG" value="Array[2, 2]"/>
  </enviroment>
</leave>
```

D.3 ES^3 workflow message

The initial ES^3 message only contains the linage, and a lot of the additional information gets lost. This information is likely to be used in future versions of the ES^3 message. The ES^3 message below contains the linage retrieved from the test run of modscag_cleanse. The log contains enough information to reconstruct the DAG in figure D.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<workflow localID="20050522T234606Z-31002">
  <relation to="T-3" from="F-1"/>
  <relation to="T-1" from="F-0"/>
  <relation to="T-6" from="F-3"/>
  <relation to="T-7" from="F-3"/>
  <relation to="T-8" from="F-4"/>
  <relation to="T-2" from="F-1"/>
  <relation to="T-4" from="F-2"/>
  <relation to="T-9" from="F-4"/>
  <relation to="T-0" from="F-0"/>
  <relation to="T-5" from="F-2"/>
  <file name="/home/haavar/painter/data/tillsnow.pic" id="F-0"/>
  <file name="/home/haavar/painter/data/tillgrnsz.pic" id="F-1"/>
  <file name="/home/haavar/painter/data/tillveg.pic" id="F-2"/>
  <file name="/home/haavar/painter/data/tillrock.pic" id="F-3"/>
  <file name="/home/haavar/painter/data/tillother.pic" id="F-4"/>
  <transformation routine="OPENR" id="T-0"/>
  <transformation routine="OPENW" id="T-1"/>
  <transformation routine="OPENR" id="T-2"/>
  <transformation routine="OPENW" id="T-3"/>
  <transformation routine="OPENR" id="T-4"/>
  <transformation routine="OPENW" id="T-5"/>
  <transformation routine="OPENR" id="T-6"/>
  <transformation routine="OPENW" id="T-7"/>
  <transformation routine="OPENR" id="T-8"/>
  <transformation routine="OPENW" id="T-9"/>
</workflow>
```

This page is intentionally left blank.

Appendix E

Bash example workflow

To show the functionality of the bash plugin I have written a tiny bash script. I chose to write my own script to demonstrate the plugin since I thought the bash scripts I found that where a part of a real scientific workflows where too long to use in an example. When the scripts grow large it is harder for humans to interpret the output. When the system is running this data is submitted to the ES^3 core and no human needs to interpret the messages.

Figure E.1 shows the linage the plugin retrieved from this bash script. Each command executed in the bash script that is not a part of the scripting language is represented as a transformation. In this example we have information flowing between the transformations, and we have some that are missing input or output. In the script the output from sed was stored in a variable in the scripting environment, and this does not show up in the logs. The command echo was invoked in a subshell to print something to the screen, and did not take any inputs and therefore it shows up in the graph as a transformation with no inputs and not outputs.

When you run a script with the bash plugin connected you can expect a lot of transformations, and many of them will miss either input or output. Only a small number of outputs will be connected to each-other. This is by my opinion a good representation what happens when a bash script is executed, since it is not a lot of information that flows between the different commands.

E.1 Source code

```
#!/bin/bash

uid=`cat /etc/passwd | grep haavar | \
 sed -n 's/\(.*:\)\{2\}\([0-9]\+\).*/\2/p'`

if [ $uid -lt 500 ]
then
    /bin/echo You are a default user

else
  /bin/echo You are a normal user
fi
```

E.2 Client internal log files

This is the internal log files of the client and they show all the information recoded when the script ran. Not all this information is used in the message sent to the server, but this is information I expect that will be used in the future. There are two log files because bash

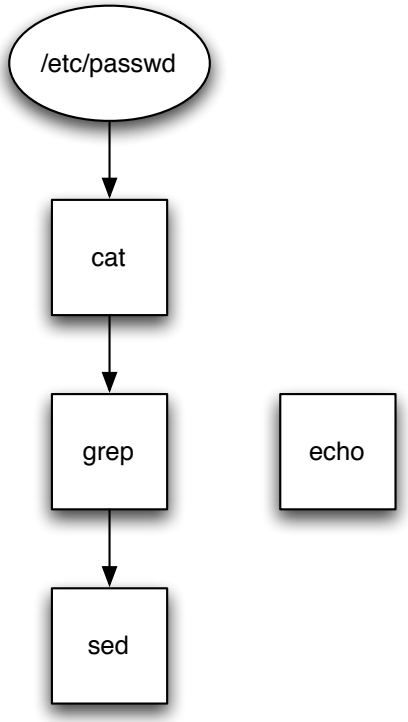

Figure E.1: A DAG representing the information flow of the bash test script

executed the body of the if-statement in a subprocess. If you look at the parent process id and the parent startup time attribute of the init element you can see that the second workflow is a part of the first. This relation needs to be determined by the transmitter before the workflow is sent to the server.

All the files opened by the process after the exec call was issued are included in the list of files. Some of the files are not data files read by the process, but libraries used by the application. It is up to the server to determine the relevance of the files.

E.2.1 20050528T225100.371521Z-6843

```
<init time="20050528T225101Z" pid="6840" stime="20050528T225100.371521Z"
    language="bash" user="haavar" hostname="spitting-duck.bren.ucsb.edu">
  <mount-points>
    <mount share="dab15:/ed15/user69/mtc" type="nfs">/home/mtc</mount>
    <mount share="dab15:/ed15/rsi" type="nfs">/home/rsi</mount>
    <mount share="dab15:/ed15/software" type="nfs">/home/software</mount>
    <mount share="dab15:/ed15/user69/stoms" type="nfs">/home/stoms</mount>
    <mount share="dab15:/ed15/user69/frew" type="nfs">/home/frew</mount>
    <mount share="dab15:/ed15/user69/haavar" type="nfs">/home/haavar</mount>
    <mount share="dab16:/ed16a/tm" type="nfs">/home/tm</mount>
    <mount share="dab15:/ed15/eil" type="nfs">/home/eil</mount>
  </mount-points>
</init>
<exec time="20050528T225100.371521Z" routine="/bin/cat"
    pid="6844" ppid="6843">
  <arguments>
    <argument>/etc/passwd</argument>
  </arguments>
  <io>
    <pipe read="true" id="std-in"/>
    <pipe write="true" id="2"/>
    <pipe write="true" id="std-err"/>
    <file read="true">/etc/ld.so.cache</file>
    <file read="true">/lib/tls/libc.so.6</file>
    <file read="true">/usr/lib/locale/locale-archive</file>
    <file read="true">/etc/passwd</file>
  </io>
</exec>
<exec time="20050528T225101.165831Z" routine="/bin/grep"
    pid="6845" ppid="6843">
  <arguments>
    <argument>haavar</argument>
  </arguments>
  <io>
    <pipe read="true" id="2"/>
    <pipe write="true" id="3"/>
    <pipe write="true" id="std-err"/>
    <file read="true">/etc/ld.so.cache</file>
    <file read="true">/lib/libpcre.so.0</file>
    <file read="true">/lib/tls/libc.so.6</file>
    <file read="true">/usr/lib/locale/locale-archive</file>
    <file read="true">/usr/lib/gconv/gconv-modules.cache</file>
    <file read="true">/usr/share/locale/locale.alias</file>
  </io>
</exec>
<exec time="20050528T225101.164442Z" routine="/bin/sed"
    pid="6846" ppid="6843">
  <arguments>
    <argument>-n</argument>
    <argument>s/\\(.*:\\)\\{2\\}\\([0-9]\\+\\).*/\\2/p</argument>
  </arguments>
  <io>
    <pipe read="true" id="3"/>
    <pipe write="true" id="1"/>
    <pipe write="true" id="std-err"/>
```

```
    <file read="true">/etc/ld.so.cache</file>
    <file read="true">/lib/tls/libc.so.6</file>
    <file read="true">/usr/lib/locale/locale-archive</file>
    <file read="true">/usr/lib/gconv/gconv-modules.cache</file>
  </io>
</exec>
```

E.2.2 20050528T225101.185212Z-6838

```
<init time="20050528T225101Z" pid="6840" stime="20050528T225101.185212Z"
    pstime="20050528T225100Z" ppid="6843" language="bash" user="haavar"
    hostname="spitting-duck.bren.ucsb.edu">
  <mount-points>
    <mount share="dab15:/ed15/user69/mtc" type="nfs">/home/mtc</mount>
    <mount share="dab15:/ed15/rsi" type="nfs">/home/rsi</mount>
    <mount share="dab15:/ed15/software" type="nfs">/home/software</mount>
    <mount share="dab15:/ed15/user69/stoms" type="nfs">/home/stoms</mount>
    <mount share="dab15:/ed15/user69/frew" type="nfs">/home/frew</mount>
    <mount share="dab15:/ed15/user69/haavar" type="nfs">/home/haavar</mount>
    <mount share="dab16:/ed16a/tm" type="nfs">/home/tm</mount>
    <mount share="dab15:/ed15/eil" type="nfs">/home/eil</mount>
  </mount-points>
</init>
<exec time="20050528T225101.185212Z" routine="/bin/echo"
    pid="6848" ppid="6838">
  <arguments>
    <argument>You</argument>
    <argument>are</argument>
    <argument>a</argument>
    <argument>normal</argument>
    <argument>user</argument>
  </arguments>
  <io>
    <pipe read="true" id="std-in"/>
    <pipe write="true" id="std-out"/>
    <pipe write="true" id="std-err"/>
    <file read="true">/etc/ld.so.cache</file>
    <file read="true">/lib/tls/libc.so.6</file>
    <file read="true">/usr/lib/locale/locale-archive</file>
  </io>
</exec>
```

E.3 ES^3 workflow message

The ES^3 message format is still in early development, and this is only the information we have decided to include so far. We will include more in the future, but we have not decided how to format the information. The messages shown below is to show the concept or reformatting the internal log to ES^3 messages.

```
<workflow localID="20050528T225100.371521Z-6843">
  <relation to="T-2" from="T-1"/>
  <relation to="T-2" from="F-6"/>
  <relation to="T-1" from="F-2"/>
  <relation to="T-2" from="F-1"/>
  <relation to="T-0" from="T-1"/>
  <relation to="T-0" from="F-3"/>
  <relation to="T-2" from="F-4"/>
  <relation to="T-2" from="T-1"/>
  <relation to="T-0" from="T-1"/>
  <relation to="T-2" from="F-5"/>
  <relation to="T-2" from="F-0"/>
  <relation to="T-1" from="F-1"/>
```

```
  <relation to="T-0" from="F-2"/>
  <relation to="T-2" from="T-1"/>
  <relation to="T-1" from="F-0"/>
  <relation to="T-0" from="F-1"/>
  <relation to="T-0" from="F-0"/>
  <relation to="T-1" from="F-5"/>
  <relation to="T-0" from="T-1"/>
  <relation to="T-2" from="F-2"/>
  <file name="/etc/ld.so.cache" id="F-0"/>
  <file name="/lib/tls/libc.so.6" id="F-1"/>
  <file name="/usr/lib/locale/locale-archive" id="F-2"/>
  <file name="/etc/passwd" id="F-3"/>
  <file name="/lib/libpcre.so.0" id="F-4"/>
  <file name="/usr/lib/gconv/gconv-modules.cache" id="F-5"/>
  <file name="/usr/share/locale/locale.alias" id="F-6"/>
  <transformation routine="/bin/cat" id="T-0">
    <arguments>
      <argument>/etc/passwd</argument>
    </arguments>
  </transformation>
  <transformation routine="/bin/sed" id="T-1">
    <arguments>
      <argument>-n</argument>
      <argument>s/\\(.*:\\)\\{2\\}\\([0-9]\\+\\).*/\\2/p</argument>
    </arguments>
  </transformation>
  <transformation routine="/bin/grep" id="T-2">
    <arguments>
      <argument>haavar</argument>
    </arguments>
  </transformation>
</workflow>
<workflow localID="20050528T225101.185212Z-6838">
  <relation to="T-0" from="F-0"/>
  <relation to="T-0" from="F-1"/>
  <relation to="T-0" from="F-2"/>
  <file name="/etc/ld.so.cache" id="F-0"/>
  <file name="/lib/tls/libc.so.6" id="F-1"/>
  <file name="/usr/lib/locale/locale-archive" id="F-2"/>
  <transformation routine="/bin/echo" id="T-0">
    <arguments>
      <argument>You</argument>
      <argument>are</argument>
      <argument>a</argument>
      <argument>normal</argument>
      <argument>user</argument>
    </arguments>
  </transformation>
</workflow>
```