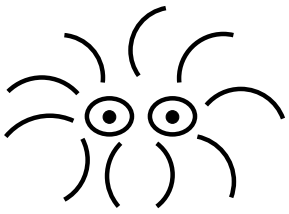
ANML Processor Manual



Cameron Kiddle

TeleSim Group
Department of Computer Science
University of Calgary

December 19, 2002

Contents

Chapter 1

Introduction

ANML, which stands for ANother Modelling Language, is a general purpose modelling language that can be used to describe various systems such as communication networks. A particular description of such a system is referred to as a model. Rules for creating models are specified in a structure called a schema. The ANML Processor is a tool used to process ANML models to determine if they are well-formed and valid. Models are well-formed if they match the ANML syntax and models are valid if they satisfy the rules specified in the schema. Definition of the ANML syntax and directions for creating schemas and models can be found in *The ANML Guide*. This document describes how to use the ANML Processor.

The ANML Processor can be used either as a stand-alone tool or as a C/C++ library. As a stand-alone tool, the ANML processor can be used to process models to determine if they are well-formed and valid. As a library, the ANML processor can be used by different applications to both process models and extract the model information. Examples of such applications might include simulators that use ANML models to describe simulation scenarios, or visualization tools used to display ANML models.

This manual continues by explaining how to use the ANML Processor as a stand-alone tool and as a library. The library interface is then described in detail, followed by an example application program that utilizes the ANML Processor. A quick reference guide to the library interface and the code for the example application program can be found in the Appendix.

Chapter 2

Using the ANML Processor

This chapter explains how to build the ANML Processor and how to use the ANML Processor as a stand-alone tool and as a library.

2.1 Building the ANML Processor

The ANML Processor can be built from the top level directory of the source code package. After entering the top level directory, enter the following at the command line prompt (denoted as '>'):

```
> ./configure
> gmake
> gmake install
```

Note that the GNU make utility `gmake` should be used as other versions of `make` may not be compatible. By default, the executable for the stand-alone processor, called `anml_proc`, will be installed in the `bin` directory of the top level directory of ANML Processor source code package. The library for the processor called `libchecker.a` will be installed in the `lib` directory of the top level directory. Enter `./configure --help` at the command prompt to learn how the paths for the `bin` and `lib` directories can be set differently.

2.2 Using the ANML Processor as a Stand-Alone Tool

The stand-alone ANML Processor just processes models to determine if they are well-formed and valid. It can be run on a model file by entering the following at the command line prompt:

```
> anml_proc -model <modelfile>
```

For example, if the name of the file containing the model is `test.anml` then the following should be entered at the command line prompt:

```
> anml_proc -model test.anml
```

If the model contained in test.anml is well-formed and valid then the following should be output by the ANML Processor:

```
Checking if model is well-formed ...
Model is well-formed.

Checking if model is valid ...
Model is valid.
```

If the model is not well-formed or valid, messages will be displayed indicating the nature and location of the errors.

To see additional command line arguments available enter the following at the command line prompt:

```
> anml_proc --help
```

The above command should produce the following output:

```
ANML Processor command line arguments:
  -model <ANML file>      file containing model to check
  [-disable_warnings]     disables warning messages
  [-well_formed_only]     checks only if model is well-formed
```

The `-model` argument must be given, but the `-disable_warnings` and `-well_formed_only` arguments are optional.

2.3 Using the ANML Processor as a Library

There is a both a C and C++ interface to the ANML Processor library. The C++ interface is essentially a wrapper of the C interface. If using the C interface the application should include the file `anml_c_manager.h` and if using the C++ interface the application should include the file `anml_manager.h`. The interface is described in detail in Chapter 3.

Chapter 3

ANML Processor Library Interface

The interface to the ANML Processor consists of a manager that allows traversal and access of model information. The manager is written in C (`anml_c_manager`) with a C++ wrapper class (`anml_manager`) also written for convenience. If using the C interface the application should include the file `anml_c_manager.h`. If using the C++ interface the application should include the file `anml_manager.h`. The C++ `anml_manager` class is described in the next subsection. The member functions used to traverse and access model information are then described in the remaining subsections. The corresponding C interface functions are given below the C++ member functions in *italics*. The only difference is that the C manager must be passed as an argument of the C interface functions.

3.1 Class anml_manager (anml_c_manager)

To process, traverse and access model information an instance of the `anml_manager` class must be created when using the C++ interface. When using the C interface an instance of the `anml_c_manager` struct must be created. The `anml_manager` class specification is given below:

```
class anml_manager {

 public:
  anml_manager( );
  anml_manager( anml_c_manager * );
  anml_manager( anml_manager * );
  ~anml_manager();

  /* functions for processing models */
  int process_model_file( const char * );

  /* functions for writing out models */
  int write_cur_model( FILE * );

  /* functions to traverse components */
  int cur_comp();
  int next_comp();
  int prev_comp();
```

```
int first_sub_comp();
int last_sub_comp();
int parent_comp();
int reset_first_comp( int );
int reset_last_comp( int );
int lev_reset_first_comp();
int lev_reset_last_comp();
int comp_find_forward( const char * );
int comp_find_backward( const char * );
int lookup_comp( const char * );

/* functions to traverse attributes of cur comp */
int cur_atr();
int next_atr();
int prev_atr();
int first_sub_atr();
int last_sub_atr();
int parent_atr();
int reset_first_atr();
int reset_last_atr();
int lev_reset_first_atr();
int lev_reset_last_atr();
int atr_find_forward( const char * );
int atr_find_backward( const char * );

/* functions to traverse values of cur atr */
int cur_atr_val();
int next_atr_val();
int prev_atr_val();
int reset_first_atr_val();
int reset_last_atr_val();

/* functions for accessing manager information */
anml_c_manager * c_manager();

/* functions to access cur model information */
int cur_model__name( char ** );
int cur_model__schema_name( char ** );

/* functions to access cur comp information */
int cur_comp__id( char ** );
int cur_comp__abs_id( char ** );
int cur_comp__name( char ** );
int cur_comp__class( char ** );
int cur_comp__app_class( char ** );
int cur_comp_isa( const char *, int * );
```

```
  /* functions for accessing cur atr information */
  int cur_atr__name( char ** );
  int cur_atr__composite_name( char ** );
  int cur_atr__type( sch_atr_type * );
  int cur_atr__mod_atr_type( mod_attribute_type * );
  int conv_cur_atr_to_id_list();

  /* functions for accessing cur atr val information */
  int cur_atr_val__string( char ** );
  int cur_atr_val__integer( int * );
  int cur_atr_val__real( double * );
  int cur_atr_val__boolean( int * );
  int cur_atr_val__abs_id( char ** );

  /* functions for setting anml_manager functionality */
  void set_print_disabled( int );
  void set_warnings_disabled( int );

 private:
  anml_c_manager * _manager;

};
```

The `anml_manager` class contains a pointer to the `anml_c_manager` object used by the C interface. All of the member functions call the corresponding C interface functions using the C manager. Most member functions return an `int` to indicate if the call was successful or not. If no errors were encountered and the call is successful `ANML_MANAGER_SUCCESS` is returned. If errors were encountered or the call was not successful `ANML_MANAGER_FAILURE` is returned. The member functions are described in the subsections to follow.

3.2 Creating, Copying and Deleting ANML Managers

This subsection describes the functions for creating, copying and deleting ANML managers.

anml_manager::anml_manager()
*anml_c_manager * init_anml_c_manager(void)*

This function creates a new `anml_manager` which can be used to process a new model file and then to traverse and access the model information.

anml_manager::anml_manager(anml_c_manager * manager)

This function creates a copy of the C `anml_c_manager` passed in as an argument and wraps it in a C++ `anml_manager`. Only the iterators and other information pertaining to the manager are copied. Copies of a manager should only be made after the model file has been processed. Otherwise, all manager copies may not have access to the model data.

anml_manager::anml_manager(anml_manager * manager)
*anml_c_manager * copy_anml_c_manager(const anml_c_manager * manager)*

This function creates a copy of the `anml_manager`. Note that the model data is not copied. Only the iterators and other information pertaining to the manager are copied. Copies of a manager should only be made after the model file has been processed. Otherwise, all manager copies may not have access to the model data.

anml_manager::~anml_manager()
*void delete_anml_c_manager(anml_c_manager * manager)*

This function deletes the anml manager. Reference counts to copies of managers are kept to ensure that model data is not deleted until all copies have been deleted.

3.3 Processing ANML Models

This subsection describes the function used to process ANML models.

int anml_manager::process_model_file(const char * filename)
*int am_process_model_file(anml_c_manager * manager, const char * filename)*

This function processes the model in the file with the specified `filename`. If the model is well formed and valid the current model of the manager is set to the processed model and `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE is returned`. The manager has been designed so that a single manager instance may process and manage multiple models. Currently, only one model is supported so a new `anml_manager` should be created for each model to be processed.

3.4 Writing ANML Models

This subsection describes the functions used to write ANML models out to a file.

int anml_manager::write_cur_model(FILE * out)
*int am_write_cur_model(FILE * out, anml_c_manager * manager)*

This function writes the current model to the `out` file passed in as an argument. The model is written out in full without the use of databases. The `_include` key-value pair for the schema will need to be added manually afterwards. If the current model is NULL `ANML_MANAGER_FAILURE` is returned. Otherwise, `ANML_MANAGER_SUCCESS` is returned.

3.5 Traversing Components

This subsection describes the functions used to traverse components of the current model. Before traversing components, `reset_first_comp` or `reset_last_comp` must be called with the desired traversal method to setup the components for traversal. Failing to call one of these functions first will result in an `ANML PROCESSOR FATAL ERROR` which causes the program to be exited. The components can be traversed using one of three methods: `ANML_MANAGER_PREORDER`, `ANML_MANAGER_POSTORDER` or `ANML_MANAGER_FREEORDER`. In the `ANML_MANAGER_PREORDER` method, components are traversed in pre-order, which is the component first and then the sub-components from left to right. In the `ANML_MANAGER_POSTORDER` method, components are traversed in post-order, which is the sub-components first from left to right and then the component. In the `ANML_MANAGER_FREEORDER` method, the user is in control of the order in which the components are traversed.

int anml_manager::cur_comp()
*int am_cur_comp(anml_c_manager * manager)*

This function returns `ANML_MANAGER_SUCCESS` if the current component of the current model is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned.

int anml_manager::next_comp()
*int am_next_comp(anml_c_manager * manager)*

This function moves to the next component of the current model, according to the traversal method currently selected, and returns `ANML_MANAGER_SUCCESS` if the next component is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned. For the `ANML_MANAGER_FREEORDER` traversal method, the next component is the next sub-component of the current component's parent.

int anml_manager::prev_comp()
*int am_prev_comp(anml_c_manager * manager)*

This function moves to the previous component of the current model, according to the traversal method currently selected, and returns `ANML_MANAGER_SUCCESS` if the previous component is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned. For the `ANML_MANAGER_FREEORDER` traversal method, the previous component is the previous sub-component of the current component's parent.

int anml_manager::first_sub_comp()
*int am_first_sub_comp(anml_c_manager * manager)*

This function moves to the first sub-component of the current component. `ANML_MANAGER_SUCCESS` is returned if the first sub-component is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned. This function may only be called if using the `ANML_MANAGER_FREEORDER` traversal method.

int anml_manager::last_sub_comp()
*int am_last_sub_comp(anml_c_manager * manager)*

This function moves to the last sub-component of the current component. `ANML_MANAGER_SUCCESS`

is returned if the last sub-component is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned. This function may only be called if using the `ANML_MANAGER_FREEORDER` traversal method.

int anml_manager::parent_comp()
*int am_parent_comp(anml_c_manager * manager)*

This function moves to the parent component of the current component. ANML_MANAGER_SUCCESS is returned if the parent component is not NULL. Otherwise, ANML_MANAGER_FAILURE is returned. This function may only be called if using the ANML_MANAGER_FREEORDER traversal method.

int anml_manager::reset_first_comp(int comp_traversal_type)
*int am_reset_first_comp(anml_c_manager * manager, int comp_traversal_type)*

This function resets the current component to the first component of the current model according to the specified comp_traversal_method. ANML_MANAGER_SUCCESS is returned if the first component is not NULL. Otherwise, ANML_MANAGER_FAILURE is returned. Their are three different traversal methods to choose from: ANML_MANAGER_PREORDER, ANML_MANAGER_POSTORDER or ANML_MANAGER_FREEORDER. These traversal methods are described at the beginning of Section 3.5. For the ANML_MANAGER_PREORDER and ANML_MANAGER_FREEORDER traversal methods, the first component is the first top level component of the model tree. For the ANML_MANAGER_POSTORDER traversal method the, first component is the left most bottom component in the model tree. Either reset_first_comp or reset_last_comp must be called before traversing the components.

int anml_manager::reset_last_comp(int comp_traversal_type)
*int am_reset_last_comp(anml_c_manager * manager, int comp_traversal_type)*

This function resets the current component to the very last component of the current model according to the specified comp_traversal_method. ANML_MANAGER_SUCCESS is returned if the last component is not NULL. Otherwise, ANML_MANAGER_FAILURE is returned. Their are three different traversal methods to choose from: ANML_MANAGER_PREORDER, ANML_MANAGER_POSTORDER or ANML_MANAGER_FREEORDER. These traversal methods are described at the beginning of Section 3.5. For the ANML_MANAGER_PREORDER traversal method, the last component is the right most bottom component in the model tree. For the ANML_MANAGER_POSTORDER and ANML_MANAGER_FREEORDER traversal methods, the last component is the last top level component of the model tree. Either reset_first_comp or reset_last_comp must be called before traversing the components.

int anml_manager::lev_reset_first_comp()
*int am_lev_reset_first_comp(anml_c_manager * manager)*

This function resets the current component to the first sub-component of the current component's parent. ANML_MANAGER_SUCCESS is returned if the first sub-component of the current component's parent is not NULL. Otherwise, ANML_MANAGER_FAILURE is returned. This function may only be called if using the ANML_MANAGER_FREEORDER traversal method.

int anml_manager::lev_reset_last_comp()
*int am_lev_reset_last_comp(anml_c_manager * manager)*

This function resets the current component to the last sub-component of the current component's parent. ANML_MANAGER_SUCCESS is returned if the last sub-component of the current component's parent is not NULL. Otherwise, ANML_MANAGER_FAILURE is returned. This function may only be

called if using the ANML_MANAGER_FREEORDER traversal method.

int anml_manager::comp_find_forward(const char * class_name)
*int am_comp_find_forward(anml_c_manager * manager, const char * class_name)*

Starting with the current component, and traversing forward according to the traversal method currently selected, this function finds the first component that is an instance of the class specified by the class_name argument. If a component is found ANML_MANAGER_SUCCESS is returned. Otherwise, ANML_MANAGER_FAILURE is returned.

int anml_manager::comp_find_backward(const char * class_name)
*int am_comp_find_backward(anml_c_manager * manager, const char * class_name)*

Starting with the current component, and traversing backward according to the traversal method currently selected, this function finds the first component that is an instance of the class specified by the class_name argument. If a component is found ANML_MANAGER_SUCCESS is returned. Otherwise, ANML_MANAGER_FAILURE is returned.

int anml_manager::lookup_comp(const char * abs_id)
*int am_lookup_comp(anml_c_manager * manager, const char * abs_id)*

This function sets the current component to be the component of the current model with the absolute identifier specified by the abs_id argument. If a component with the specified absolute identifier is found, ANML_MANAGER_SUCCESS is returned. Otherwise, ANML_MANAGER_FAILURE is returned. The absolute identifier of a component is its complete hierarchical identifier starting from the top level of the model. This function sets the current traversal method to be ANML_MANAGER_FREEORDER and does not require reset_first_comp or reset_last_comp to be called first.

3.6 Traversing Attributes of Current Component

This subsection describes the functions used to traverse attributes of the current component. Before traversing attributes, reset_first_atr or reset_last_atr must be called to setup the attributes for traversal. Failing to call one of these functions first will result in an ANML PROCESSOR FATAL ERROR which causes the program to be exited.

int anml_manager::cur_atr()
*int am_cur_atr(anml_c_manager * manager)*

This function returns ANML_MANAGER_SUCCESS if the current attribute of the current component is not NULL. Otherwise, ANML_MANAGER_FAILURE is returned.

int anml_manager::next_atr()
*int am_next_atr(anml_c_manager * manager)*

This function moves to the next attribute of the current component. ANML_MANAGER_SUCCESS is returned if the next attribute is not NULL. Otherwise, ANML_MANAGER_FAILURE is returned.

int anml_manager::prev_atr()
*int am_prev_atr(anml_c_manager * manager)*

This function moves to the previous attribute of the current component. `ANML_MANAGER_SUCCESS` is returned if the previous attribute is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned.

int anml_manager::first_sub_atr()
*int am_first_sub_atr(anml_c_manager * manager)*

This function moves to the first sub-attribute of the current attribute. `ANML_MANAGER_SUCCESS` is returned if the first sub-attribute is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned. This function may only be called if current attribute is a composite attribute. Composite attributes have a `mod_atr_type` of `MOD_COMP_ATR`.

int anml_manager::last_sub_atr()
*int am_last_sub_atr(anml_c_manager * manager)*

This function moves to the last sub-attribute of the current attribute. `ANML_MANAGER_SUCCESS` is returned if the last sub-attribute is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned. This function may only be called if current attribute is a composite attribute. Composite attributes have a `mod_atr_type` of `MOD_COMP_ATR`.

int anml_manager::parent_atr()
*int am_parent_atr(anml_c_manager * manager)*

This function moves to the parent attribute of the current attribute. `ANML_MANAGER_SUCCESS` is returned if the parent attribute is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned.

int anml_manager::reset_first_atr()
*int am_reset_first_atr(anml_c_manager * manager)*

This function resets the current attribute to the first attribute of the current component. If the first attribute is not NULL, `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE` is returned. Either `reset_first_atr` or `reset_last_atr` must be called before traversing the attributes of the current component.

int anml_manager::reset_last_atr()
*int am_reset_last_atr(anml_c_manager * manager)*

This function resets the current attribute to the last attribute of the current component. If the last attribute is not NULL, `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE` is returned. Either `reset_first_atr` or `reset_last_atr` must be called before traversing the attributes of the current component.

int anml_manager::lev_reset_first_atr()
*int am_lev_reset_first_atr(anml_c_manager * manager)*

This function resets the current attribute to the first sub-attribute of the current attribute's parent. If the first sub-attribute of the current attribute's parent is not NULL, `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE` is returned. This function is intended for use with composite attributes.

int anml_manager::lev_reset_last_atr()
*int am_lev_reset_last_atr(anml_c_manager * manager)*

This function resets the current attribute to the last sub-attribute of the current attribute's parent. If the last sub-attribute of the current attribute's parent is not NULL, `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE` is returned. This function is intended for use with composite attributes.

int anml_manager::atr_find_forward(const char * atr_name)
*int am_atr_find_forward(anml_c_manager * manager, const char * atr_name)*

Starting with the current attribute, and traversing forward on the same attribute level, this function finds the attribute with the name specified by the `atr_name` argument. If the attribute is found, `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE` is returned.

int anml_manager::atr_find_backward(const char * atr_name)
*int am_atr_find_backward(anml_c_manager * manager, const char * atr_name)*

Starting with the current attribute, and traversing backward on the same attribute level, this function finds the attribute with the name specified by the `atr_name` argument. If the attribute is found, `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE` is returned.

3.7 Traversing Values of Current Attribute

This subsection describes the functions used to traverse attribute values of the current attribute. With the exception of `cur_atr_val`, these functions may only be called on attributes with a list of values. These attributes have a `MOD_ATR_TYPE` of `MOD_VAL_LIST_ATR`. Before traversing attribute values, `reset_first_atr` or `reset_last_atr` must be called to setup the attribute values for traversal. Failing to call one of these functions first, will result in an `ANML PROCESSOR FATAL ERROR` which causes the program to be exited.

int anml_manager::cur_atr_val()
*int am_cur_atr_val(anml_c_manager * manager)*

This function returns `ANML_MANAGER_SUCCESS` if the current attribute value of the current attribute is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned.

int anml_manager::next_atr_val()
*int am_next_atr_val(anml_c_manager * manager)*

This function moves to the next attribute value of the current attribute. `ANML_MANAGER_SUCCESS` is returned if the next attribute value is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned. This function may only be called if the current attribute has a list of values. The `MOD_ATR_TYPE` of such an attribute is `MOD_VAL_LIST_ATR`.

int anml_manager::prev_atr_val()
*int am_prev_atr_val(anml_c_manager * manager)*

This function moves to the previous attribute value of the current attribute. `ANML_MANAGER_SUCCESS` is returned if the previous attribute value is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned. This function may only be called if the current attribute has a list of values. The `MOD_ATR_TYPE` of such an attribute is `MOD_VAL_LIST_ATR`.

int anml_manager::reset_first_atr_val()
*int am_reset_first_atr_val(anml_c_manager * manager)*

This function resets the current attribute value to the first attribute value of the current attribute. If the first attribute value is not NULL, `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE` is returned. This function may only be called if the current attribute has a list of values. The `MOD_ATR_TYPE` of such an attribute is `MOD_VAL_LIST_ATR`. Either `reset_first_atr_val` or `reset_last_atr_val` must be called before traversing the attribute values of the current attribute.

int anml_manager::reset_last_atr_val()
*int am_reset_last_atr_val(anml_c_manager * manager)*

This function resets the current attribute value to the last attribute value of the current attribute. If the last attribute value is not NULL, `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE` is returned. This function may only be called if the current attribute has a list of values. The `MOD_ATR_TYPE` of such an attribute is `MOD_VAL_LIST_ATR`. Either `reset_first_atr_val` or `reset_last_atr_val` must be called before traversing the attribute values of the current attribute.

3.8 Accessing Manager Information

This subsection describes the functions that access data members of an `anml_manager`.

anml_c_manager * c_manager()

This function returns the C manager that this C++ manager is a wrapper for.

3.9 Accessing Current Model Information

This subsection describes the functions that access information pertaining to the current model. Note that the requested information is returned using an argument and that the return values for the functions are `ANML_MANAGER_SUCCESS`, if the current model is not NULL and `ANML_MANAGER_FAILURE` otherwise.

int anml_manager::cur_model__name(char ** model_name)
*int get_am_cur_model__name(anml_c_manager * manager, char ** model_name)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*model_name` to the `(char *)` pointer of the name of the current model, if the current model is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*model_name` is set to NULL.

int anml_manager::cur_model__schema_name(char ** schema_name)
*int get_am_cur_model__schema_name(anml_c_manager * manager, char ** schema_name)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*schema_name` to the `(char *)` pointer of the name of the schema used by the current model, if the current model is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*schema_name` is set to NULL.

3.10 Accessing Current Component Information

This subsection describes the functions that access information pertaining to the current component. Either `reset_first_comp` or `reset_last_comp` must be called before calling functions to access current component information. Failure to do so will result in an `ANML PROCESSOR FATAL ERROR` which causes the program to be exited. Note that the requested information is returned using an argument and that the return values for the functions are `ANML_MANAGER_SUCCESS`, if the current component is not NULL and `ANML_MANAGER_FAILURE` otherwise.

int anml_manager::cur_comp__id(char ** id)
*int get_am_cur_comp__id(anml_c_manager * manager, char ** id)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*id` to the `(char *)` pointer of the level id of the current component, if the current component is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*id` is set to NULL.

int anml_manager::cur_comp__abs_id(char ** abs_id)
*int get_am_cur_comp__abs_id(anml_c_manager * manager, char ** abs_id)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*abs_id` to a `(char *)` pointer of the absolute identifier of the current component, if the current component is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*abs_id` is set to NULL. The absolute identifier of a component is its hierarchical identifier beginning at the top level of the model. The memory pointed to by `*abs_id` will be overwritten the next time this function or `cur_atr_val__abs_id` is called, so the returned result should be copied or used before either of these functions are called again.

int anml_manager::cur_comp__name(char ** name)
*int get_am_cur_comp__name(anml_c_manager * manager, char ** name)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*name` to the `(char *)` pointer of the name of the current component, if the current component is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*name` is set to NULL. The name of the component is the name of the class the component belongs to if directly instantiated or the name of the database component that was used to instantiate the component.

int anml_manager::cur_comp__class(char ** class_name)
*int get_am_cur_comp__class(anml_c_manager * manager, char ** class_name)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*class_name` to the `(char *)` pointer of the class name of the current component, if the current component is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*class_name` is set to NULL.

int anml_manager::cur_comp__app_class(char ** app_class_name)
*int get_am_cur_comp__app_class(anml_c_manager * manager, char ** app_class_name)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*app_class_name` to the `(char *)` pointer of the application class name of the current component, if the current component is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*app_class_name` is set to NULL. Note, if the application class name was not set in the schema then `*app_class_name` will be NULL.

int anml_manager::cur_comp_isa(const char * class_name, int * isa)
*int am_cur_comp_isa(anml_c_manager * manager, const char * class_name, int * isa)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*isa` to 1, if the current component is an instance of the class specified by the `class_name` argument, or to 0 if not, if the current component is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*isa` is set to 0.

3.11 Accessing Current Attribute Information

This subsection describes the functions that access information pertaining to the current attribute. Either `reset_first_atr` or `reset_last_atr` must be called before calling functions to access current attribute information. Failure to do so will result in an `ANML PROCESSOR FATAL ERROR` which causes the program to be exited. Note that the requested information is returned using an argument and that the return values for the functions are `ANML_MANAGER_SUCCESS`, if the current attribute is not NULL and `ANML_MANAGER_FAILURE` otherwise.

int anml_manager::cur_atr__name(char ** name)
*int get_am_cur_atr__name(anml_c_manager * manager, char ** name)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*name` to the `(char *)` pointer of the name of the current attribute, if the current attribute is not NULL. Otherwise, `ANML_MANAGER_FAILURE`

is returned and `*name` is set to NULL.

int anml_manager::cur_atr__composite_name(char ** composite_name)
*int get_am_cur_atr__composite_name(anml_c_manager * manager, char * composite_name)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*composite_name` to the `(char *)` pointer of the composite name of the current attribute, if the current attribute is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*composite_name` is set to NULL. The composite name of an attribute includes the names of its parent attributes, separated by '.'s, starting with the highest level parent. For example if `b` is a sub-attribute of `a`, then the composite name of `b` is '`a.b`'.

int anml_manager::cur_atr__type(sch_atr_type * type)
*int get_am_cur_atr__type(anml_c_manager * manager, sch_atr_type * type)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*type` to the attribute type, if the current attribute is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*type` is set to `SCH_INVALID`. The `sch_atr_type` values are specified in Table 3.1.

Attribute Type	`sch_atr_type` Value
`integer`	`SCH_INTEGER`
`real`	`SCH_REAL`
`boolean`	`SCH_BOOLEAN`
`string`	`SCH_STRING`
`comp_atr`	`SCH_COMP_ATR`
`id_type`	`SCH_ID_TYPE`
`integer_list`	`SCH_INTEGER_LIST`
`real_list`	`SCH_REAL_LIST`
`boolean_list`	`SCH_BOOLEAN_LIST`
`string_list`	`SCH_STRING_LIST`

Table 3.1: `sch_atr_type` Values

int anml_manager::cur_atr__mod_atr_type(mod_attribute_type * mod_atr_type)
*int get_am_cur_atr__mod_atr_type(anml_c_manager * manager, mod_attribute_type * mod_atr_type)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*mod_atr_type` to the `mod_attribute_type`, if the current attribute is not NULL. Otherwise, `*mod_atr_type` is set to `MOD_INVALID_ATR` and `ANML_MANAGER_FAILURE` is returned. This additional attribute type field is given because an attribute of the `id_type` may have a single value, a list of values or may be composite having a range of values. The `mod_attribute_type` values are specified in Table 3.2.

int anml_manager::conv_cur_atr_to_id_list()
*int am_conv_cur_atr_to_id_list(anml_c_manager * manager)*

This function may only be called on attributes of the `id_type`. If the current attribute is not NULL, `ANML_MANAGER_SUCCESS` is returned and the `id_type` attribute is converted to a list of identifiers,

Attribute Type	mod_attribute_type Value
Single Value	MOD_VAL_ATR
Value List	MOD_VAL_LIST_ATR
Composite	MOD_COMP_ATR

Table 3.2: mod_attribute_type Values

such that the mod_attribute_type will be MOD_VAL_LIST_ATR. Otherwise, ANML_MANAGER_FAILURE is returned. This function is useful in dealing with attributes of the id_type so that all three mod_attribute_type cases do not have to be dealt with separately.

3.12 Accessing Current Attribute Value Information

This subsection describes the functions that access information pertaining to the current attribute value. These functions may only be called on attributes with a mod_attribute_type of MOD_VAL_ATR or MOD_VAL_LIST_ATR. Additionally, for attributes with mod_attribute_type of MOD_VAL_LIST_ATR, either reset_first_atr_val or reset_last_atr_val must be called before calling functions to access current attribute value information. Failure to do so will result in an ANML PROCESSOR FATAL ERROR which causes the program to be exited. Note that the requested information is returned using an argument and that the return values for the functions are ANML_MANAGER_SUCCESS, if the current attribute value is not NULL and ANML_MANAGER_FAILURE otherwise.

int anml_manager::cur_atr_val__string(char ** string_val)
*int get_am_cur_atr_val__string(anml_c_manager * manager, char ** string_val)*

This function returns ANML_MANAGER_SUCCESS and sets *string_val to the (char *) pointer of the string value of the current attribute value, if the current attribute value is not NULL. Otherwise, ANML_MANAGER_FAILURE is returned and *string_val is set to NULL. The string value can be returned for attributes of all sch_atr_types except for SCH_COMP_ATR.

int anml_manager::cur_atr_val__integer(int * int_val)
*int get_am_cur_atr_val__integer(anml_c_manager * manager, int * int_val)*

This function returns ANML_MANAGER_SUCCESS and sets *int_val to the integer value of the current attribute value, if the current attribute value is not NULL. Otherwise, ANML_MANAGER_FAILURE is returned and *int_val is set to 0. This function can only be called if the attribute has a sch_atr_type of SCH_INTEGER or SCH_INTEGER_LIST.

int anml_manager::cur_atr_val__real(double * real_val)
*int get_am_cur_atr_val__real(anml_c_manager * manager, double * real_val)*

This function returns ANML_MANAGER_SUCCESS and sets *real_val to the real value of the current attribute value, if the current attribute value is not NULL. Otherwise, ANML_MANAGER_FAILURE is returned and *real_val is set to 0. This function can only be called if the attribute has a sch_atr_type of SCH_REAL or SCH_REAL_LIST.

int anml_manager::cur_atr_val__boolean(int * bool_val)
*int get_am_cur_atr_val__boolean(anml_c_manager * manager, int * bool_val)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*bool_val` to 1 if true and 0 if false, if the current attribute value is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*bool_val` is set to 0. This function can only be called if the attribute has a `sch_atr_type` of `SCH_BOOLEAN` or `SCH_BOOLEAN_LIST`.

int anml_manager::cur_atr_val__abs_id(char ** abs_id)
*int get_am_cur_atr_val__abs_id(anml_c_manager * manager, char ** abs_id)*

This function returns `ANML_MANAGER_SUCCESS` and sets `*abs_id` to a `char *` pointer of the absolute identifier value, if the current attribute value is not NULL. Otherwise, `ANML_MANAGER_FAILURE` is returned and `*abs_id` is set to NULL. The absolute identifier of a component is its hierarchical identifier beginning at the top level of the model. The memory pointed to by `*abs_id` will be overwritten the next time this function or `cur_comp__abs_id` is called so the returned result should be copied or used before either of these functions are called again. This function can only be called if the attribute has a `sch_atr_type` of `id_type`.

3.13 Setting Manager Functionality

These functions allow the setting of various manager functionalities.

void anml_manager::set_print_disabled(int print_disabled)
*void set_am__print_disabled(anml_c_manager * manager, int print_disabled)*

This function allows ANML print statements to be disabled by passing in 1 as an argument, or enabled by passing in 0 as an argument. By default, ANML print statements are enabled. ANML print statements are output when a model is processed to indicate the progress of the processor.

void anml_manager::set_warnings_disabled(int warnings_disabled)
*void set_am__warnings_disabled(anml_c_manager * manager, int warnings_disabled)*

This function allows ANML warning messages to be disabled by passing in 1 as an argument, or enabled by passing in 0 as an argument. By default, ANML warning messages are enabled. ANML warning messages are output if potential problems with a model are found. Even with warnings, a model will be well formed and valid as long as no errors are detected.

Chapter 4

Example Program

This section describes portions of a simple C++ program (`net_proc.C`) that uses the ANML Processor. The program uses an `anml_manager` to process, traverse and display information of ANML models that utilize the `NetSchema`, a schema that can be used for the description of simple communication networks. For further information on the `NetSchema` refer to The ANML Guide where it is used as an example schema. The full code for the program is given in Appendix B. For the code given in this section line numbers are included for easy reference purposes.

As the program is utilizing the C++ `anml_manager` the file `anml_manager.h` must be included.

```
22>  #include "anml_manager.h"
```

The `main` function for the program is given below:

```
29>  int
30>  main( int argc, char * argv[] )
31>  {
32>    anml_manager * manager = NULL;
33>    int i = 0;
34>    char * model_filename = NULL;
35>
36>    /* extract model filename to process from command line arguments */
37>    for( i = 0; i < argc; ++i ) {
38>      if( strcmp( argv[i], "-model" ) == 0 )
39>        break;
40>    }
41>    ++i;
42>    if( i < argc ) {
43>      model_filename = argv[i];
44>    }
45>    else {
46>      fprintf(stderr,"-model <filename> must be specified.\n" );
47>      exit( 1 );
48>    }
49>
50>    /* create the anml_manager and process the model file */
51>    manager = new anml_manager();
```

```
52>    if( (manager->process_model_file( model_filename )) !=
53>        ANML_MANAGER_SUCCESS )
54>      exit( 1 );
55>
56>    /* traverse and display information on the different model components */
57>    traverse_hosts( manager );
58>    traverse_routers( manager );
59>    traverse_p2p_links( manager );
60>    traverse_lan_links( manager );
61>
62>    /* clean up */
63>    delete manager;
64>
65>    exit( 0 );
66>  }
```

On lines 37 to 48 the name of the file containing the ANML model is extracted from the command line arguments. A new `anml_manager` is created on line 50. The model file is then processed on lines 52 to 54. The member function `process_model_file` will return `ANML_MANAGER_SUCCESS` if the model was found to be both well-formed and valid. If errors were detected `ANML_MANAGER_FAILURE` will be returned. Functions are then called to traverse different model components on lines 57 to 60. The manager is deleted on line 63.

The functions for traversing and displaying information on components are fairly similar. The function for traversing components that are instances of the class `Host` is given below as an example.

```
68>  void
69>  traverse_hosts( anml_manager * manager )
70>  {/* Traverses and displays information on all components that are instances
71>    * of the class Host */
72>
73>    char * abs_id = NULL;
74>    int buffer_size = 0;
75>    char * lan_link_abs_id = NULL;
76>
77>
78>    fprintf( stdout, "Here are the Hosts:\n" );
79>    fprintf( stdout, "--------------------\n\n" );
80>
81>    manager->reset_first_comp( ANML_MANAGER_PREORDER );
82>
83>    while( manager->comp_find_forward( "Host" ) == ANML_MANAGER_SUCCESS ) {
84>      manager->cur_comp__abs_id( &abs_id );
85>      fprintf( stdout, "%s\n", abs_id );
86>
87>      manager->reset_first_atr();
88>      if( manager->atr_find_forward( "buffer_size" ) != ANML_MANAGER_SUCCESS ) {
89>        fprintf( stderr, "Failed to find 'buffer_size' attribute.\n" );
90>        exit( 1 );
```

```
91>      }
92>      manager->cur_atr_val__integer( &buffer_size );
93>      fprintf( stdout, "   buffer_size: %d Bytes\n", buffer_size );
94>
95>      manager->reset_first_atr();
96>      if( manager->atr_find_forward( "lan_link" ) == ANML_MANAGER_SUCCESS ) {
97>        manager->cur_atr_val__abs_id( &lan_link_abs_id );
98>        fprintf( stdout, "   lan_link: %s\n", lan_link_abs_id );
99>      }
100>
101>     fprintf( stdout, "\n" );
102>     manager->next_comp();
103>   }
104>   fprintf( stdout, "\n" );
105> }
```

Before traversing components the components must be reset as shown on line 81 using the member function `reset_first_comp`. This resets the current component pointer to the first component of a specified traversal order. Components can be traversed in pre-order, post-order or free-order as described in Section 3.5. In this example the components are set to be traversed in pre-order. The `while` loop on lines 83 to 103 traverses all of the components that are instances of the class `Host`. The member function `comp_find_forward` (line 83) finds the first component in the forward traversal direction, starting with the current component, that is an instance of the specified class. If a component of the specified class is found, `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE` is returned. On line 102 the next component is traversed to using the member function `next_comp`.

The absolute identifier of the Host is extracted using the member function `cur_comp__abs_id` on line 84. An absolute identifier is the full hierarchical identifier beginning at the top level of the model. The returned identifier should be used immediately or copied as the same memory location will be overwritten the next time a call to access an absolute identifier is made. The attributes of the host are then traversed on lines 87 to 99. As was the case for component traversal, the attributes must be reset before traversing attributes of the current component. On line 87 that current attribute pointer is reset to the first attribute of the current Host using the member function `reset_first_atr`. The member function `atr_find_forward` on line 88 searches the attributes of the current Host in the forward direction, starting with the current attribute, for the attribute with the name `buffer_size`. If the attribute is found, `ANML_MANAGER_SUCCESS` is returned. Otherwise, `ANML_MANAGER_FAILURE` is returned. The integer value of the `buffer_size` attribute, which is of the `integer` type, is retrieved on line 92 using the member function `cur_atr_val__integer`. On line 95 the attributes are reset to the first attribute and on line 96 a search is made for the `lan_link` attribute. Since this attribute is specified as an optional attribute in the `NetSchema`, the program is not exited if it is not found. The `lan_link` attribute is of the `id_type` and the absolute format of the identifier is retrieved on line 97 using the member function `cur_atr_val__abs_id`. The returned identifier should be used immediately or copied as the same memory location will be overwritten the next time a call to access an absolute identifier is made.

Instead of searching for each individual attribute separately, the attributes of the Host could be traversed just once via the member function `next_atr`. Actions could be performed based on the name of the attribute currently being traversed. For components with a long list of attributes this would definitely be the more efficient approach.

Appendix A

ANML Processor Library Interface Quick Reference

This section contains a quick reference to the C++ `anml_manager` member functions and the associated functions used with the C `anml_c_manager`. The associated C functions are presented below the C++ member functions in italics.

`anml_manager()` *anml_c_manager * init_anml_c_manager(void)*	Creates a new ANML manager.
`anml_manager( anml_c_manager * )`	Creates C++ manager copy of C manager.
`anml_manager( anml_manager * )` *anml_c_manager * copy_anml_c_manager(* *const anml_c_manager *)*	Creates copy of manager.
`~anml_manager()` *void delete_anml_c_manager(anml_c_manager *)*	Deletes manager.

Table A.1: Creating, Copying and Deleting ANML Managers (Section 3.2)

`int process_model_file( const char * )` *int am_process_model_file(anml_c_manager *, const char *)*	Processes a model file.

Table A.2: Processing ANML Models (Section 3.3)

`int write_cur_model( FILE * )` *int am_process_model_file(FILE *, anml_c_manager *)*	Writes current model to file.

Table A.3: Writing ANML Models (Section 3.4)

`int cur_comp()` *int am_cur_comp(anml_c_manager *)*	Indicates if current component is not NULL.
`int next_comp()` *int am_next_comp(anml_c_manager *)*	Traverses to next component.
`int prev_comp()` *int am_prev_comp(anml_c_manager *)*	Traverses to previous component.
`int first_sub_comp()` *int am_first_sub_comp(anml_c_manager *)*	Traverses to first sub-component of current component. For use with `ANML_MANAGER_FREEORDER` traversal only.
`int last_sub_comp()` *int am_last_sub_comp(anml_c_manager *)*	Traverses to last sub-component of current component. For use with `ANML_MANAGER_FREEORDER` traversal only.
`int parent_comp()` *int am_parent_comp(anml_c_manager *)*	Traverses to parent of current component. For use with `ANML_MANAGER_FREEORDER` traversal only.
`int reset_first_comp( int )` *int am_reset_first_comp(anml_c_manager *, int)*	Reset to first component. `ANML_MANAGER_PREORDER/POSTORDER/FREEORDER` traversals possible.
`int reset_last_comp( int )` *int am_reset_last_comp(anml_c_manager *, int)*	Reset to last component. `ANML_MANAGER_PREORDER/POSTORDER/FREEORDER` traversals possible.
`int lev_reset_first_comp()` *int am_lev_reset_first_comp(anml_c_manager *)*	Reset to first component on current level. For use with `ANML_MANAGER_FREEORDER` traversal only.
`int lev_reset_last_comp()` *int am_lev_reset_last_comp(anml_c_manager *)*	Reset to last component on current level. For use with `ANML_MANAGER_FREEORDER` traversal only.
`int comp_find_forward( const char * )` *int am_comp_find_forward(anml_c_manager *, const char *)*	Starting with the current component and moving forward, finds the first component that is an instance of the input class.
`int comp_find_backward( const char * )` *int am_comp_find_backward(anml_c_manager *, const char *)*	Starting with the current component and moving backward, finds the first component that is an instance of the input class.
`int lookup_comp( const char * )` *int am_lookup_comp(anml_c_manager *, const char *)*	Finds the component with an absolute identifier that matches the input identifier. Traversal method is set to `ANML_MANAGER_FREEORDER`

Table A.4: Traversing Components (Section 3.5)

`int cur_atr()` *int am_cur_atr(anml_c_manager *)*	Indicates if current attribute is not NULL.
`int next_atr()` *int am_next_atr(anml_c_manager *)*	Traverses to next attribute.
`int prev_atr()` *int am_prev_atr(anml_c_manager *)*	Traverses to previous attribute.
`int first_sub_atr()` *int am_first_sub_atr(anml_c_manager *)*	Traverses to first sub-attribute of current attribute. For use with composite attributes only.
`int last_sub_atr()` *int am_last_sub_atr(anml_c_manager *)*	Traverses to last sub-attribute of current attribute. For use with composites attributes only.
`int parent_atr()` *int am_parent_atr(anml_c_manager *)*	Traverses to parent of current attribute.
`int reset_first_atr()` *int am_reset_first_atr(anml_c_manager *)*	Reset to first attribute.
`int reset_last_atr()` *int am_reset_last_atr(anml_c_manager *)*	Reset to last attribute.
`int lev_reset_first_atr()` *int am_lev_reset_first_atr(anml_c_manager *)*	Reset to first attribute on current level. Intended for use with composite attributes.
`int lev_reset_last_atr()` *int am_lev_reset_last_atr(anml_c_manager *)*	Reset to last attribute on current level. Intended for use with composite attributes.
`int atr_find_forward( const char * )` *int am_atr_find_forward(anml_c_manager *, const char *)*	Starting with the current attribute and moving forward, finds the attribute with the specified name.
`int atr_find_backward( const char * )` *int am_atr_find_backward(anml_c_manager *, const char *)*	Starting with the current attribute and moving backward, finds the attribute with the specified name.

Table A.5: Traversing Attributes of Current Component (Section 3.6)

`int cur_atr_val()` *int am_cur_atr_val(anml_c_manager *)*	Indicates if current value is not NULL.
`int next_atr_val()` *int am_next_atr_val(anml_c_manager *)*	Traverses to next value. For use with list attributes only.
`int prev_atr_val()` *int am_prev_atr_val(anml_c_manager *)*	Traverses to previous value. For use with list attributes only.
`int reset_first_atr_val()` *int am_reset_first_atr_val(anml_c_manager *)*	Reset to first value. For use with list attributes only.
`int reset_last_atr_val()` *int am_reset_last_atr_val(anml_c_manager *)*	Reset to last value. For use with list attributes only.

Table A.6: Traversing Values of Current Attribute (Section 3.7)

`anml_c_manager * c_manager()`	Returns the C manager.

Table A.7: Accessing Manager Information (Section 3.8)

`int cur_model__name( char ** )` *int get_am_cur_model__name(anml_c_manager *, char **)*	Retrieves name of current model.
`int cur_model__schema_name( char ** )` *int get_am_cur_model__schema_name(anml_c_manager *, char **)*	Retrieves name of schema used by current model.

Table A.8: Accessing Current Model Information (Section 3.9)

`int cur_comp__id( char ** )` *int get_am_cur_comp__id(anml_c_manager *, char **)*	Retrieves level identifier of the current component.
`int cur_comp__abs_id( char ** )` *int get_am_cur_comp__abs_id(anml_c_manager *, char **)*	Retrieves absolute identifier of the current component.
`int cur_comp__name( char ** )` *int get_am_cur_comp__name(anml_c_manager *, char **)*	Retrieves name of the current component.
`int cur_comp__class( char ** )` *int get_am_cur_comp__class(anml_c_manager *, char **)*	Retrieves name of the class that the current component is an instance of.
`int cur_comp__app_class( char ** )` *int get_am_cur_comp__app_class(anml_c_manager *, char **)*	Retrieves name of the application class of the class that the current component is an instance of.
`int cur_comp_isa( const char *, int * )` *int am_cur_comp_isa(anml_c_manager *, const char *, int *)*	Indicates if the current component is an instance of the specified class.

Table A.9: Accessing Current Component Information (Section 3.10)

`int cur_atr__name( char ** )` *int get_am_cur_atr__name(anml_c_manager *, char **)*	Retrieves name of the current attribute.
`int cur_atr__composite_name( char ** )` *int get_am_cur_atr__composite_name(anml_c_manager *, char **)*	Retrieves composite name of the current attribute.
`int cur_atr__type( sch_atr_type * )` *int get_am_cur_atr__type(anml_c_manager *, sch_atr_type *)*	Retrieves the type of the current attribute.
`int cur_atr__mod_atr_type( mod_attribute_type * )` *int get_am_cur_atr__mod_atr_type(anml_c_manager *, mod_attribute_type *)*	Retrieves the `mod_attribute_type` of the current attribute.
`int conv_cur_atr_to_id_list()` *int am_conv_cur_atr_to_id_list(anml_c_manager *)*	Current attribute converted to list of identifiers. Attribute must be of `id_type`.

Table A.10: Accessing Current Attribute Information (Section 3.11)

`int cur_atr_val__string( char ** )` *int get_am_cur_atr_val__string(anml_c_manager *, char **)*	Retrieves string value of current value.
`int cur_atr_val__integer( int * )` *int get_am_cur_atr_val__integer(anml_c_manager *, int *)*	Retrieves integer value of current value. Value must be an integer.
`int cur_atr_val__real( double * )` *int get_am_cur_atr_val__real(anml_c_manager *, double *)*	Retrieves real value of current value. Value must be a real number.
`int cur_atr_val__boolean( int * )` *int get_am_cur_atr_val__boolean(anml_c_manager *, int *)*	Retrieves boolean value of current value. `false` = 0 and `true` = 1. Value must be a boolean.
`int cur_atr_val__abs_id( char ** )` *int get_am_cur_atr_val__abs_id(anml_c_manager *, char **)*	Retrieves absolute identifier of current value. Value must be an identifier.

Table A.11: Accessing Current Attribute Value Information (Section 3.12)

`void set_print_disabled( int )` *void set_am__print_disabled(anml_c_manager *, int)*	Disables ANML print statements.
`void set_warnings_disabled( int )` *void set_am__warnings_disabled(anml_c_manager *, int)*	Disables ANML warning messages.

Table A.12: Setting Manager Functionality (Section 3.13)

Appendix B

Code for Example Program

This section gives the source code for `net_proc.C`, the C++ example program described in Section 4.

```
/*
** This file is copyright (C) 2002 Telesim Group
** (University of Calgary)
**
** This program is distributed WITHOUT ANY WARRANTY; without even the
** implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
** PURPOSE. See the file LICENSE for more details.
**
*/

/*
 * This is a program to demonstrate the use of ANML by another application.
 * Using the anml_manager this program traverses and displays info on the
 * different network components of an ANML model that use the NetSchema.
 */

#include <fstream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <new.h>
#include "anml_manager.h"

static void traverse_hosts( anml_manager * );
static void traverse_routers( anml_manager * );
static void traverse_p2p_links( anml_manager * );
static void traverse_lan_links( anml_manager * );

int
main( int argc, char * argv[] )
{
  anml_manager * manager = NULL;
```

```
  int i = 0;
  char * model_filename = NULL;

  /* extract model filename to process from command line arguments */
  for( i = 0; i < argc; ++i ) {
    if( strcmp( argv[i], "-model" ) == 0 )
      break;
  }
  ++i;
  if( i < argc ) {
    model_filename = argv[i];
  }
  else {
    fprintf(stderr,"-model <filename> must be specified.\n" );
    exit( 1 );
  }

  /* create the anml_manager and process the model file */
  manager = new anml_manager();
  if( (manager->process_model_file( model_filename )) !=
      ANML_MANAGER_SUCCESS )
    exit( 1 );

  /* traverse and display information on the different model components */
  traverse_hosts( manager );
  traverse_routers( manager );
  traverse_p2p_links( manager );
  traverse_lan_links( manager );

  /* clean up */
  delete manager;

  exit( 0 );
}

void
traverse_hosts( anml_manager * manager )
{/* Traverses and displays information on all components that are instances
  * of the class Host */

  char * abs_id = NULL;
  int buffer_size = 0;
  char * lan_link_abs_id = NULL;


  fprintf( stdout, "Here are the Hosts:\n" );
  fprintf( stdout, "-------------------\n\n" );
```

```
  manager->reset_first_comp( ANML_MANAGER_PREORDER );

  while( manager->comp_find_forward( "Host" ) == ANML_MANAGER_SUCCESS ) {
    manager->cur_comp__abs_id( &abs_id );
    fprintf( stdout, "%s\n", abs_id );

    manager->reset_first_atr();
    if( manager->atr_find_forward( "buffer_size" ) != ANML_MANAGER_SUCCESS ) {
      fprintf( stderr, "Failed to find 'buffer_size' attribute.\n" );
      exit( 1 );
    }
    manager->cur_atr_val__integer( &buffer_size );
    fprintf( stdout, "   buffer_size: %d Bytes\n", buffer_size );

    manager->reset_first_atr();
    if( manager->atr_find_forward( "lan_link" ) == ANML_MANAGER_SUCCESS ) {
      manager->cur_atr_val__abs_id( &lan_link_abs_id );
      fprintf( stdout, "   lan_link: %s\n", lan_link_abs_id );
    }

    fprintf( stdout, "\n" );
    manager->next_comp();
  }
  fprintf( stdout, "\n" );
}

void
traverse_routers( anml_manager * manager )
{/* Traverses and displays information on all components that are instances
  * of the class Router */

  char * abs_id = NULL;
  double proc_delay = 0.0;
  int buffer_size = 0;
  char * lan_link_abs_id = NULL;

  fprintf( stdout, "Here are the Routers:\n" );
  fprintf( stdout, "---------------------\n\n" );

  manager->reset_first_comp( ANML_MANAGER_PREORDER );

  while( manager->comp_find_forward( "Router" ) == ANML_MANAGER_SUCCESS ) {
    manager->cur_comp__abs_id( &abs_id );
    fprintf( stdout, "%s\n", abs_id );

    manager->reset_first_atr();
```

```
    if( manager->atr_find_forward( "proc_delay" ) != ANML_MANAGER_SUCCESS ) {
      fprintf( stderr, "Failed to find 'proc_delay' attribute.\n" );
      exit( 1 );
    }
    manager->cur_atr_val__real( &proc_delay );
    fprintf( stdout, "   proc_delay: %.2e s\n", proc_delay );

    manager->reset_first_atr();
    if( manager->atr_find_forward( "buffer_size" ) != ANML_MANAGER_SUCCESS ) {
      fprintf( stderr, "Failed to find 'buffer_size' attribute.\n" );
      exit( 1 );
    }
    manager->cur_atr_val__integer( &buffer_size );
    fprintf( stdout, "   buffer_size: %d Bytes\n", buffer_size );

    manager->reset_first_atr();
    if( manager->atr_find_forward( "lan_links" ) == ANML_MANAGER_SUCCESS ) {
      manager->conv_cur_atr_to_id_list();
      manager->reset_first_atr_val();
      fprintf( stdout, "   lan_links: ");
      while( manager->cur_atr_val() == ANML_MANAGER_SUCCESS ) {
        manager->cur_atr_val__abs_id( &lan_link_abs_id );
        fprintf( stdout, "%s  ", lan_link_abs_id );
        manager->next_atr_val();
      }
      fprintf( stdout, "\n" );
    }

    fprintf( stdout, "\n" );

    manager->next_comp();
  }
  fprintf( stdout, "\n" );
}

void
traverse_p2p_links( anml_manager * manager )
{/* Traverses and displays information on all components that are instances
  * of the class P2P_Link */

  char * abs_id = NULL;
  char * nodeA_abs_id = NULL;
  char * nodeB_abs_id = NULL;
  double delay = 0.0;
  double rate = 0.0;
  int mtu = 0;
```

```
fprintf( stdout, "Here are the P2P Links:\n" );
fprintf( stdout, "----------------------\n\n" );

manager->reset_first_comp( ANML_MANAGER_PREORDER );

while( manager->comp_find_forward( "P2P_Link" ) == ANML_MANAGER_SUCCESS ) {
  manager->cur_comp__abs_id( &abs_id );
  fprintf( stdout, "%s\n", abs_id );

  manager->reset_first_atr();
  if( manager->atr_find_forward( "nodeA" ) != ANML_MANAGER_SUCCESS ) {
    fprintf( stderr, "Failed to find 'nodeA' attribute.\n" );
    exit( 1 );
  }
  manager->cur_atr_val__abs_id( &nodeA_abs_id );
  fprintf( stdout, "   nodeA: %s\n", nodeA_abs_id );

  manager->reset_first_atr();
  if( manager->atr_find_forward( "nodeB" ) != ANML_MANAGER_SUCCESS ) {
    fprintf( stderr, "Failed to find 'nodeB' attribute.\n" );
    exit( 1 );
  }
  manager->cur_atr_val__abs_id( &nodeB_abs_id );
  fprintf( stdout, "   nodeB: %s\n", nodeB_abs_id );

  manager->reset_first_atr();
  if( manager->atr_find_forward( "rate" ) != ANML_MANAGER_SUCCESS ) {
    fprintf( stderr, "Failed to find 'rate' attribute.\n" );
    exit( 1 );
  }
  manager->cur_atr_val__real( &rate );
  fprintf( stdout, "   rate: %.2f Mbps\n", rate );

  manager->reset_first_atr();
  if( manager->atr_find_forward( "delay" ) != ANML_MANAGER_SUCCESS ) {
    fprintf( stderr, "Failed to find 'delay' attribute.\n" );
    exit( 1 );
  }
  manager->cur_atr_val__real( &delay );
  fprintf( stdout, "   delay: %.2e s\n", delay );

  manager->reset_first_atr();
  if( manager->atr_find_forward( "mtu" ) != ANML_MANAGER_SUCCESS ) {
    fprintf( stderr, "Failed to find 'mtu' attribute.\n" );
    exit( 1 );
  }
  manager->cur_atr_val__integer( &mtu );
```

```
    fprintf( stdout, "   mtu: %d Bytes\n", mtu );

    fprintf( stdout, "\n" );
    manager->next_comp();
  }
  fprintf( stdout, "\n" );
}

void
traverse_lan_links( anml_manager * manager )
{/* Traverses and displays information on all components that are instances
  * of the class LAN_Link */

  char * abs_id = NULL;
  double rate = 0.0;
  double delay = 0.0;
  int mtu = 0;

  fprintf( stdout, "Here are the LAN Links:\n" );
  fprintf( stdout, "-----------------------\n\n" );

  manager->reset_first_comp( ANML_MANAGER_PREORDER );

  while( manager->comp_find_forward( "LAN_Link" ) == ANML_MANAGER_SUCCESS ) {
    manager->cur_comp__abs_id( &abs_id );
    fprintf( stdout, "%s\n", abs_id );

    manager->reset_first_atr();
    if( manager->atr_find_forward( "rate" ) != ANML_MANAGER_SUCCESS ) {
      fprintf( stderr, "Failed to find 'rate' attribute.\n" );
      exit( 1 );
    }
    manager->cur_atr_val__real( &rate );
    fprintf( stdout, "   rate: %.2f Mbps\n", rate );

    manager->reset_first_atr();
    if( manager->atr_find_forward( "delay" ) != ANML_MANAGER_SUCCESS ) {
      fprintf( stderr, "Failed to find 'delay' attribute.\n" );
      exit( 1 );
    }
    manager->cur_atr_val__real( &delay );
    fprintf( stdout, "   delay: %.2e s\n", delay );

    manager->reset_first_atr();
    if( manager->atr_find_forward( "mtu" ) != ANML_MANAGER_SUCCESS ) {
      fprintf( stderr, "Failed to find 'mtu' attribute.\n" );
      exit( 1 );
```

```
    }
    manager->cur_atr_val__integer( &mtu );
    fprintf( stdout, "   mtu: %d Bytes\n", mtu );

    fprintf( stdout, "\n" );
    manager->next_comp();
  }
  fprintf( stdout, "\n" );
}
```