```
 1 #!/usr/local/bin/perl
 2
 3 use STAR::Parser;
 4 use STAR::DataBlock;
 5 use STAR::Dictionary;
 6
 7 my $file = $ARGV[0];
 8
 9 @dicts = STAR::Parser->parse(-file=>$file, -dict=>1);
10
11 exit unless ( $#dicts >= 0 ); # exit if nothing there
12
13 print_document_header;
14
15 foreach $dictionary (@dicts) { # usually one
16   # Get the dictionary name and version
17   $dictname = ($dictionary->get_item_data(
18                            -item=>"_dictionary.title"))[0];
19   $dictversion = ($dictionary->get_item_data(
20                          -item=>"_dictionary.version"))[0];
21
22   print "Dictionary $dictname, version $dictversion\n";
23
24   # Get list of save-frame names, sorted alphabetically
25   @saveblocks = sort $dictionary->get_save_blocks;
26
27   foreach $saveblock (@saveblocks) { # For each save frame
28     @categories = $dictionary->get_item_data(
29                                    -save=>$saveblock,
30                                    -item=>"_category.id");
31     if ( $#categories == 0 ) { # category save frame
32       format_category($saveblock); # format the category
33
34       $category = $categories[0] ;
35       foreach $block (@saveblocks) { # re-traverse blocks
36         @itemcategoryids = $dictionary->get_item_data(
37                                    -save=>$block,
38                                -item=>"_item.category_id");
39         @itemname = $dictionary->get_item_data(
40                                    -save=>$block,
41                                    -item=>"_item.name");
42
43         if ( $#itemcategoryids == 0) { # category pointer
44           format_item($block)
45                if $itemcategoryids[0] =  /^$category$/i;
46         } elsif ( $#itemname == 0 &&
47                     $itemname[0] =  /^_$category\./i ) {
48           format_item($block);
49         }
50
51       } # end foreach of items matching current category
52     }
53   } # end foreach of save frames in the dictionary
54 } # end foreach loop per included dictionary
55
56 print_document_footer;
57 exit;
```

Fig. 5.3.6.1. Skeleton version of an application to format a CIF dictionary for publication. Only the main program fragment is shown. Line numbering is provided for referencing in the text. *format_category*, *format_item* and the *print_document_** commands are calls to external subroutines not included in this extract.

in the input dictionary is assembled by retrieving the value associated with the data item **_category.id** as the component save frames are scanned in sequence. Note that for applications within dictionaries, the method takes a `-save=>$saveblock` parameter to allow the extraction of items from specific save frames. For manipulations of data files, this parameter is omitted. In lines 17–20, the method is called without this parameter because the name and version of the dictionary are expected to be found in the outer part of the file, not within any save frame.

get_keys allows a user to display the structure of a CIF or dictionary file and can be used to analyse the content of an unknown input file. When written to a terminal, the string that is returned by this method appears as a tabulation of the items present at

Table 5.3.6.1. *Object methods provided by the STAR::DataBlock Perl module*

Method	Description
store	Saves a DataBlock object to disk
get_item_data	Returns all the data for a specified item
get_keys	Returns a string with a hierarchically formatted list of hash keys (data blocks, save blocks, categories and items) found in the data structure of the DataBlock object
get_items	Returns an array with all the items present in the DataBlock
get_categories	Returns an array with all the categories present in the DataBlock
insert_category	Inserts a category into a data structure
insert_item	Insert an item into a data structure
set_item_data	Sets the data content of an item according to a supplied array

the different levels in the data structure hierarchy, each level in the hierarchy being indicated by the amount of indentation (Fig. 5.3.6.2).

The *get_items* and *get_categories* methods are largely self-explanatory. The items or categories in the currently active DataBlock object are returned in array context.

insert_item and *insert_category* are the complements of these methods, designed to allow the insertion of new items or categories. Where appropriate (*i.e.* in dictionary applications), the save frame into which the insertion is to be made can be specified.

The remaining method, *set_item_data*, is called to set the data of item `$item` to an array of data referenced by `$dataref`:

```
$data_obj->set_item_data( -item=>$item,
                          -dataref=>$dataref );
```

As usual, an optional parameter `-save=>$save` may be included for dictionary applications where a save frame needs to be identified; the value of the variable *$save* is the save-frame name.

Note that the current version of the module does not support the creation and manipulation of data loops, although the *get_item_data* method will correctly retrieve arrays of data values from a looped list.

There are five methods available to set or retrieve attributes of a DataBlock object, namely: *file_name* for the name of the file in which the DataBlock object was found; *title* for the title of the DataBlock object (*i.e.* the name of the CIF data block with the leading **data_** string omitted); *type* for the type of data contained – 'data' for a DataBlock object but 'dictionary' for an object in the STAR::Dictionary subclass; and *starting_line* and *ending_line* for the start and end line numbers in the file where the data block is located. The method *get_attributes* returns a string containing a descriptive list of attributes of the DataBlock object.

5.3.6.1.4. *STAR::Checker*

This module implements a set of checks on a data block against a dictionary object and returns a value of '1' if the check was successful, '0' otherwise. The check tests a specific set of criteria:

(i) Are all items in the DataBlock object defined in the dictionary?

(ii) Are mandatory items present in the data block?

(iii) Are dependent items present in the data block?

(iv) Are parent items present?

(v) Do the item values conform to item type definitions in the dictionary?

Obviously, these criteria will not be appropriate for all purposes, and are in any case fully developed only for DDL2 dictionaries. An optional parameter `-options=>'1'` may be set to write a list of specific problems to the standard error output channel.