Application program interface

The API provides the following.

```
celltype bottom;     /* unique "unknown" cell */
listtype nil;        /* unique empty list */
dectype equalsdec;   /* declaration of relational operator (_ = _) */
listtype documents;  /* abstract syntax tree of whole spec */
sttype symtable;     /* symbol table */

typedef enum {BY_PIPE, BY_SOCKET} apimode;
typedef enum {WHOLE_SPEC, NEEDED_PARAS} biomode;
typedef enum {CORRECT, WRONG, QUITTED} checkerstatus;

checkerstatus cadiz_init(apimode amode, biomode bmode, char *prog, char *
listtype cadiz_command(cmdtype cmd);
void cadiz_kill(void);
void cadiz_suicide(void);
```

To use the above, source must include the following.

```
#include "api.h"
```

The following libraries have to be linked. They are listed in an acceptable order; other orders might not work.

```
api.a list.a types.a handles.o heap.a misc.a eject.a
```

The `cadiz_init` function invokes cadiz as a separate process using the given prog and arg list. For example... `cadiz_init(BY_PIPE, WHOLE_SPEC, "cadiz", "-v", "-l", "spec.z", NULL)`

`cadiz_init` sets up communication streams to receive the data structures representing a typechecked Z specification and to allow subsequent invocation of commands and receipt of results. The first argument determines the mechanism used for these streams. If there is already a cadiz process created by this means, `cadiz_init` calls `cadiz_kill` first—there can be only one at once. It waits for cadiz to typecheck the specification. If no errors are detected in the specification, the root pointers `bottom`, `nil`, `equalsdec`, `documents` and `symtable` are initialized, and `CORRECT` is returned. How much of the specification is transferred at once is determined by the second argument to `cadiz_init`. If some errors are detected in the specification, `WRONG` is returned. If cadiz cannot be executed, or if it is quitted by the user before typechecking is completed, `QUITTED` is returned.

The `cadiz_command` function causes execution of command `cmd` by cadiz. The `cmd` has attributes for the command's operands, which should have been set to selections referring to parts of the abstract syntax tree or to literals as appropriate for the command. See `src/lib/types/cmd.d` for specification of these attributes. A list is returned of those paragraphs that the command would have added to the left-hand window if cadiz had been run interactively. Commands which engage in other interaction, such as dialogue boxes, might not work through this interface.

The `cadiz_kill` function causes the process created by `cadiz_init` to terminate, and waits for it to do so (thus ensuring that any output streams are flushed).

The `cadiz_suicide` function blocks the invoking process until the cadiz process

itself chooses to exit.

For an example of the use of this API, see `src/procs/apitest/`.

When a Z paragraph (`def`) or declaration (`dec`) is an argument to a command, cadiz uses any existing copy of that data structure in preference to the command's argument. Other data structures are copied afresh. In practice, problems arising from this can be avoided by using an applicative style of coding, building new abstract syntax trees in preference to modifying existing ones. The reverse situation, where cadiz modifies an attribute of a goal of which a copy has previously been made across the API, has no correspondingly simple solution. The reasons for this are explained in "Efficient Binary Transfer of Pointer Structures" by Ian Toyn and Alan J. Dix, Software – Practice and Experience, 24(11) pp1001-1023 November 1994.

IT 24-Apr-2002