Importing Rules and Objects into Check Point Firewall using DBEDIT

While it's the recommend way to do, managing your objects and rules solely through Check Point SmartDashboard may be cumbersome.
Bad enough, there exists no real CLI interface, that would allow for real scripting. Well, there exists DBEDIT, which allows for automated creation of objects and even rules ... sort of.
However there's barely official documentation about it, if not Martin Hoz had taken the time to write the very useful Object Filler utility, which you find over at the Check Point User Group.

And unless you don't want to go into the Check Point OPSEC API, DBEDIT (by the help of Object Filler) is the way to go.

If you are however seeking into developping a fully automated, non-interactive import script then you are way off. There's actually lots of limitations in using DBEDIT as I found during writing an auto-import tool for my employer. Here's just a few to name:

- Existing objects of any kind cannot be overwritten, doing so will always trigger a Y/N prompt
- Rules can be created into any policy set, but ...
- Existing rules cannot be overwritten/replaced
- New rules cannot be inserted freely between existing rules
- Rules can however be appended to any policy - you need however the number of the last rule in the rule database to use it as an offset for the rules
- Existing rules can be deleted from any policy
- New policies can be created as well

So if you want to edit existing policies, you need to take the long road by dumping polices with odumper, edit them (programmatically) as needed, and re-import them.

If you eventually mastered the art of creating DBEDIT input files programmatically (preferrably by the help of Object Filler), you still need to import them.

Here's a short prototype perl script that basically creates a database revision, counts the to-be-imported DBEDIT statements, runs DBEDIT and finally counts output messages from DBEDIT. This will allow for simple comparison if the number of import statements matched with the number of acknowledged statements.

It allows allows for some basic error checking due to the fact how DBEDIT works. And yet, since unexpected behaviour most of the time triggers a Y/N prompt in DBEDIT, this can be used for error handling as well. If you simply add a timer before forking the process, you then simply wait for the timer to expire. Thus allowing you to forcibly exit DBEDIT upon error. But, since at that point already some of the DBEDIT input commands might have been comitted, it's a good option to revert to a previous database revision.

```
#!/usr/bin/perl

use strict;
use Fcntl;

use constant DBEDIT_BIN  = "/path/to/dbedit";
use constant DBVER_BIN  = "/path/to/dbver";
use constant CP_USERNAME = "admin username";
use constant CP_PASSWORD = "admin password";

{
 my $_DBEFile = "/path/to/dbedit_import_file.dbe";
 my $_DBELogFile = "/path/to/dbedit.log";

 my $_state_tracker = ();                # array hash to hold state tracking information, initialize with '0'
 $_state_tracker->{'total_statements'}   = 0;
 $_state_tracker->{'imported_statements'}= 0;
 $_state_tracker->{'duplicate_objects'}  = 0;
 $_state_tracker->{'locked_objects'}     = 0;
 $_state_tracker->{'invalid_objects'}    = 0;

 printf "NOTICE: Scanning for config statements to import ...n";
 sysopen(DBE_FILE, $_DBEFile, O_RDONLY) or die "Can't read from '$_DBEFile': $!n";
 while() {
  if( m/(?:update|delete) .*$/ ) { $_state_tracker->{'total_statements'}++; }
 }
 close(DBE_FILE);
 printf "NOTICE: We seem to have '%d' config statements to be imported.n", $_state_tracker->{'total_statements'};

 # it's a good idea to create a new database revision
 # before we start
 #
 system( sprintf( "%s -s localhost -u %s -w %s -m create %s %s |",
    DBVER_BIN, CP_USERNAME, CP_PASSWORD,
    sprintf("db_snapshot_%s", POSIX::strftime("%m-%d-%Y_%H%M%S", localtime)),
    sprintf("db_snapshot_%s", POSIX::strftime("%m-%d-%Y_%H%M%S", localtime))
   )
  );

 # now let's fork dbedit to import our DBE file
 #
 eval {
  local $SIG{ALRM} = sub { die "timed outn" }; # note: 'n' required at end of string !
  alarm 90;       # set timer to 90 secs

  printf "NOTICE: Running DBEDIT now ...n";
  system( sprintf("%s -s localhost -u %s -p %s -f %s > %s 2>&1",
    DBEDIT_BIN, CP_USERNAME, CP_PASSWORD, $_DBEFile, $_DBELogFile ) );

  alarm 0;        # reset timer
 };

 # handle import results
 # (both error OR success)
 #
 if ($@) {
  # something bad happened
  # while importing data ...

  # since we forked DBEDIT, it may still be running
  # at this point -- we need to forcibly kill it
  # so as we leave no stale session behind
  #
  system("/usr/bin/killall dbedit");

  # do whatever we need to do for error handling
  # for example revert to a previous database revision
  #

 } else {

  # analyze output logfile for certain patterns
  #
  sysopen(DBEDIT_LOG, $_DBELogFile, O_RDONLY) or die "Can't open file for reading '$_DBELogFile': $!n";
  while (  ) {
   if( m~.*updated successfully~ || m~.*deleted~ ) {
    $_state_tracker->{'imported_statements'}++;
   }
   if( m~Object Already Exists~ ) {
    $_state_tracker->{'duplicate_objects'}++;
   }
   if( m~Object Lock Failed~ ) {
    $_state_tracker->{'locked_objects'}++;
   }
   if( m~.*Validation error~ ) {
    $_state_tracker->{'invalid_objects'}++;
   }
  }
  close(DBEDIT_LOG);

  # ok, now that we have scanned the logfile,
  # we must evaluate the results
  #
  printf "NOTICE: Here's our import results:n" .
   sprintf( "        -> Total Update Statements  : %sn", $_state_tracker->{'total_statements'}) .
   sprintf( "        -> Total Statements Imported: %sn", $_state_tracker->{'imported_statements'}) .
   sprintf( "        -> Total Duplicated Objects : %sn", $_state_tracker->{'duplicate_objects'}) .
   sprintf( "        -> Total Locked Objects     : %sn", $_state_tracker->{'locked_objects'}) .
   sprintf( "        -> Total Invalid Objects    : %sn", $_state_tracker->{'invalid_objects'});

  # do whatever we need to do according to these results
  #
       }

 }
```

I wrote this piece of code some four years back already, yet it might still be useful enough today and give a starting point.
I have more code around for importing complex rulesets and objects to the database via DBEDIT, though I'm currently not allowed to publish this at this time.