EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
European Laboratory for Particle Physics

Internal Note/DAQ
ALICE reference number
ALICE-INT-2001-15 V 1.3
Institute reference number

Date of last change
2001-06-16

The DATE SMI run control

Author:

S. Chapeland

Abstract:

The DATE V3 system includes shell scripts for the control of external processes to be executed at the start-of-run and at the end-of-run. This feature has been complemented by the addition of a State Machine Interface (SMI) package to synchronise and co-ordinate their execution both at the single detector level and at the experiment levels. This note describes the integration of SMI with the DATE system.

Contact Person P. Vande Vyvre	e-mail:Pierre.Vande.Vyvre@cern.ch
Tel: + 41-22 767.83.36 / Fax : + 41-22.767.95.85	Creation Date: 2001-January-18

Distribution lists
For Approval:

For Information:

EDMS Number: 312494 version 1.0

1. Introduction

In DATE V3 [1], the LDC or GDC software can be customized by the addition of detector specific programs. These programs are started through common (to all LDCs or GDCs) or specific (to one LDC or GDC) Unix shells scripts. These scripts are executed at every start and at every end of run and are then executed independently from the DATE system. However, there is a need to synchronise the execution of these programs with the DATE start and end of run sequences without modification of the DATE run-control itself.
Different methods and tools have been investigated to identify a solution for control of processes by the run-control [1, 2]. None of these options has been found sufficiently satisfactory to be included in the DATE system.
A first prototype has also been developed to interface DATE with the SCADA (System for Control and Data Acquisition) system currently used at CERN for the development of Detector Control Systems [3]. This prototype has shown that on the one hand the user interface of the SCADA system was not well adapted to the control of a DAQ system and on the other hand the SCADA system itself requires the addition of an external package for the control of external programs. The external package that has been used for a test with the SCADA is the State Machine Interface [4] developed for Delphi and used by Delphi and a few other experiments since several years.
A second prototype has then been started to integrate directly DATE, and in particular the run-control, with the SMI package. This note describes the architecture of this integration, the modifications done in the DATE run control and the installation of the DIM (Distributed Information Management System) [6] and the SMI packages.

2. The DATE and SMI packages

This note assumes a good knowledge of the DATE system and of the basic SMI and DIM concepts. See the references to find a list of relevant documents. SMI and DATE basic knowledge can be found in Ref [4, 5].

3. Usage of SMI for DATE

The idea of finite state machines is to describe the behavior of a complex system in a modular and readable way. Moreover, given that DATE has to be connected to different detectors, the front-end software may require the addition of detector specific software. For instance, when a run starts, some detector specific processes dedicated to a given hardware may need to be started. This can even be some more complex procedure including different phases such as calculating pedestals, loading them and checking the status. A script-based description of the actions to do at start of run (and in each of the phases of the run) allows flexibility and modifications at each level.

SMI was tried for this prototype because it was developed at CERN, intensively tested in the Delphi experiment and still maintained. It has all the features required to implement finite state machines and distribution of the system. The SMI system uses DIM as the communication layer. The DIM package includes publish and subscribe facilities organized around a domain name server.

4. Architecture

The architecture is based on DATE v3.7. No modifications were done in any other package than on the run control. The software has been kept compatible with DATE v3.7. The new run control can even work in the v3.7 native mode.

The SMI package is a software tool to control complex distributed systems. The control is based on a model made of the system in terms of objects with a finite state machine behavior. SMI includes a dedicated language used to express the model. The SMI language is used to write SMI programs or "scripts" that are executed by the SMI interpreter or "state manager".

The SMI objects used to model the system belong to two distinct categories. Some of these objects correspond to existing hardware or software elements of the system such as the DATE processes. In this case the SMI object is "associated" to a program. Another category of object is logical objects that are abstract entities that exist only in the SMI script.

Each DATE process is associated to a corresponding SMI object. Each SMI object behavior is described in a SMI script, giving the different states, possible actions and triggering events. These scripts are easier to read and maintain than lower level language code. Each script is interpreted by a process called «state manager».

The DATE run control logical architecture is shown in Figure 1.

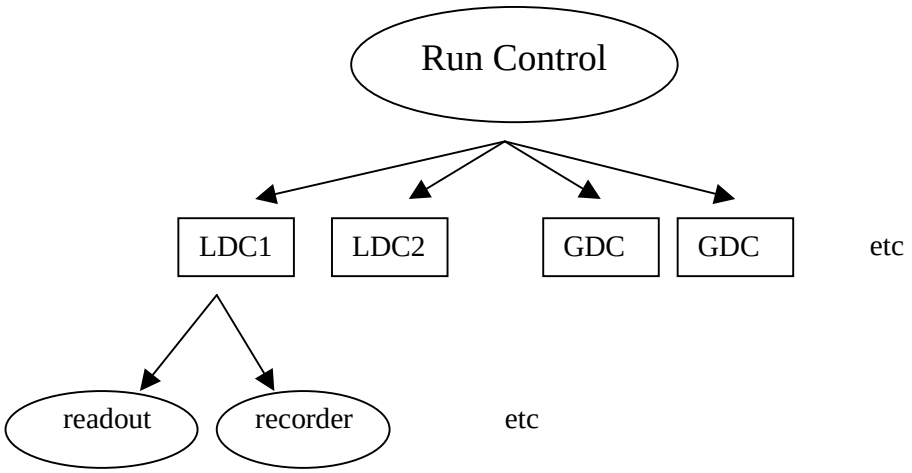

Figure 1 - DATE run control logical architecture

The root of the tree is the run control process. The main purposes of the runControl are :

- provide a user interface to input run parameters and control the system (start, stop commands, status display…);
- communicate with the subsystems (LDCs, GDCs, …), upload run parameters, control and synchronize them (send commands, check status).

The current version of the user interface is adequate for the current applications and has not been modified. Only the control part was changed and needs to be discussed here.

The run control itself has a corresponding SMI object, called `DATE_RUN`. This object's behavior is completely described in a SMI script. Links between user interface and the actions in each state have been established, so that a click on the START button actually triggers a START action in the SMI interpreter that simulates the object. Some actions done at start or end of run

are not handled by the SMI control yet, like the run parameters downloading. These actions are still written in a Tcl script.

On the lower level, there are objects corresponding to the LDCs and GDCs. Their associated processes (on each machine) are named «LDC_SMI_SERVER» and «GDC_SMI_SERVER». These objects have a well defined interface with the object DATE_RUN. Their states and actions are described in the `DATE_RUN` SMI script. The state machine behavior of a LDC or a GDC is local to a machine. It is possible to change the script, provided the interface is respected. The SMI class SMI_SERVER indicated in Listing 1 defines this interface.

Figure 2 shows all SMI objects used for the run control and how they interact with each other. Bold arrows represent the control flow. Dashed arrows show links that don't deal with control, but which are more specific to the implementation.

```
class: SMI_SERVER /associated
      state: STOPPED
            action: START
      state: STARTING
      state: READY
            action: SET_RUNNING
      state: RUNNING
            action: STOP
      state: ENDING

      state: ERROR
            action: STOP
```

Listing 1 – State machine description of a LDC or GDC object

Coexistence of objects with the same name is allowed by the usage of SMI domains. A SMI domain is a name space to which an object belongs and which limits the visibility of this name. Therefore, the system can hold an unlimited amount of «LDC_SMI_SERVER» objects, if all are in a different SMI domain.

The SMI domain feature is used to keep clean boundaries between machines, and to have an homogeneous naming convention on each machine. Currently, the SMI domain is based on the ip name and type (LDC or GDC) of a machine, to avoid conflicts and to ease the deployment.

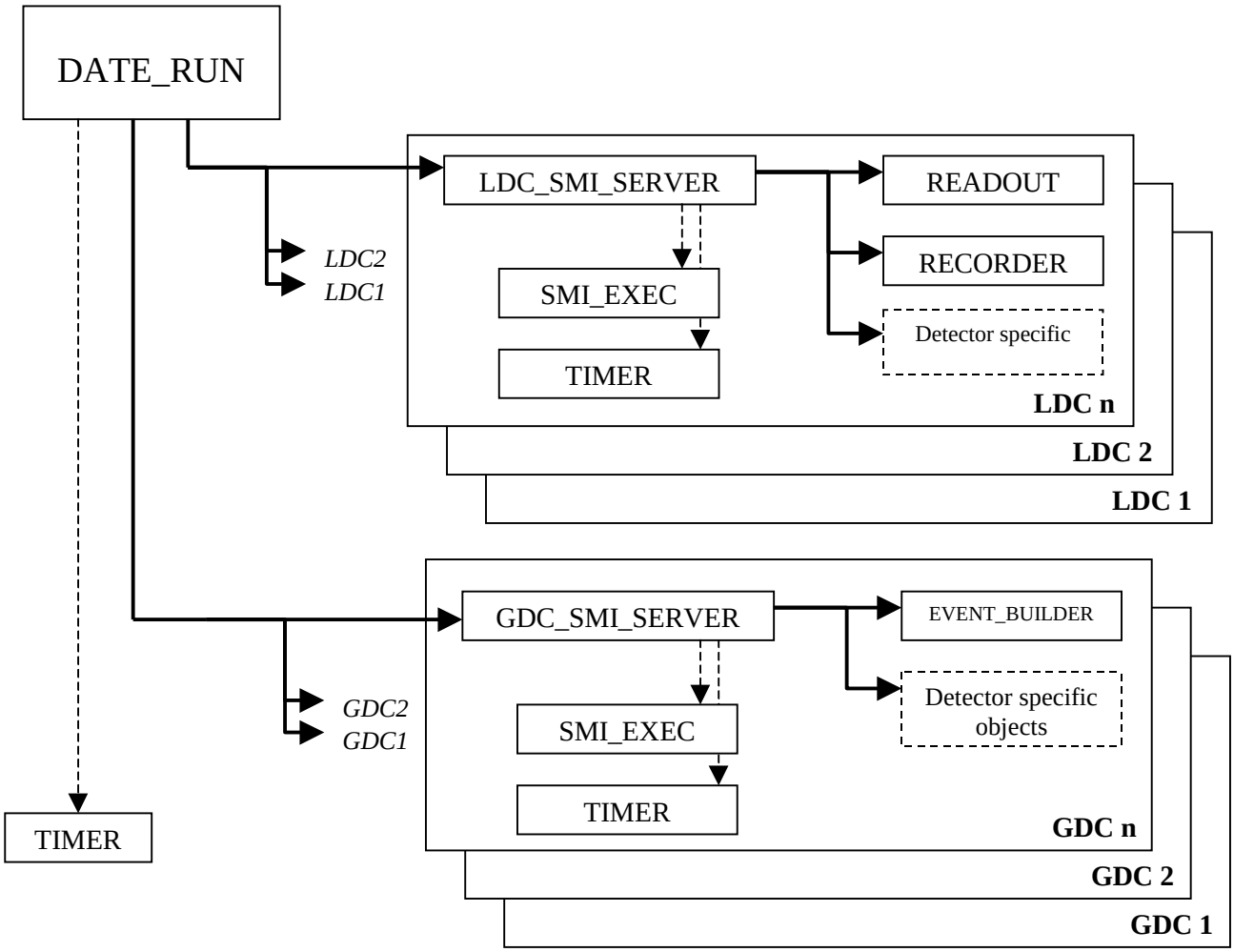

Figure 2 - SMI objects of the DATE run control

The LDCs and GDCs are selected by the user in the DATE run control. A number of macros have been developped to write the run control SMI script without knowing a priori which are the machines involved in the run. These macros are of the type `$ALL_LDC`, `$ALL_GDC`, `$ANY_LDC`, `$ANY_GDC`. They are replaced by the effective list of machines by the Tcl program during the connect command once they have been defined. The run control executes this substitution immediately before launching the `DATE_RUN` SMI interpreter. An example of a simple SMI script for object `DATE_RUN` is given in Listing 2.

```
state: STOPPED
      action: START
            do START $ALL_LDC::LDC_SMI_SERVER
            do START $ALL_GDC::GDC_SMI_SERVER
            do START(TIME=20) timer
            terminate_action/state=STARTING
      action: DISCONNECT
            terminate_action/state=NOT_CONNECTED
```

```
state: STARTING
      when ($ALL_LDC::LDC_SMI_SERVER in_state READY)
            AND ($ALL_GDC::GDC_SMI_SERVER in_state READY)
            do SET_READY

      when (timer in_state TIME_OUT)
            do SET_ERROR

      action: SET_READY
            do STOP timer
            terminate_action/state=READY
      action: SET_ERROR
            do STOP timer
            terminate_action/state=ERROR
```

Listing 2 - Example of SMI script for run control

On each LDC (or GDC), there are other SMI objects running, used for several low level purposes. The «smi_exec» object provides a set of actions to communicate with lower level DATE processes, like readout or recorder. It has an SMI associated process, written in C, that extends the restrictive capabilities of SMI scripting to all the features of the C-language. For example, this smi_exec object can be used to set or read flags in the DATE shared memory segment, to log messages, to start processes. Another SMI object that is found everywhere is the `TIMER` object, which gives a way to have timeouts in SMI scripts.
An example of SMI script for a LDC is given in Listing 3.

```
state: STARTING
when LDC_SMI_SERVER in_state STARTING do EXECUTE_START

action: EXECUTE_START
      do SET_RUNSTATUS(runstatus="STARTING") smi_exec
      do REINIT_SHM_VARS smi_exec

      if (smi_exec not_in_state READY) then
            terminate_action/state=ERROR
      endif

      do START recorder
      if (recorder not_in_state READY) then
            terminate_action/state=ERROR
      endif

      do START readout
      if (readout not_in_state READY) then
            terminate_action/state=ERROR
      endif

      do SET_RUNSTATUS(runstatus="READY") smi_exec

      terminate_action/state=READY
```

Listing 3 – Example of SMI script for LDC

5. SMI Tests

Extensive testing has been done to understand the features provided by SMI and to check the behaviour of SMI objects on a large scale. Among them, the following features were tested:

- commands between objects of same or different domains
- associated processes / objects
- SMI classes
- queue priority when several actions are sent to the same object
- synchronisation between several objects
- asynchronous actions / priority of actions:
  - 2 objects with the same «when»
  - several «when» within one object
- usage of the SMI Run Time Library
- sending a command to a large number of associated processes (up to 100 on one machine) and synchronisation between them

6. Tests with the SMI DATE run control

Extensive testing has been performed, including either standalone tests with a limited number of machines, or tests with a few tens of machines using the third ALICE Data Challenge setup (ADC3).

Tests have been done with rapid start/stop commands. Typically, the maximum number of events was set to 100 to 200 in the DATE parameters, so that machines ask for the stop of the run a few seconds (around 5) after being started. In these conditions, a good stability has been reached. More than 4000 start/stop in 14 hours (1 run start and stop every 20 seconds) has been successively triggered without error in a 3 LDCs x 1 GDC configuration.

Tests of synchronisation have also been successful in the ADC3 setup, in a configuration reaching 13 LDCs x 13 GDCs running full speed.

The DATE performance has not been influenced by the introduction of SMI. Start and stop phases are even 10% to 35% faster, depending on the configuration. The scalability of the DIM domain name server should be checked, especially when there is a large number of nodes (more than 50). In a 13x13 configuration, the DIM domain name server (DNS) machine is far from saturation: less than 5% CPU usage is reached at maximum during one second at start of run, on a 2x400MHz PC.

The execution time of the sequences for start and stop of run is satisfactory in the current prototype of the SMI DATE run control.

7. Conclusion

The prototype provides adequate functionality to control processes, with a good flexibility. The configuration can be defined at run time with a few macros, and scripts can be modified easily by hand to add new processes or phases. SMI was well adapted to develop this process control, even if a lack of useful features was noticed, like a timer that has been added externally.

The integration was fast and easy. SMI can be learned quickly (one week). The development and integration, included testing lasted around 5 months. The result is efficient given resources involved, compared to more complex and expensive tools. The new version of the run control is now more flexible to control external processes without any loss of performance.
Most of the remaining work concerns SMI scripts, to feature good error recovery. The basis can be used to develop a more complex and automated control system.

8. References

1. M.Macowicz, ALICE DAQ System Control - User Requirements Document, ALICE Internal Note/DAQ 1998/01.
2. M.Macowicz, ALICE Data Acquisition System Control: Assessment of methods and tools for the development, ALICE Internal Note/DAQ 1998/02.
3. SCADA: http://itcowww.cern.ch/pvss2/index.htm
4. SMI web site: http://delonline.cern.ch/d$onl/smixx/doc/www/SMI.HTML
5. CERN ALICE DAQ group, ALICE DATE User's Guide - DATE V3.7, ALICE Internal Note/DAQ 2000/31.
6. DIM, a Portable, Light Weight Package for Information Publishing, Data Transfer and Inter-process Communication , CHEP 2000, Padova, Italy, February 2000

Appendix 1:DATE SMI Run Control Installation

Upgrade from date v3.7

- Get and Install DATE v3.7
- Get the DATE_SMI_upgrade package

New installation

- Get and install the DATE_SMI package

In any case:

- Get and Install DIM and SMI
  They should be installed in $DATE_ROOT/dim and $DATE_ROOT/smi
  If this is not the case, $DATE/runControl/packageParams has to be modified with the appropriate values.

- A machine has to be chosen to run the DIM domain name server (DNS)

  - run "dns" process on it
  - set the environment variable:
    DIM_DNS_NODE my_dns_node.cern.ch
    should be written in `$DATE_SITE_CONFIG/smi.config` file.
    (The DIM_DNS_NODE variable is currently defined in packageParams.)

The SMI DATE run control can thereafter be started.

Appendix 2: DATE runControl – Some details of implementation

The SMI version of runControl is based on DATE v3.7. It is compatible with DATE v3.7. (only rcServer.sh must be changed, due to a new non-smi feature: `DATE_SITE` is set on LDCs/GDCs when connecting).

The non-SMI control mode is still available. You can set/unset the Tcl variable `SMI_runControl` to select which mode to use. This should be done ONLY before connect. This is implemented by switches everywhere required in the code, to keep both versions compatible and to avoid replicated code. All modifications are tagged by the `#SMI` header.

Only the start_run and stop_run procedures are done in another way, because of their complexity. There is a `SMI_start_run` and a `NOTSMI_start_run` procedure (and the same for stop run). If the non-smi code is changed in one of the procedures, it should be also changed in the other one.

It should be noticed that the current implementation is not completely smi controlled, due to the event builder. In fact, the event-builder daemon has a peer to peer communication with `runControl.tcl`, which forbids a smi control similar to the readout and recorder. This should be changed in next releases, so that the smi start and stop in the tcl code are simpler, synchronous and clean. A good synchronisation between smi and the evB is not possible yet.

A new label has been added in the window, next to the status label. It is called `status_det`, standing for status_detailed. This label displays the phases for connect, start, stop, disconnect. In smi mode, the status displayed is the smi status. It is obtained by polling: there were some problems with the TK library when getting it by callback. The callback functions for smi are implemented in the "smi/tcl library" written in `runControl.c`. It works fine as far as the variable isn't displayed in the window. It could be used by a control routine/watchdog.

Processes architecture

The main idea is that each process of the DATE architecture is modeled by a smi object which reflects its actual state. Commands are sent from the run control through smi. As some backward compatibility is required, and because some critical processes should be kept as they are (readout, recorder...), it was not possible to make each of process an smi-associated process.

Therefore some workarounds were used to keep clean boundaries between a smi run control and some non smi processes.

We can describe the main objects of the architecture and the corresponding processes from top to down (runControl to each subprocess) (see Fig. 3).

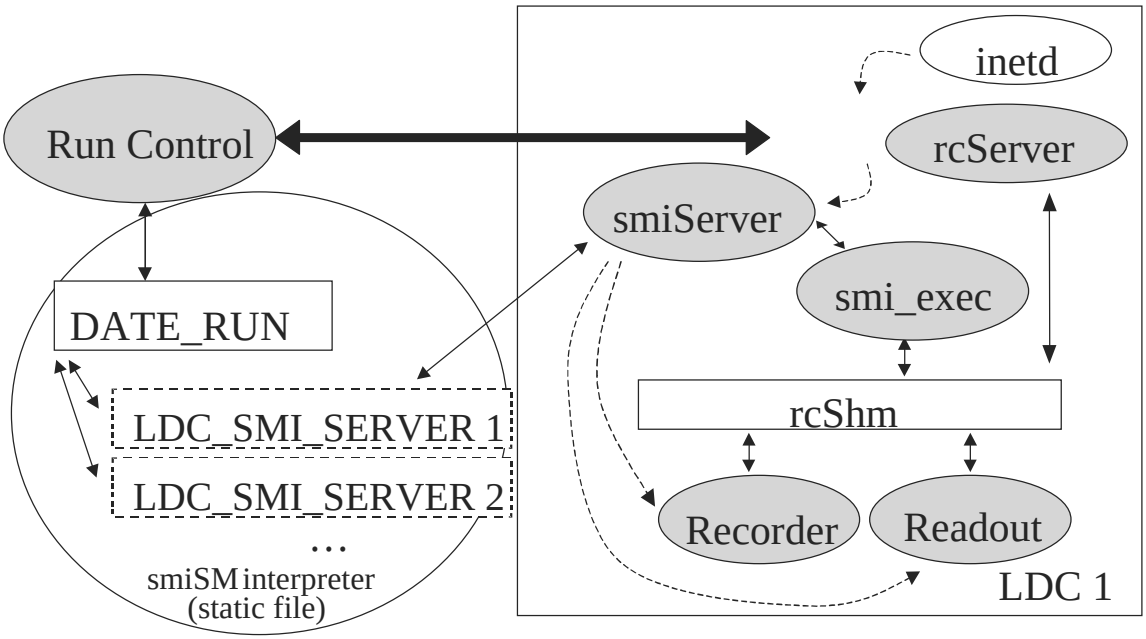

Figure 3 - DATE SMI run control objects and corresponding processes.

The runControl has its behavior reflected by the `DATE_RUN` smi object. Each action on a button (connect, disconnect, start, stop) triggers a smi-action on `DATE_RUN`. The status displayed is the smi status of this object.

For each LDC or GDC, there is a SMI_SERVER object, which communicates with the `DATE_RUN` object. The process associated to the SMI_SERVER object is the "smiServer". This process is the same on all the LDCs and on all the GDCs (there is only one for LDCs and one for GDCs). It is differentiated by its domain name, which is based on the machine name where it runs. "`smiServer`" can understand several commands, like "start" and "stop", and has a well defined number of possible states, known by `DATE_RUN`. Its state summarize the state of all the subprocesses it handles. It does a part of what the `rcServer` did before, when handling `Start` and `Stop` commands.

On a LDC, for instance, `smiServer` controls `readout` and `recorder` processes. If one of these gets error, `smiServer` will come to "`error`" state. The state machine is still relatively simple in this prototype but can be modified with a greater number of states (`readout_error` or `recorder_error` for instance).

The "`readout`" and "`recorder`" DATE processes have their equivalent in smi. The smi object tries to reflect at best the status of the DATE process. There is a mechanism to warn the smi object if the DATE process dies.
Control of `readout` and `recorder` is done through the shared memory segment. Because the smi syntax is not intended to do complex things (e.g. accessing this segment), a SMI_EXEC object has been created. It can execute a number of actions defined in its "library". It has an associated process, which is written in C, so it can do almost everything. This corresponding associated process is named "smi_exec". For instance, a "`set_flag`" function has been written, so that you can set the flag of readout from a smi script. The only thing to do is to ask the SMI_EXEC object to do it for you. The way to communicate with this object is simple. It

can only be "ready" or "error", and handles the actions declared. The smiServer also uses smi_exec, to launch processes for instance.

Users can add any number of new processes like readout and recorder. They will be controled by smiServer, and nothing has to be added elsewhere (e.g. in run control). The library handled by smi_exec can also be extended to new features easily.

To summarize, here is the list of processes running:

Run Control machine:

- DATE runControl
- SMI interpreter interpreting the temporary file created by the runControl.Tcl after expansion of the macros of the runControl.smi script (object DATE_RUN).
- An smi_timeout process is used so that time outs can be handled in smi scripts.

LDCs:

- SMI interpreter for LDC_smiServer.smi script (smi objects: smiServer, smi_exec, readout, recorder, user defined ...)
- smi_exec (smi associated process)
- user defined processes
- DATE readout and recorder
- DATE rcServer

GDCs:

- SMI interpreter of GDC_smiServer.smi script (smi objects: smiServer, smi_exec, eventBuilder, user defined ...)
- smi_exec (smi associated process)
- user defined processes
- DATE rcServer
- DATE evB daemon

In SMI control mode, rcServer is only used to upload parameters and settings to LDCs and GDCs. It also starts the SMI server shell when requested (command SMI_START). This shell starts itself the 2 processes SMI interpreter and smi_exec. It is not used for control otherwise.

Appendix 3: DATE run control files modified

1. Previous system

The control was done by ASCII messages between run control process and rcServers processes running on each machine, via a peer to peer socket based communication.

2. New system

The rcServer is kept for parameter download purposes (rcShm access on every LDC and GDC), but is no more involved in the control. It just starts the SMI servers.
Start/Stop procedures were moved from rcServers to smiServers (launch readout, recorder, ...). The protocol with existing DATE processes (readout and recorder) is kept.

The SMI runControl is developped on top of DATE v3.7

The new source files and the source files modified to support SMI are listed hereafter in the section on DATE_RC directory.
The updated list of files created in the executable directory is indicated in the section DATE_RC_BIN directory.

It should also be noticed that both DIMDIR and SMIDIR are used in this new version of the runControl. They are currently located in /date (/date/dim and /date/smixx). The DIM_DNS_NODE machine is defined in the smi.config file in DATE_SITE_CONFIG directory.

Log files are located in DATE_SITE_LOGS directory. Currently, the output of SMI interpreters are redirected to various log files (for each machine: one for smiServer and one for smi_exec). There is also the smi_runcontrol log file which holds the output for run_control.smi (object DATE_RUN). These log files are generated using the "log_bypage" process that keeps only the end of the logs. Indeed the amount of log is too large and only a small fraction of it is useful. The process log_bypage is started together with the SM interpreter.

3. DATE_RC directory

New source files for SMI runControl	Description
GDC_smiServer.smi	SMI script for GDC
LDC_smiServer.smi	SMI script for LDC
runControl.smi	smi pre-code for runControl (not true SMI: it can include some macros such as: $ALL_LDC, $ALL_GDC ...)
smi_exec.c	code for the SMI associated object smi_exec used by LDC/GDC SMIservers to execute some "C code"
smiStart_runControl.sh	script to launch the SMI runControl and the timeout.
smiStart_LDC_smiServer.sh	script to launch the SMI LDC server and smi_exec.

smiStart_GDC_smiServer.sh	script to launch the SMI GDC server and smi_exec.
smi_timeout.c	code for an SMI associated object which behaves like a timer
log_bypage.c	source code of log_bypage, process used to make log files of defined size
packageParams.Linux	SMI configuration is platform dependent
packageParams.SunOS	SMI configuration is platform dependent

Files modified	Modifications
rcServer.sh	- the shell prompts for the DATE_SITE environment variable and sets it. This is sent by runControl at CONNECT - SMI configuration file smi.config to define the environment variables for DIM.
rcServer.c	- cleaning subprocesses when exiting
rcServer.tcl	- SMI - still fully compatible with Date v3.7
runControl.c	- SMI library functions to be accessed from runControl.tcl
runControl.tcl	- see file for modifications
Makefile	- making of new files - linking to SMI and DIM libraries where required
packageParams	- SMI configuration added The DIM_DNS_NODE variable is currently defined here. A better place to define would be in smi.config. - A special set of libraries for the building of smi_control. It could be inserted in the gmake command or in the GNU_makefile.
runControl.sh, runControl.csh	SMI and platform specific configuration added

4. DATE_RC_BIN directory

New files	Description
GDC_smiServer.sobj	smi compiled code for GDC smiServer
LDC_smiServer.sobj	smi compiled code for LDC smiServer
log_bypage	Simple process that writes its standard input a file from which maximum size can be specified. When max size

	is reached, the 1st part of the file is deleted. It enables to keep a track of the last few kilobytes of log produced by the very verbose SMI interpreters running. As this log is used only for hard issues, there is no need to keep it all, just the end is useful.
runControl.smi	The same as the source file
rc_tmp.smi	Temporary file created by runControl.tcl at CONNECT.The macros in run_control.smi are substituted by the right values: $ALL_LDC by the actual list of LDCs... It is then "compiled" to runControl.sobj
runControl.sobj	Read explanation above
smi_exec	SMI associated process to smi_exec SMI object, used in LDC/GDC smi_servers
smi_timeout	SMI associated process to TIMER SMI objects.
smiStart_GDC_smiServer.sh smiStart_LDC_smiServer.sh	The same as for the files in the smiStart_runControl.sh source directory

5. DATE_SITE_CONFIG directory

smi.config	This files holds the environment variables specific to a DATE setup. At the present time, this is only the DIM_DNS_NODE variable, which tells which host runs the dim dns server.