ECMWF Feature article

from Newsletter Number 129 – Autumn 2011

COMPUTING

Managing work flows with ecFlow

www.ecmwf.int/en/about/news-centre/media-resources

doi:10.21957/nr843dob

This article appeared in the Computing section of ECMWF Newsletter No. 129 – Autumn 2011, pp. 30–32.

Managing work flows with ecFlow

Avi Bahra

For almost three decades ECMWF has used the SMS (Supervisor Monitoring Scheduler) package to control the workflow for the Centre's operational models and systems. Written at ECMWF, it allows the design, submission and monitoring of jobs both in the Research and Operations Departments, and provides common tools for scientists, analysts and operators to cooperate. A large number of organisations also use SMS (from both Member and non-Member States). Development of SMS has now stopped. The software will be supported only on the currently tested computer platforms, not on any new ones.

The replacement, ecFlow, will supersede SMS in the near future. It is a complete rewrite using object oriented methodology and modern standardised components; it will act as a comprehensive replacement for SMS. The rewrite will help improve maintainability, allow easier modification and introduce object orientated features. Proprietary script languages, such as CDP, have been replaced by Python.

Once ecFlow has been fully validated at ECMWF, it will be made available to Member States.

Key features of ecFlow

ecFlow is a work-flow manager that enables users to run a large number of programs, with dependencies on each other and on time, in a controlled environment. It provides tolerance, for hardware and software failures, combined with good restart capabilities.

ecFlow runs as a server receiving requests from clients. The command line interface, the graphical interface (ecFlowview), scripts and the Python API (application interface) are the clients. The server is based on C++/boost ASIO and uses TCP/IP for communication. Multiple servers can be run on the same hardware. ecFlow submits tasks (jobs) and receives acknowledgements from tasks via specific commands embedded in the scripts. The relationship between tasks is stored in ecFlow, and it is able to submit tasks dependent on the status of other tasks and attributes like time.

The command line interface for ecFlow allows the suite definition to be loaded and retrieved from the server. Also it provides a rich set of commands for communication with the server and provides similar capabilities to SMS. For example, the Python API allows the entire suite definition structure to be specified and loaded into the server. A suite is a collection of interrelated tasks. In ecFlow suites are described by a definition file. The Python API also provides functionality for client to server communication. In addition, it allows checking of the suite, testing the defined interrelations between tasks, and other references and limits.

ecFlow comes with a user manual, online tutorial, Python API and reference documentation.

Other features of ecFlow

ecFlow is written to be as platform independent as possible by using standardised libraries. We have built and are testing ecFlow on the following operating systems: Linux (SUSE 10.3/11.3 and SLES 11), HP-UX (11.23) and AIX (5.3).

Another feature of ecFlow is the simulator. It can validate suites before the real job submission. The simulator can be run without the need for scripts or the server. Some suites have attributes that allow it run without ever completing; however others, that are known to complete, can be checked for programming deadlocks with the simulator. An interim graphical interface to ecFlow, ecFlowview, is used to visualise and manage the suite hierarchy.

'Child commands' are embedded in submitted scripts that communicate with the ecFlow server. It supports variable inheritance and also has a white list file for authentication of read/write client requests. It shares the ability to dynamically add or remove whole suites on the fly using the command line interface or the Python API.

The default script extension has been changed to reflect the software change with the ".ecf" extension used rather than ".sms". There is a pre-processor to handle include files and variable substitution in ecFlow, though SMS variables have been replaced with ECF_ labelled equivalents (e.g. the SMS variable SMSHOME is replaced with ECF_HOME). These naming conventions are easy to migrate by using a script to substitutes one set of names with the other. In this way the investment users have made in developing their SMS suites can be preserved.

 doi:10.21957/nr843dob

Design features

With the use of standardised libraries and by designing it in an object orientated way, maintenance and enhancement of ecFlow is easier. ecFlow has been written and developed with C++ and Python. The development of ecFlow has been test driven, with a large set of unit and regression tests. These tests validate the expected behaviour and detect regressions in functionality and performance. They are run on each of the supported platforms. ecFlow also works on 64-bit operating systems like AIX and Linux.

The command line interface, along with the Python API, replaces the CDP text interface. The Python API provides two main functions. It allows the suite definition to be built and it provides functionality for interaction between the client and server. This enables the suite definition to be loaded and retrieved from the server. However, the use of Python is not mandatory: the suite definition file can be built using any language and loaded into the ecFlow server via the command line interface.

When a task loses communication with its submitting server it is called a zombie. ecFlow has been extended to allow for customisable handling of zombies. A new attribute specifies how zombies are handled by the client or server. For example, we can add a zombie attribute so that child label commands no longer block scripts from progressing. This can be added at different levels in the node tree hierarchy.

The log file in ecFlow records each request and any resulting state changes. We can quickly determine the number of requests per minute that the server is handling. This allows for better load management (i.e. by splitting the load onto multiple servers). Additionally the ecFlow client can produce a graphical plot (using gnuplot) showing the proportional impact of child versus user requests and the total number of requests per minute.

ecFlow has also been designed with better error checking, such as the following.

- Trigger expressions checking is improved. When a definition is loaded external references are checked in the server. This provides early warnings on trigger expression that never evaluate.
- Job generation can be checked before loading the definition into the server. This will check that script can be located, the pre-processor can expand the include files and variable substitution works. Recursive use of “include” files is also detected and flagged.
- References to limits are checked to ensure they exist. Also the tokens specified on the in-limit are checked to make sure that they are not larger than the limit size.
- Correct by construction approach is used when building the suite definition with the Python API. See Box A for an example.
- The abort child command now has an option to allow a reason for the abort to be provided, thus facilitating better handling of script problems.
- Definition files can be simulated without the need for underlying scripts or server.

Examples of expression, job and deadlock checking are shown in Box B.

Correct by construction

A

The following examples show usage of the new Python API. The API supports a correct by construction approach. For example, adding tasks of the same name at the same level will throw a RuntimeError exception:

Example 1

```
import ecflow
defs = ecflow.Defs()
suite = defs.add_suite(“s1”)
suite.add_task(“t1”)
suite.add_task(“t1”)
# RuntimeError exception thrown
```

>> RuntimeError: Add Task failed: A task of name ‘t1’ already exist on node SUITE:/s1

Example 2

Adding dependencies, such as dates, are also checked:

```
import ecflow
defs = ecflow.Defs()
suite = defs.add_suite(“s1”)
task t1 = suite.add_task(“t1”)
t1.add_date(1,14,2007)
# day,month,year, month is not valid
```

>> IndexError: Invalid Date(day,month,year): the month >=0 and month <= 12, where 0 means wild card

Checking

B

Expression checking

Some checking has to be deferred until the definition is fully defined. Here is a simple example showing the checking of trigger expressions:

```
import ecflow
defs = ecflow.Defs()
suite = defs.add_suite("s1");
suite.add_task("t2")
suite.add_task("t1").add_trigger("t2 == active)")
assert len(defs.check()) != 0, "Expected Error:
miss-matched brackets in expression."
```

Job checking

Job creation is the process of locating an '.ecf' script corresponding to a task and then generating a job file. This can be checked before a definition is loaded into the server using the Python API.

```
# Generate jobs for the *ALL* tasks in the
definition given by variable 'defs'
# and print errors to standard out.
import ecflow
defs = ecflow.Defs()
suite = defs.add_suite("s1");
suite.add_task("t1")
job_ctrl = JobCreationCtrl()
defs.check_job_creation( job_ctrl )
print job_ctrl.get_error_msg()
```

Job control provides additional functionality to control which nodes are generated and control over the directory used for job generation.

Dead lock checking

Simulation allows a suite definition to be checked without the need for scripts or a server. By default the simulation will run for a year before quitting. This can take a couple a seconds to a few minutes depending on the complexity of the suite definition. However, it is most useful where we have a definition which is known to complete. Here is an example which will cause a deadlock that is detectable by the simulator.

```
import os
from ecflow import *
defs = Defs()
suite = defs.add_suite("dead_lock")
fam = suite.add_family("family")
fam.add_task("t1").add_trigger("t2 == complete")
fam.add_task("t2").add_trigger("t1 == complete")
theResult = defs.simulate();
assert len(theResult) != 0, "Expected simulation
to return errors, but found none"
print theResult
os.remove("defs.depth") # provides reason why
simulation could not complete  os.remove("defs.
flat") # provides reason why simulation could
not complete
```

Early checking of the suite definition will help to speed up the development of suites.

Incompatibilities

ecFlow is a replacement for SMS providing similar capabilities. Hence not all the functionality will be present in the first release. Currently auto-restore and auto-migrate constructs have not been implemented. This functionality might be added in a future version. Some SMS constructs (i.e. owner, action, text and abort) are no longer supported.

ecFlow does not provide, looping and conditional statements or function definitions. However, since we will have a defined file format, any language can be used to generate the structure of the definition file. Python integration replaces CDP and will provide looping and conditional statements and functions. It should be noted, however, that the new Python interface offers the opportunity to re-design and re-write suite definition in an object-oriented way. This has the distinct advantage of allowing a more compact, more easily maintainable and re-usable representation of suite definitions.

Migration

ecFlow is currently undergoing internal validation and testing by the Operations Department at ECMWF. A series of internal and external training courses have been organised for September 2011 to give initial access to a beta version of the software. These courses are designed for current users of SMS who want to migrate to ecFlow. These initial user tests should identify any missing features or issues with the new software that could be added or fixed before the general release of the software early in 2012. The computer user training course organised for Member States in spring 2012 will only be covering the ecFlow software.

The migration of ECMWF Operations is closely linked with the migration of the ECMWF Research environment (especially PrepIFS) as Operations and Research are sharing the same technical framework. It is planned to use the migration process to harmonise the suite definition and scripts files even further, which should allow an improved transfer of changes between Operations and Research. It is planned to finish this migration process and remove any operational dependencies on SMS by the end of 2012.

doi:10.21957/nr843dob

Technology used

Building ecFlow uses the latest C++ compilers. Currently we have tested building with the following compilers: gcc version 4.2.1/4.5 on Linux (SUSE 10.3/11.3), aCC-A.06.20 on HP-UX and xlC version 11.1 on AIX(ibm_power6/rs6000). We have also made use of Python version 2.5–2.7.

To support the generation of online documentation we use Sphinx-poco. This allows the online tutorial Python samples to be compiled separately and to be used in the documentation. Sphinx also supports documentation of the Python API by loading the ecFlow Python extension and automatically extracting the doc strings.

We use the Boost C++ libraries. They are a set of free peer-reviewed portable C++ source libraries in preparation for inclusion in future C++ standard libraries. These libraries are described in Box C.

To build the project we use the bjam command-line tool that drives the Boost Build system. It provides the build and test mechanism that allows ecFlow to be constructed in the same way across different platforms. In addition, it allows debug, release and profile builds without knowledge of the compiler options on each platform.

Boost C++ libraries used in ecFlow

C

Boost-ASIO library is used to provide the core of the client-server implementation. It also provides a deadline timer for polling and support for time outs.

- Boost-Python library enables seamless interoperability between C++ and the Python programming language.
- Boost-Program options are used to parse the client and server program options, and they provide the corresponding help strings.
- Boost-Spirit provides the parsing for expressions. The abstract syntax tree is then created from the spirit nodes.
- Boost-Test library is used for unit and regression tests of the C++ and Python code.
- Boost-Date-Time library is used in the suite calendar, time based attributes and polling.
- Boost-File System library is used for file queries in a platform independent manner.

Future developments

The ecFlow user interface (ecFlowview) uses the same technology and user paradigm as XCDP. The graphical user interface (GUI) is X-Windows based. This places limits on the machines where user interface can be run, since the X windows libraries will require a separate installation. This will be replaced in the near future, with a more intuitive and up to date GUI.

When ecFlow is released the online documentation and tutorial can be accessed from:

- http://www.ecmwf.int/publications/manuals/ecflow/

We welcome feedback on any aspect of ecFlow. Suggestions for further developments will be assessed and, where appropriate, they will be prioritised and fed into future versions.

doi:10.21957/nr843dob