Plan Execution Interchange Language (PLEXIL)

Software User's Manual

Table of Contents

PURPOSE 5
PLEXIL and Safety-Critical Systems 5
User Instruction 5

Notices:

License:

Disclaimers:

HARMLESS AGREEMENT WITH ANY ENTITIES THAT ARE PROVIDED WITH THE SOFTWARE AND/OR TECHNICAL DATA.

NOTWITHSTANDING THE ABOVE, WHEN RECIPIENT IS PROHIBITED BY LAW FROM PROVIDING INDEMNIFICATION, THE INDEMNIFICATION REQUIREMENTS SPECIFIED ABOVE SHALL NOT APPLY TO RECIPIENT.

PURPOSE

Plan Execution Interchange Language (PLEXIL) is a Core Flight System (or Core Flight Software) (cFS) based application, developed by the Autonomy Operating System project at NASA. cFS is open-source, reusable flight software, also developed by NASA and targeted to space missions. PLEXIL runs within the cFS framework, and is a wrapper integrating the open source PLEXIL Executive into the cFE infrastructure. Software features include:

- Executes PLEXIL plans stored on local file system
- Can be event-driven or clock-driven
- Intended for "soft real time" discrete control

PLEXIL and Safety-Critical Systems

PLEXIL was not developed to the software standards required to run on and diagnose safety-critical systems. It is research-level software, Class E and non-safety critical, in the NASA terms defined by its software procedural requirements document NPR 7150.2. It is released for use in non-critical research systems and demonstration projects. PLEXIL has gone through some internal software engineering procedures, such as requirements verification and internal code reviews, so it is hoped that the software runs as expected, but these were not done to the level required for higher classifications by NPR 7150.2.

User Instruction

Getting the Code:

PLEXIL is expected to be released for general source release by the NASA software office. To get PLEXIL, start at http://software.nasa.gov and search for "PLEXIL". Follow the instructions on the page to request the software. Additional information about the PLEXIL Executive can be found at http://plexil.sourceforge.net/wiki/index.php/Main_Page.

Integrating PLEXIL into the cFS build:

Prebuilding the PLEXIL distribution for your target platform is recommended. The files `plexil/scripts/update-plexil32.sh` and `plexil/scripts/update-plexil64.sh` are used to check out and build the PLEXIL Executive libraries and tools required by the PLEXIL cFE app for a Linux cFE environment. They install the PLEXIL includes into `plexil/include`, the libraries into `plexil/lib` and `plexil/lib64` respectively, and the compiler into `plexil/jars`. These products can be checked into a source code repository for convenience.

To build the PLEXIL cFE app, simply add the target `plexil` to the list of CMake targets for one or more CPUs.

The CMake script file `cfe/cmake/plexil.cmake` implements a CMake rule `add_plexil_plans` for compiling PLEXIL plans during the cFS build process, and installing them into the cFS image.

The file `apps/plexil/fsw/mission_inc/interface.xml` is a sample interface configuration file for the PLEXIL cFE app.

Interfacing to PLEXIL step-by-step guide:

Every cFS project will require some custom interfacing. This section explains how to implement it.

See the include files in `plexil/include` and `aos/apps/plexil/fsw/src` for the PLEXIL classes and types named in this guide.

1. PLEXIL has a limited repertoire of types. It has:

   - `Boolean` (true/false);
   - `Integer` (32-bit signed int);
   - `Real` (IEEE double float);
   - `String` (C++ `std::string`);

   and one-dimensional arrays of each of the preceding types (`BooleanArray`, `IntegerArray`, `RealArray`, `StringArray`).

2. The `Value` class is a *discriminated union* capable of representing any of the above, as well as some types used only internally. The information coming from and going to the PLEXIL cFE app must be expressible in `Value` instances.
3. PLEXIL has well-defined APIs for querying and commanding external devices and the environment.

   - A `State` is a tuple of a lookup name (a String), and zero or more `Value` instances representing parameters to the lookup call.
   - A `Command` also has a name and zero or more parameters, and is associated with a command status value (the enumeration `CommandHandle`) and an optional *return value*.
   - The abstract base class `InterfaceAdapter` is the API seen by the PLEXIL Exec for interfacing to the outside world.
   - The abstract base class `AdapterExecInterface` defines the API for getting data back into PLEXIL from the external environment.

4. The PLEXIL cFE app - specifically the `CfeAdapter` class - uses table lookups to dispatch commands and to interpret incoming cFE Software Bus messages. These handlers are C++ functions with the following signatures:

   - Command handlers: `void myCommandHandler(Command *cmd, AdapterExecInterface *intf);`
   - Message handlers: `void myMessageHandler(CFE_SB_Msg_t *msgHdr, AdapterExecInterface *intf);`

5. A command handler should digest the command parameters, perform the requested function, and send back command acknowledgement to the PLEXIL Exec. A typical command handler might look like this:

```
#include "AdapterExecInterface.hh"
#include "Command.hh"

void my_command_handler(Command *cmd, AdapterExecInterface *mgr)
{
  // Extract parameters from command
  Integer n;
  Real x;
  cmd->getArgValues()[0].getValue(n);
  cmd->getArgValues()[1].getValue(x);
  // Perform the command
  call_my_subsys(n, x);
  // Acknowledge the command has been sent
  mgr->handleCommandAck(cmd, COMMAND_SENT_TO_SYSTEM);
}
```

6. A message handler should extract the relevant information from the cFE Software Bus message, turn it into Value instances, and send it back to the Exec. A typical message handler might look like this:

```
#include "AdapterExecInterface.hh"
extern "C" {
#include "cfe_sb.h"
}

void my_message_handler(CFE_SB_Msg_t *msgHdr, AdapterExecInterface *mgr)
{
  // Extract data from message
  const My_Msg_t *myMsg = (My_Msg_t *) msgHdr;
  Value n = (Integer) myMsg->my_subsys_n;
  Value x = (Real) myMsg->my_subsys_x;
  Value ret = (String) myMsg->my_command_result;
  // Send it to PLEXIL
  mgr->handleValueChange(State("MySubsysN"), n);
  mgr->handleValueChange(State("MySubsysX"), x);
  // Send a command return value
  mgr->handleCommandReturn(stored_cmd, ret);
  // Acknowledge the command has been completed
  mgr->handleCommandAck(stored_cmd, COMMAND_SUCCESS);
}
```

7. Register your handlers with the PLEXIL cFE app:

```
#include "CfeAdapter.hh"

void init_my_handlers()
{
  // Set message and command handlers
  CfeAdapter &adapter = CfeAdapter::instance();
  adapter.setMessageHandler(my_message_handler, MY_MESSAGE_MID);
```

```
  adapter.setCommandHandler(my_command_handler, "MY_COMMAND");
}
```

Add a call to `init_my_handlers()` to the function `plexil_initialize_interfaces()` in `apps/plexil/fsw/src/exec_app.cc`, after the call to `theApp->initialize()`.