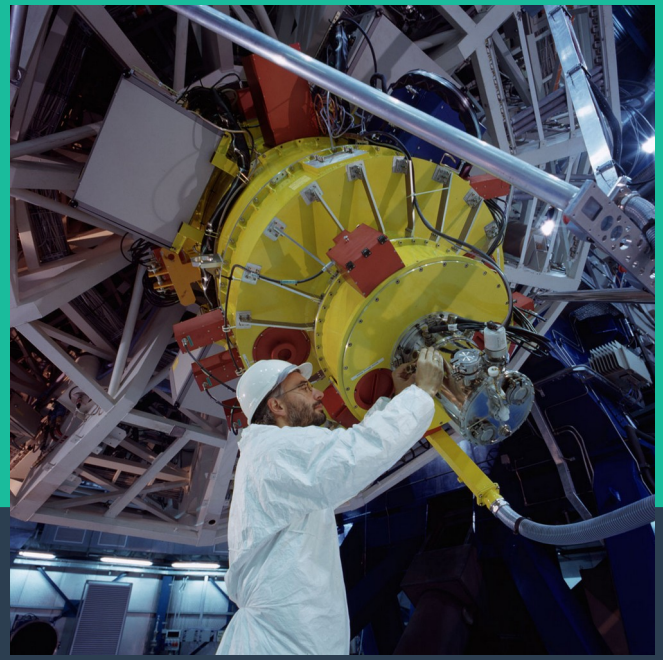
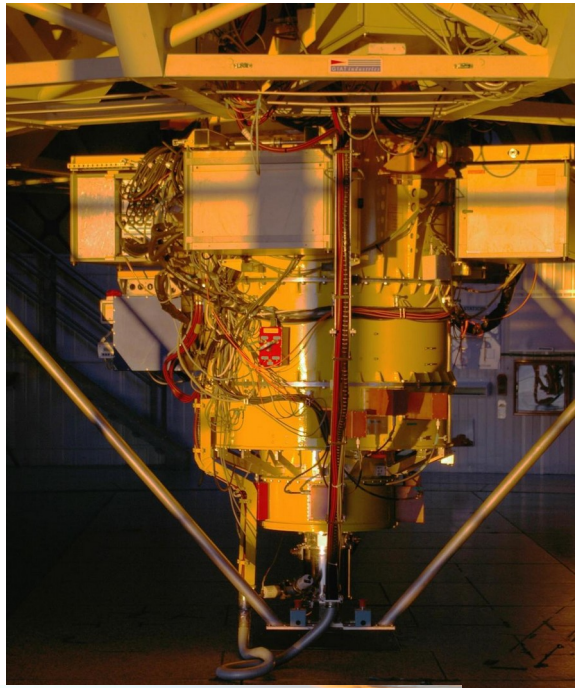
Special devices on ELT-SW: lessons learned on FORS1



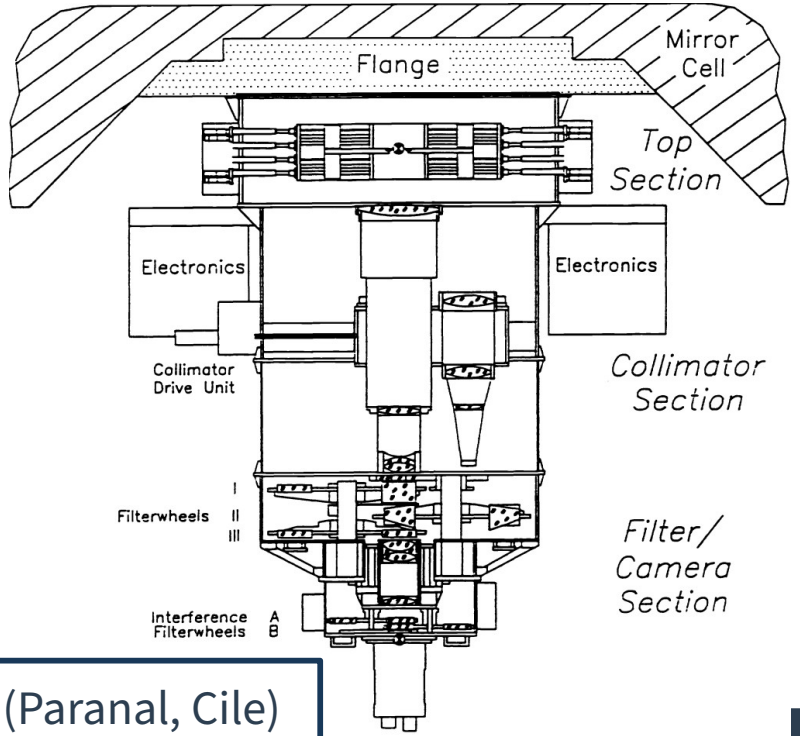
Giorgio Calderone, Paolo Di Marcantonio

INAF OATs

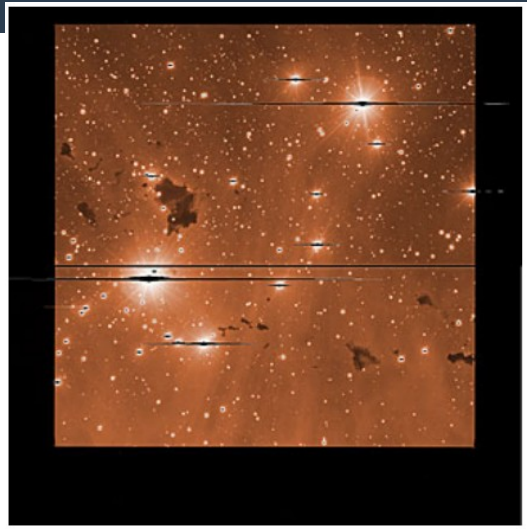
FORS2 (FOcal Reducer and low dispersion Spectrograph)



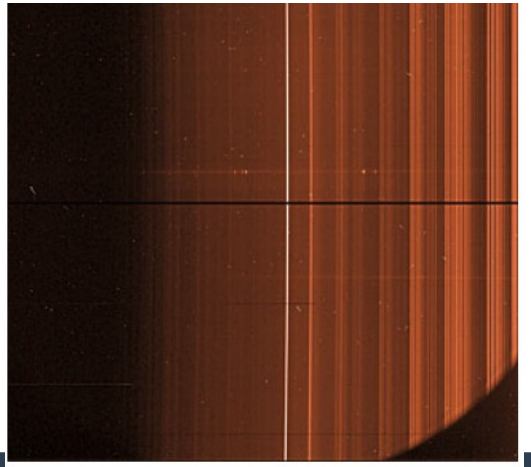
Imager (0.125''-0.25''/pixel, FOV: 7'x7')
Spectrograph (R=260-2600)
Wavelength range: 330-1100 nm



ESO/VLT (Paranal, Cile)

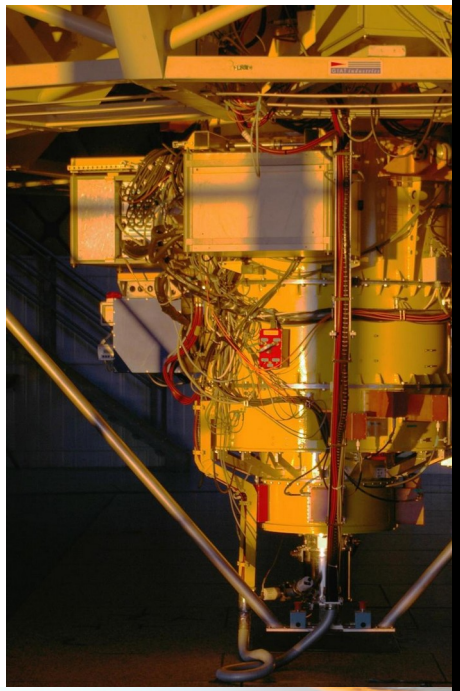


IC 2944 with FORS2



A raw spectrum obtained with FORS

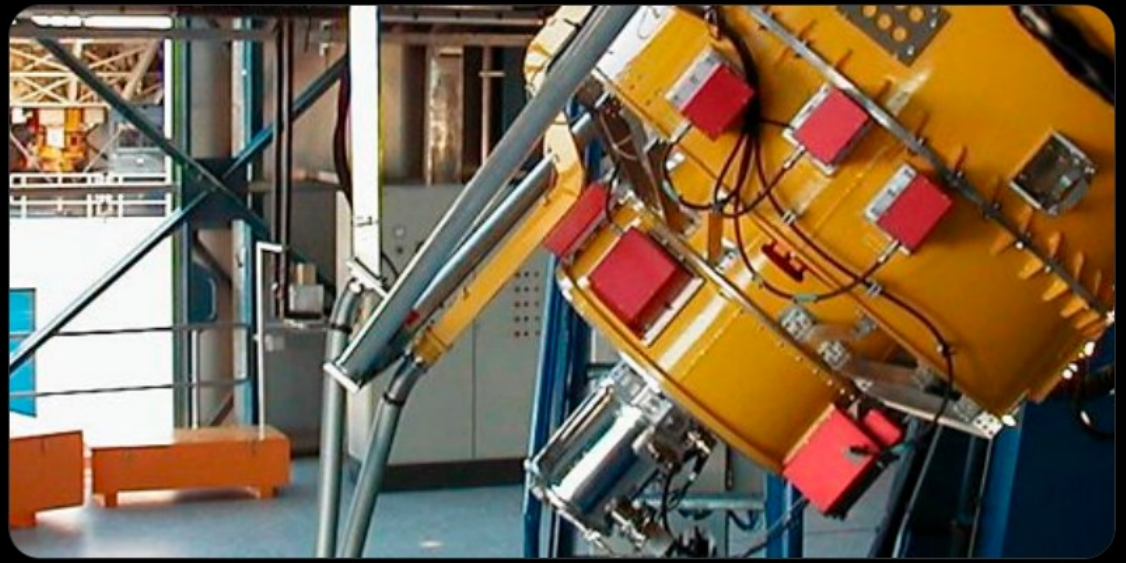
FORS2 (FOcal Reducer and low dispersion Spectrograph)



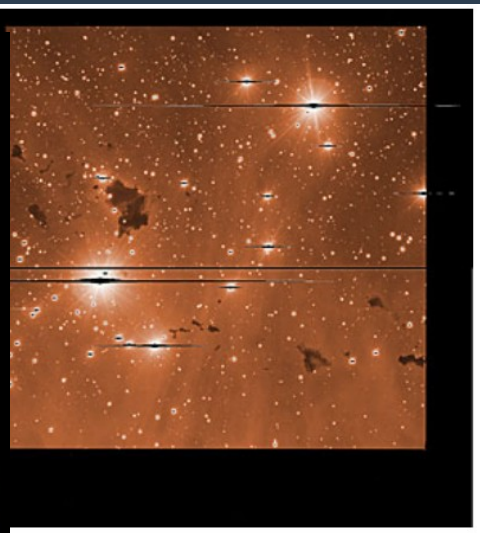
← Tweet

ESO @ESO

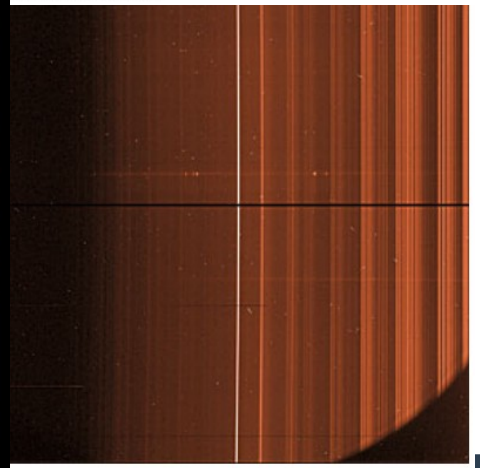
Of all instruments at Paranal, this one is the Swiss Army knife. Read more about FORS2 on UT1 socsi.in /n90Ve



5:00 PM · Aug 14, 2017 · Orlo



44 with FORS2



spectrum obtained with FORS

FORS timeline

FORS1 (1999-2009)

Parts are now in Trieste for the FORS-Up project

FORS2 (2000-ongoing)

Refurbished in 2009

FORS1 *(outcome of FORS-Up)*

Operations: 2025

Will replace FORS2

Sala integrazione (Basovizza, OATs)

FORS1 ed ELT-SW

FORS1 sarà il primo strumento ad utilizzare gli standard ESO-ELT:

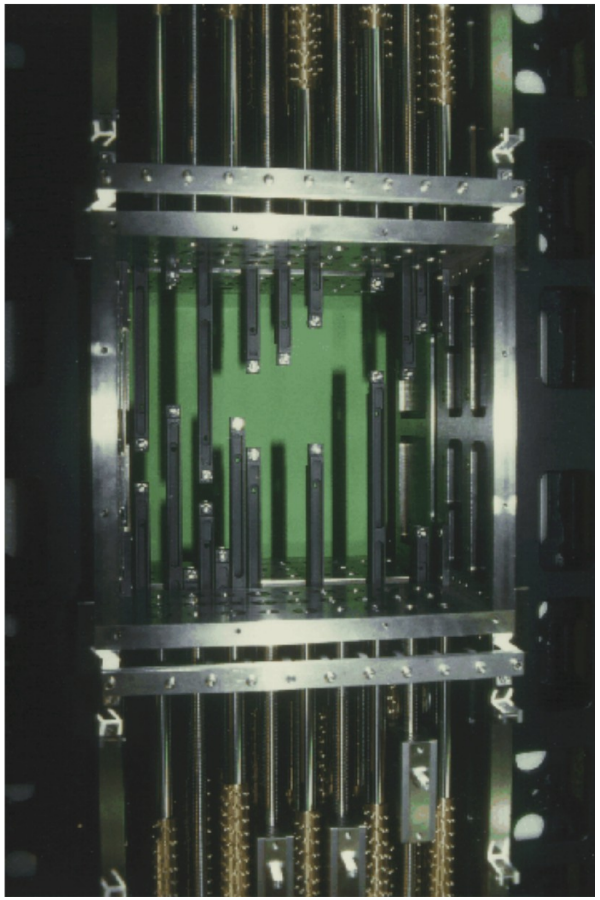
- Nuovo framework per il software di controllo (ELT-SW IFW v4);
- Nuovo framework per la gestione del detector scientifico (NGC-II);
- Nuovo framework per la comunicazione fra ambienti ELT e VLT (gateway);
- Nuovo framework per lo sviluppo di codice PLC (→ *talk Antonio*).

Multi-Object Spectroscopy with movable slitlets (MOS)

Long slit spectroscopy on up to 19 targets simultaneously:

- **38 blades / 19 pairs;**
- **Each blade should be positioned independently:**
  - ➔ *Set blade position;*
- **Each pair is a slit:**
  - ➔ *Set center and width;*
- **Avoid collisions!**
  - ➔ *PLC;*

Implementation of a special device is required.



MOS special device implementation

Standard RPC methods:

- RPC_Init()
- RPC_Enable()
- RPC_Disable()
- RPC_Stop()
- RPC_Reset()

MOS-specific RPC methods:

- RPC_MoveBlade (blade, position)
- RPC_SetSlit(center, width)
- RPC_ParkBlades()

MOS-specific state machine:

- “Presetting” → “Positioning”;
- “Tracking” state not needed;
- New state “Approaching”;
- New methods to change state.

(changes applies to PLC code and simulator);

How to generate files from IFW template?

```
cookiecutter ifw-templates-v1.0.1/project
project_name [myproject]: fors1
project_description [my project description]:
project_prefix [xxx]: f1
component_name [mycomponent]: fcs
device_name [mydevice]: mos
```

- Use “cookiecutter”
- (generates 28 files: device configuration, FCS/client and WS/PLC communication, simulator);
- Multiple special devices? need independent simulators (wscript), hence:
  - “f1-ics/fcs/devsim/src” becomes “f1-ics/fcs/devsim/**mos/**src”;
- Manually add generated files to your tree.

```
(base) algenib eltdev:~/XXX/IFW4/fors1 1006 > find . -iname '*mos*'
./resource/nomad/mos1.yaml.tpl
./resource/nomad/mos1sim.nomad
./f1-ics/fcs/clib/resource/schema/mos_schema.json
./f1-ics/fcs/clib/src/f1/fcs/clib/mos_setup.py
./f1-ics/fcs/devices/src/include/fcs/devices/mos.hpp
./f1-ics/fcs/devices/src/include/fcs/devices/mosConfig.hpp
./f1-ics/fcs/devices/src/include/fcs/devices/mosEncDec.hpp
./f1-ics/fcs/devices/src/include/fcs/devices/mosHwErrors.hpp
./f1-ics/fcs/devices/src/include/fcs/devices/mosLcsIf.hpp
./f1-ics/fcs/devices/src/include/fcs/devices/mosLcsIf.ipp
./f1-ics/fcs/devices/src/include/fcs/devices/mosRpcErrors.hpp
./f1-ics/fcs/devices/src/mos.cpp
./f1-ics/fcs/devices/src/mosConfig.cpp
./f1-ics/fcs/devices/src/mosLcsIf.cpp
./f1-ics/fcs/devsim/resource/config/fcs/devsim/mos
./f1-ics/fcs/devsim/resource/config/fcs/devsim/mos/mos.namespace.yaml
./f1-ics/fcs/devsim/resource/config/fcs/devsim/mos/mos.scxml.xml
./f1-ics/fcs/devsim/resource/config/fcs/devsim/mos/mos1.cfg.yaml
./f1-ics/fcs/devsim/resource/config/fcs/devsim/mos/mos1.namespace.xml
./f1-ics/fcs/devsim/src/fcs_devsim_mos
./f1-ics/fcs/devsim/src/fcs_devsim_mos/mos.py
./f1-ics/fcs/devsim/src/fcsDevsimMos.py
./f1-ics/fcs/gui/wdglib/src/include/fcs/gui/wdglib/mos.h
./f1-ics/fcs/gui/wdglib/src/mos.cpp
./f1-ics/fcs/gui/wdglib/src/mosWdg.ui
./f1-ics/fcs/server/resource/config/fcs/definitions/mos.yaml
./f1-ics/fcs/server/resource/config/fcs/definitions/mosMap.yaml
./f1-ics/fcs/server/resource/config/fcs/devices/mos1.yaml
./f1-ics/fcs/server/resource/config/fcs/mapping/mos.yaml
```

Invoke an RPC with arguments

```
void MosLcsIf::SetSlit(double center, double width) {
      LOG4CPLUS_TRACE_METHOD(m_logger,__PRETTY_FUNCTION__);

      LOG4CPLUS_INFO(m_logger, "[" << m_config->GetName() << "] "
                     << "Executing RPC_SETSLIT ...");
      fcf::common::VectorVariant attr_list;
      std::string obj;
      std::string proc;
      proc = m_config->GetProcId(GetMapValue(fcf::devmgr::common::CAT_RPC,
                                             RPC_SETSLIT));
      obj = m_config->GetObjId();
      try {
        fcf::common::Variant center_var = center;
        fcf::common::PairVariant pair1(std::string(""), center_var);
        attr_list.push_back(pair1);

        fcf::common::Variant width_var = width;
        fcf::common::PairVariant pair2(std::string(""), width_var);
        attr_list.push_back(pair2);

        ExecuteRpc(obj, proc, attr_list);
        LOG4CPLUS_INFO(m_logger, "[" << m_config->GetName() << "] "
                       << "Successfull call of Mos SetSlit: ");
      }
      catch (std::exception& e) {
        const std::string msg = "[" + m_config->GetName()
            + "] SetSlit failure: " + e.what();
        LOG4CPLUS_ERROR(m_logger, msg);
        throw std::runtime_error(msg);
      }
    }
```

- **Consider RPC_SetSlit(center, width)**
- **Use the following types:**
  - fcf::common::Variant
  - fcf::common::PairVariant

Invoke an RPC with arguments

```
void MosLcsIf::SetSlit(double center, double width) {
      LOG4CPLUS_TRACE_METHOD(m_logger,__PRETTY_FUNCTION__);

      LOG4CPLUS_INFO(m_logger, "[" << m_config->GetName() << "] "
                     << "Executing RPC_SETSLIT ...");
      fcf::common::VectorVariant attr_list;
      std::string obj;
      std::string proc;
      proc = m_config->GetProcId(GetMapValue(fcf::devmgr::common::CAT_RPC,
                                             RPC_SETSLIT));
      obj = m_config->GetObjId();
      try {
        fcf::common::Variant center_var = center;
        fcf::common::PairVariant pair1(std::string(""), center_var);
        attr_list.push_back(pair1);

        fcf::common::Variant width_var = width;
        fcf::common::PairVariant pair2(std::string(""), width_var);
        attr_list.push_back(pair2);

        ExecuteRpc(obj, proc, attr_list);
        LOG4CPLUS_INFO(m_logger, "[" << m_config->GetName() << "] "
                       << "Successfull call of Mos SetSlit: ");
      }
      catch (std::exception& e) {
        const std::string msg = "[" + m_config->GetName()
            + "] SetSlit failure: " + e.what();
        LOG4CPLUS_ERROR(m_logger, msg);
        throw std::runtime_error(msg);
      }
    }
```

- **Consider RPC_SetSlit(center, width)**
- **Use the following types:**
  - fcf::common::Variant
  - fcf::common::PairVariant

JSON schema (one level, no validation)

```
"type": "object",
"properties": {
    "action": {
        "type": "string",
        "enum": ["SETSLIT", "MOVEBLADE", "PARKBLADES"],
        "description": "Mos action."
    },
    "center": {
        "type": "number",
        "description": "Slit center"
    },
    "width": {
        "type": "number",
        "description": "Slit width"
    },
    "blade": {
        "type": "number",
        "description": "Blade A (1) or blade B (2)"
    },
    "pos": {
        "type": "number",
        "description": "Blade position"
    }
},
"required": ["action"]
```

JSON schema (one level, no validation)

```
"type": "object",
"properties": {
    "action": {
        "type": "string",
        "enum": ["SETSLIT", "MOVEBLADE", "PARKBLADES"],
        "description": "Mos action."
    },
    "center": {
        "type": "number",
        "description": "Slit center"
    },
    "width": {
        "type": "number",
        "description": "Slit width"
    },
    "blade": {
        "type": "number",
        "description": "Blade A (1) or blade B (2)"
    },
    "pos": {
        "type": "number",
        "description": "Blade position"
    }
},
"required": ["action"]
```

```
{"id": "mos1", "param": {"mos": {"action": "SETSLIT", "center":2, "width":10}} }
{"id": "mos1", "param": {"mos": {"action": "MOVEBLADE", "blade":1, "pos":10}} }
{"id": "mos1", "param": {"mos": {"action": "PARKBLADES"}}}
```

JSON schema (Multi-level, with validation)

```
"type": "object",
"properties": {
    "SETSLIT": {
        "description": "Set slit center and width.",
        "type": "object",
        "properties": {
            "center": { "type": "number", "description": "Slit center", "minimum": -120, "maximum": 120 },
            "width":  { "type": "number", "description": "Slit width", "minimum": 0, "maximum": 10 }
        },
        "required" : ["center", "width"]
    },
    "MOVEBLADE": {
        "description": "Move a single blade.",
        "type": "object",
        "properties": {
            "blade": {"type": "number", "description": "Blade ID", "minimum": 1, "maximum": 2},
            "pos": {"type": "number", "description": "Slit width", "minimum": -120, "maximum": 120}
        },
        "required" : ["blade", "pos"]
    },
    "PARKBLADES": {
        "type": "object",
        "description": "Park both blades.",
        "properties": {}
    }
},
"anyOf": [
    {"required" : ["SETSLIT"]},
    {"required" : ["MOVEBLADE"]},
    {"required" : ["PARKBLADES"]}
]
```

JSON schema (Multi-level, with validation)

```
"type": "object",
"properties": {
    "SETSLIT": {
        "description": "Set slit center and width.",
        "type": "object",
        "properties": {
            "center": { "type": "number", "description": "Slit center", "minimum": -120, "maximum": 120 },
            "width":  { "type": "number", "description": "Slit width", "minimum": 0, "maximum": 10 }
        },
        "required" : ["center", "width"]
    },
    "MOVEBLADE": {
        "description": "Move a single blade.",
        "type": "object",
        "properties": {
            "blade": {"type": "number", "description": "Blade ID", "minimum": 1, "maximum": 2},
            "pos": {"type": "number", "description": "Slit width", "minimum": -120, "maximum": 120}
        },
        "required" : ["blade", "pos"]
    },
    "PARKBLADES": {
        "type": "object",
        "description": "Park both blades.",
        "properties": {}
    }
},
"anyOf": [
    {"required" : ["SETSLIT"]},
    {"required" : ["MOVEBLADE"]},
    {"required" : ["PARKBLADES"]}
]
```

```
{"id": "mos1", "param": {"mos": {"SETSLIT": {"center":2, "width":10}}}}
{"id": "mos1", "param": {"mos": {"MOVEBLADE": {"blade":1, "pos":10}}}}
{"id": "mos1", "param": {"mos": {"PARKBLADES": {}}}}
```

JSON schema

- Documentation and templates shows only “one level” schema
  - bulk of validation is (supposed to be) performed in C++ code;
- “One level” maps to SPF…
  - but SPF will probably be deprecated in IFW5;
- “Multi level” schema allows easy validation of payloads;
  - minor constraints may still be validated in C++ code;
- All JSON schema and payloads may be validated using external tools, e.g.
  - https://jsonlint.com/
  - https://www.jsonschemavalidator.net/

Simulator implementation

- Start from a UML/SysML State Machine model (e.g. MagicDraw);
- Use COMODO (a model-to-text transformation toolkit) to generate the SCXML document “mos.scxml.xml”;
- Modify “mos.namespace.yaml” (generated from IFW template):
  - Add new RPC;
  - Add new PLC attributes;
- Use “coreGenOpcuaProfile” to generate “mos1.namespace.xml”;
- Implement the logic of state transitions (“*f1-ics/fcs/devsim/mos/src/fcs_devsim_mos/mos.py*”);
- Use the latter as simulator configuration file;
- The simulator state machine and RPC methods should be the same as those on the PLC;
- → we can connect to the simulator using, e.g. “OPCUAClient”!!