IWXXM Converter using ecCodes and PyXB : An example of use of ecCodes

20/11/2013
Daniel Dieguez Arias
Meteo-France

14th Workshop on meteorological operational systems
Weather information for aviation is moving to XML format

International Civil Aviation Organization (ICAO) Annex 3
- Amendment 76: Nov 2013 “States in a position to do so should exchange METAR, SIGMET and TAF in a digital form (XML)”
- Amendment 77: Nov 2016 “METAR, SIGMET and TAF should be exchanged in a digital form”

Task team Aviation XML (TT-AvXML) has developed a logical data model named IWXXM (ICAO Meteorological Information Exchange Model)
- Defines XML format for the reports required by ICAO (the equivalent of existing METAR, SIGMET and TAF)
Objective:

- Translate METAR, TAF and SIGMET messages (ascii format) into IWXXM format
- Meteo-France is involved in SESAR activity and format definition due to its operational activity
- ECMWF is involved in format definition and has initiated a software tool named ecCodes
- Therefore Meteo-France and ECMWF have started a collaboration to develop a tool to convert METAR, TAF and SIGMET into IWXXM
Converter architecture overview
First component

- ecCodes:
  - takes as input TAF and METAR messages
  - uses ecCodes TAF and ecCodes METAR rules to interpret the messages
  - generates key/values representation of the messages applying the rules over the messages
- ecCodes rules are written following the regulations (Annex 3)
First component

Output in key/values format

Regulations

Interface (Python, C/C++ ...)

ecCodes

METAR
TAF
Messages

TAF and METAR ecCodes rules are written interpreting the regulations
TAF Regulations

Output in key values format

FM 51-XIV Ext. TAF Aerodrome forecast

CODE FORM:

\[
\begin{align*}
\text{TAF AMD or} & \quad \text{TAF COR or} & \quad \text{TAF} & \quad \text{CCCC} & \quad \text{YYGGgZ} & \quad \text{NIL} & \quad \text{or} \\
& & & & & \quad \text{or} & \quad \text{Y}_1\text{Y}_1\text{G}_1\text{G}_1/\text{Y}_2\text{Y}_2\text{G}_2\text{G}_2 & \quad \text{ddddfGf}_{mf_{m}} & \quad \text{KT} & \quad \text{or} \\
& & & & & & & \quad \text{or} & \quad \text{CNL} & \quad \text{MPS} \\
\end{align*}
\]

\[
\begin{align*}
\text{VVVV} \quad \text{w'w'} & \quad \{ \text{N}_8\text{N}_8\text{h}_8\text{h}_8\text{h}_8 \} & \quad \text{or} & \quad \text{VVVh}_8\text{h}_8\text{h}_8 & \quad \text{or} & \quad \text{NSC} \\
\text{CAVOK} & \quad & \{ \text{YYGG}/\text{Y}_8\text{Y}_8\text{G}_8\text{G}_8 \} & \quad \text{ddddfGf}_{mf_{m}} & \quad \text{KT} & \quad \text{or} \\
& & & & \quad \text{or} & \quad \text{MPS} \\
\text{TXTf}_f\text{Tf}_f/\text{Y}_f\text{Y}_f\text{G}_f\text{G}_f\text{Z} & \quad \text{TNTf}_f\text{Tf}_f/\text{Y}_f\text{Y}_f\text{G}_f\text{G}_f\text{Z} & \quad \text{VVVV} \quad \text{w'w'} & \quad \{ \text{N}_8\text{N}_8\text{h}_8\text{h}_8\text{h}_8 \} & \quad \text{or} & \quad \text{VVVh}_8\text{h}_8\text{h}_8\text{h}_8 & \quad \text{or} \quad \text{NSC} \\
\end{align*}
\]

TAF and METAR ecCodes rules are written interpreting the regulations.
if (substr(g,-2,2) is "KT") {
    alias windPresent=one;
    constant windUnits="knots" :dump;
    rename(g,windInKnots) ;
    modify windInKnots : hidden;
    if (is_integer(windInKnots,3,2)){
        windSpeed=to_string(windInKnots,3,2) :
dump;
    }
    if (substr(windInKnots,0,3) is "VRB") {
        constant windDirection= "Variable" : dump;
        alias windVariableDirection = true;
    }
}
TAF EGPD 300458Z 3006/3106 **25008KT** 9999 FEW045 BECMG 3010/3013 19012KT TEMPO 3013/3019 20015G25KT 6000 RA BKN010

**TAF message**

The diagram illustrates the process of converting METAR and TAF rules into eccodes rules. The regulations are first interfaced with an interface that uses Python, C/C++ language, or similar. This interface generates eccodes rules as input to the METAR and TAF Messages. The diagram emphasizes that TAF and METAR eccodes rules are written by interpreting the regulations.
## ecCodes output

### Output in key/values format

- **dayEndForecastPeriod = 31**
- **hourEndForecastPeriod = 06**
- **windUnits = knots**
- **windSpeed = 08**
- **windDirection = 250**
- **clouds1 = FEW045**
- **prevailingVisibility = 9999**

_TAF and METAR ecCodes rules are written interpreting the regulations_
Second component

- **Python back-end:**
  - takes as input the key/values from ecCodes
  - uses PyXB (Python XML Schema Binding) to have a python representation of the IWXXM model
    - PyXB provides automatically the IWXXM model Python classes
  - encodes TAF and METAR messages in IWXXM format
Second component

- **IWXXM messages**
- **Python back-end**
  - **Python classes**
  - (Key / values) from ecCodes
- **PyXB**
- **IWXXM schemas**

PyXB creates automatically representation of the IWXXM schemas in python
surfaceWind=iwxxm.AerodromeSurfaceWindForecastPropertyType
   (iwxxm.AerodromeSurfaceWindForecastType())

surfaceWind.AerodromeSurfaceWindForecast.meanWindSpeed=
   nsngroup.SpeedType(taf.get_value("windSpeed"))

surfaceWind.AerodromeSurfaceWindForecast.meanWindSpeed.uo
   m=taf.get_value("windUnits")
surfaceWind=iwxxm.AerodromeSurfaceWindForecastPropertyType(iwxxm.AerodromeSurfaceWindForecastType())

surfaceWind.AerodromeSurfaceWindForecast.meanWindSpeed=nsgroup.SpeedType(taf.get_value("windSpeed"))

surfaceWind.AerodromeSurfaceWindForecast.meanWindSpeed.uom=taf.get_value("windUnits")
surfaceWind=iwxxm.AerodromeSurfaceWindForecastPropertyType(
iwxxm.AerodromeSurfaceWindForecastType())

surfaceWind.AerodromeSurfaceWindForecast.meanWindSpeed=_nsgroup.SpeedType(taf.get_value("windSpeed"))

surfaceWind.AerodromeSurfaceWindForecast.meanWindSpeed.uo
m=taf.get_value("windUnits")
IWXXM message
84% (METAR) and 87% (TAF) of messages are successfully converted.

Not converted messages are classified:
- Not well-formed messages:
  - Some messages do not respect the regulations
- Not decoded messages:
  - Bugs in the converter
  - ecCodes rules not yet enough well defined
Conclusions

- Combination of eCodes and PyXB provides a quick way to implement a converter from TAF, METAR to IWXXM
- METAR and TAF decoder can only be written interpreting regulations. Long implementation time is needed although ecCodes rules language makes the process quicker
- Use of IWXXM model in python classes requires a good knowledge of the model
- More work is still necessary to produce a full operational converter
- The tool will be used in SESAR project
- We are open for collaboration
Any Questions?

Please, provide your feedback and comments to daniel.dieguez-arias@meteo.fr

Thank you