Dealing with error in applications

Author: Wiljan Derks

Why we do this

- Application should adhere to one principle of dealing with errors. This makes understanding of the software easier.
- We want to be able to collect errors in an integral way:
  - Dealing with error text
  - Dealing with error description
  - Logging of errors
  - Documentation of errors

Representing errors

Error is 32 bit integer containing:

- Severity
  - Oke: Success, Informational
  - Not Oke: Warning, Error, Severe
- Facility
  - Unique number of this error range
- Error number
  - Unique number of the error with the facility

Repesentation copied from VMS

How to use inside code (1)

Routine return error code

```
procedure dosomething (……  Error : out Integer);
  begin
      ...
      Error := Mot_Useerror;
      ...
  end
```

After calling always check the error

```
dosomething (Error);
if Status_Oke (Error) then
  ...
```

How to use inside code (2)

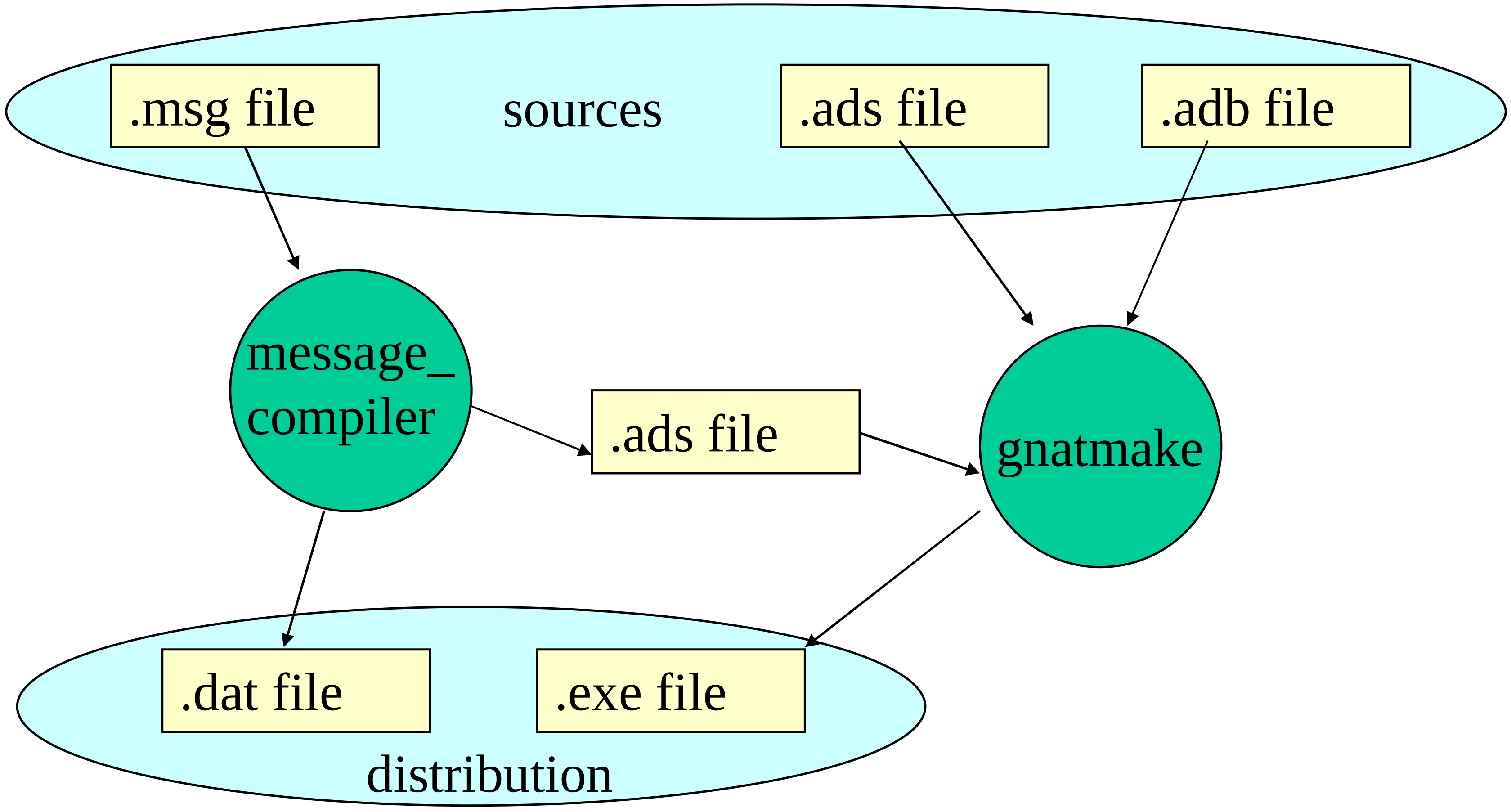
Routine raises exception with that error code

```
procedure dosomething (…… );
  begin
      ...
      Raise_Exception (Mot_Useerror);
      ...
  end
```

To catch error use exception handler:

```
begin
    dosomething (…);
exception when E : others =>
    --  exception handling code
end;
```

Building cycle



Contents of .msg file

```
.facility Mot,3
.severity error
!----------------------------------------------------
! The following errors are software errors
!----------------------------------------------------
UseError            <Wrong use, calling routine when not allowed>
! Some internal error detected. Report to ITEC.
MotorNotReady       <Motor not ready>
CommunicationError <Communication error>
EncoderDisconnect  <Encoder disconnect>
! The encoder seems no longer to be connected to the motion
! Controller. Please check the cabling of it.
.end
```

Facility name

Brief error description

Error name

Full error description

Support packages

```
Package Eln
        Success : constant Integer := 1;  --  Success status
        function Status_Oke (Status : Integer) return Boolean;
Package Eln.Exceptions
        function Exception_To_Status (X : Exception_Occurrence) return Integer;
        procedure Raise_Exception (Status : Integer);
        function Status_To_Text (Status : Integer) return String;
        function Status_To_Description (Status : Integer) return String;
```

Integration into Visual ITEC

Dealing of status codes within Visual ITEC

- When visual_itec knows something is a status, it will display the proper text (Status_To_Text result) and have a proper tooltip (Status_To_Description).

How to let Visual ITEC know:

- Returning a status as item text: use make_status
- When exec_command or modify_item calls fail: return proper status

Error logging

We call is ESM: Equipment status monitoring.

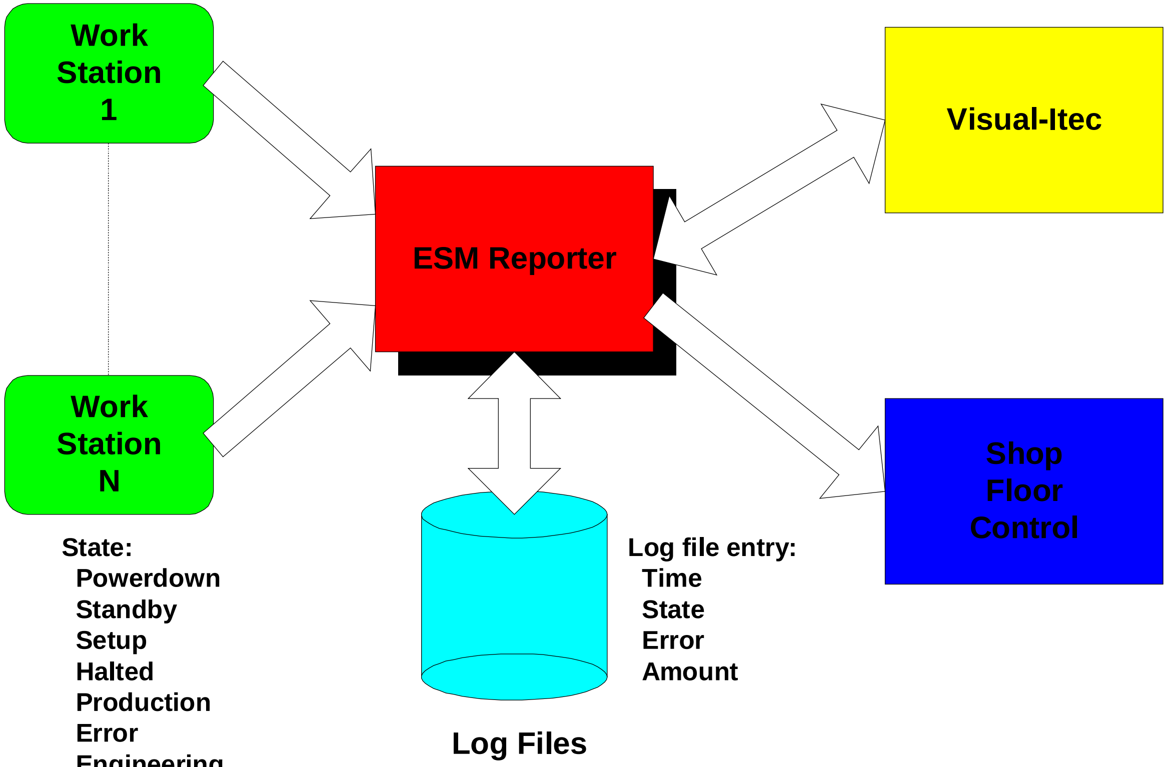
We log:

- Time
- Errors
- State changes

When error log component build in the application it adds the following functionality:

- – Life pareto of errors that have occurred
- – Percentage of time spend in states
- – Command to reset the .esm file (clears the history)

ESM software component



State:
- Powerdown
- Standby
- Setup
- Halted
- Production
- Error
- Engineering

Log file entry:
- Time
- State
- Error
- Amount

Conclusions

Error mechanism used in all our applications

Results:

- **Easy and simple way to deal with errors**
- **Error unique over all applications**
- **Also solves documentation of errors:**
  - **Tooltips show descriptions**
  - **Possible to build list of errors + descriptions automatic**
- **Allows logging of errors in a integral and compact way**