ABL Logging

From Files to the Cloud, with No Code Changes

Peter Judge
pjudge@progress.com

What is logging?

Using the ABL logging component

Implementation internals

Customising loggers

Why do we need logging?

1. We messed up
   - Maybe someone we depend on messed up
   - Maybe someone who knows us, but is far, far away messed up

2. We need to know when things happened
   - Verification of software operation, business processes, etc
   - See 1 above

Logging captures and records these things

Logging requirements

- Sequential set of events
  - Typically ordered by time
  - May need to be immutable
- Log data tends to be shared, so a common format, like ASCII text, is advised
- Events are
  - Issued by applications/code
  - Recorded by another (sub)system / component
- Events include one or more pieces of
  - Data PUG Challenge Africa started
  - Metadata INFO 2019-03-06T09:00:00.000+02:00 pjudge

In the beginning

… there was PUT and MESSAGE

Then came LOG-MANAGER in 10.0A

LOG-MANAGER is pretty decent

- The `LOG-MANAGER` is a built-in logging framework that's used by AppServers
- System / product-level stuff

`LOG-ENTRY-TYPES`	QryInfo, 4GLTrace, DynObjects.*
`LOGGING-LEVEL`	of various potencies 1..*n*
`NUM-LOG-FILES, LOG-THRESHOLD`	control log file rollover

- Applications can write to the log using ABL

```
WRITE-MESSAGE('Log message','Log group')
```

- `LOG-MANAGER` is a one-size-fits-all log (single file, file-only)
  - You can't depend on the log always *always* being there
  - You can't control the format of the output

https://documentation.progress.com/output/ua/OpenEdge_latest/dvdbg/logging-in-openedge.html#

Common logging infrastructure features

- Logging config should not be the responsibility of the application developer
- Sysadmins should be able to "twiddle the knobs" in production / deployed environments
  - What kinds of messages are logged:
    errors, warnings, info etc
  - Where the messages go:
    files, /dev/null, db tables, ElasticStashKibanaSearch
- Writing log messages should be simple for devs
  - No `IF LOG-MANAGER:LOGFILE <> ? THEN WRITE-MESSAGE()`
- Low to no performance impact

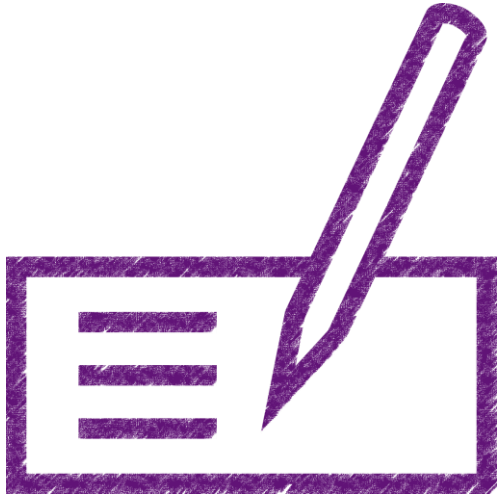
If not LOG-MANAGER then what?

- We needed the ability to separate the writing of log messages from the configuration of the outputs (and formats)
- After some research, we found SLF4J, a well-regarded and popular Java logging framework

> The Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks (e.g. java.util.logging, logback, log4j) allowing the end user to plug in the desired logging framework at *deployment* time. https://www.slf4j.org/

- We created a set of classes in the OpenEdge.Logging namespace that are based on the concepts behind SLF4J
  - Available from 11.7.0 onwards in the `OpenEdge.Core.pl`

Using the logging component



Using the OpenEdge.Logging component

```
using OpenEdge.Logging.*.

define variable Logger as ILogWriter no-undo.

// Get a reference to a logger
assign Logger = LoggerBuilder:GetLogger('Demo.PUGSA').

// now we write our messages ...
Logger:Info('PUGSA started').

Logger:Warn(substitute('General session speaker &2 started at &1',
                       iso-date(now), get-db-client():user-id).

// yak yak yak

Logger:Trace('General session end').
```

Using the OpenEdge.Logging component

```
using OpenEdge.Logging.*.

define variable Logger as ILogWriter no-undo.

// Get a reference to a logger
assign Logger = LoggerBuilder:GetLogger('Demo.PUGSA').

// now we write our messages ...
Logger:Info('PUGSA started').

Logger:Warn(substitute('General session speaker [illegible] started at &1',
                       iso-date(now), get-db-client([illegible]ser

// yak yak yak

Logger:Trace('General session end').
```

Interface-based so you can use any implementation

Always returns a valid reference, even when there's no logger defined / set up

Using the OpenEdge.Logging component

```
using OpenEdge.Logging.*.

define variable Logger as ILogWriter no-undo.

// Get a reference to a logger
assign Logger = LoggerBuilder:GetLogger('Demo.PUGSA').

// now we write our messages ...
Logger:Info('PUGSA started').

Logger:Warn(substitute('General session speaker &2 started at &1',
                       iso-date(now), get-db-client():user-id).

// yak yak yak

Logger:Trace('General session end').
```

Write message data only

Using the OpenEdge.Logging component

```
using OpenEdge.Logging.*.

define variable Logger as ILogWriter no-undo.

// Get a reference to a logger
assign Logger = LoggerBuilder:GetLogger('Demo.PUGSA').

// now we write our messages ...
Logger:Info('PUGSA started').

Logger:Warn(substitute('General session speaker &2 started at &1',
                       iso-date(now), get-db-client():user-id).

// yak yak yak

Logger:Trace('General session end').
```

Method name indicates type of message

Application Developer's API

- `OpenEdge.Logging.ILogWriter` is the logger itself
  - Consumes `OpenEdge.Logging.LogMessage` via named method
  - Has a name
  - Specifies a logging level
- `OpenEdge.Logging.ISupportLogging` interface signals that a class supports logging via a Logger property (optional)

```
interface OpenEdge.Logging.ISupportLogging:
    // A reference to the Logger in use by an implementer
    define public property Logger as OpenEdge.Logging.ILogWriter no-undo get. set.

end interface.
```

Application Developer API: ILogWriter

```
interface OpenEdge.Logging.ILogWriter:
  // (mandatory) Name for this logger
  define public property Name as character no-undo get.
  // (mandatory) The level being logged at
  define public property LogLevel as using OpenEdge.Logging.LogLevelEnum no-undo get.
  // Event methods for INFO messages
  method public void Info(input pcMessage as character).
  method public void Info(input pcMessage as character,  input poError as Progress.Lang.Error).

  method public void Info(input poMessage as OpenEdge.Logging.LogMessage).
  method public void Info(input poMessage as OpenEdge.Logging.LogMessage,
                          input poError as Progress.Lang.Error).

  method public void Info(input pcMessageGroup as character, input pcMessage as character).
  method public void Info(input pcMessageGroup as character,
                          input pcMessage as character,
                          input poError as Progress.Lang.Error).
  // Event methods for ERROR, WARN, FATAL, DEBUG, TRACE messages all have the same signatures as above
```

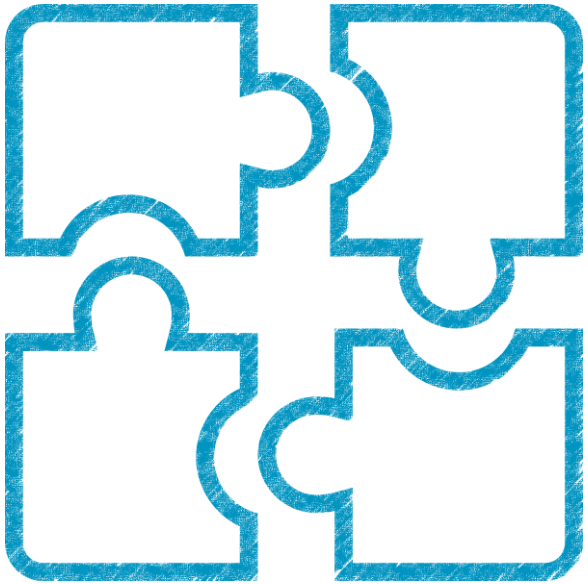
Application Developer API: LogMessage

```
class OpenEdge.Logging.LogMessage serializable:
  // (mandatory) The group for this log message
  define public property GroupName as character no-undo get. private set.
  // (mandatory) The base text of the message. May contain substitution
  // parameters like &1 or {}
  define public property BaseText as character no-undo get. private set.
  // (mutable) The formatted message for writing to the logger target
  define public property Message as character no-undo get. set.

  // Returns a context value for a given key
  method public Progress.Lang.Object GetContext(input pKey as character):
  // Adds context values to this message
  method public void AddContext(input pKey as character,
                                input pContext as Progress.Lang.Object):
```

That's it! For the application dev

Progress®



Implementation internals

Getting a logger instance

- The LoggerBuilder
- ConfigFileLoggerBuilder

The LogFilter chain

- LogEvent
- Filter implementations
- Tokens

Default / shipped implementations

1. The VOID / sink logger
   - Does nothing except exist
   - Default / fall-back logger (if we can't otherwise find or build a logger)

```
OpenEdge.Logging.VoidLogger
```

2. The filter-based logger
   - Passes an *event* down a chain (linked-list) of one or more filters
   - Filters format (transform) and write log messages

```
OpenEdge.Logging.Logger
OpenEdge.Logging.ILoggerFilter
OpenEdge.Logging.LogEvent
```

Getting a logger

```
// Get a reference to a logger
assign Logger = LoggerBuilder:GetLogger('Demo.PUGSA.Talks').
```

Use `LoggerBuilder:GetLogger(<logger-name>)` factory method to get a logger

<logger-name> can be anything, by convention follows the "Named Hierarchy" pattern used by log4j and other logging frameworks; this dotted-name pattern works very will with OOABL type names since there's a natural hierarchy in class and package names; for example

```
OpenEdge.Net.ServerConnection.ClientSocket
```

The following algorithm is used to determine which logger configuration to use

1. Find an exact match to *Logger-name* .
2. If not found, if the *Logger-name* has at least one dot, chop off the last (right-most) dot-delimited entry and repeat step 1
   a) Stop if a match is found or there are no more dot-delimited entries
3. If not found, repeat step 1 with the value of `LoggerBuilder:DefaultLogger` if set
4. If no logger is found at this point,
   - If the `LOG-MANAGER` is active, then build a logger based on it
   - If not, use the `OpenEdge.Logging.VoidLogger`

```
❑ Demo.PUGSA.Talks
✓ Demo.PUGSA
  Demo
  <default>
```

Default/standard logger builder

- The LoggerBuilder follows the abstract factory pattern; the default (only) builder is the `OpenEdge.Logging.ConfigFileLoggerBuilder`
- This builds loggers from configurations in a specific JSON file called `logging.config`
- The config file is checked for changes when a logger is requested, and reloaded if needed

```
//logging.config
{
  "DEFAULT_LOGGER": "OpenEdge",
  "logger": {
    "OpenEdge": {
      "logLevel": "ERROR",
      "filters": [
        "ERROR_FORMAT",
        "BACK_WORDS_FORMAT",
        "FULL_TEXT_FORMAT",
        {
          "name": "NAMED_FILE_WRITER",
          "fileName": "${session.temp-dir}/one.log",
          "appendTo": true
        }
      ]
    }
  },
  "filter": {
    "BACK_WORDS_FILTER":
                        "Example.Filters.ReverseWordsFormat"
  }
}
```

Logger Builder – in code

```
logger = LoggerBuilder
         :Build('com.data.service')
         // logging level
         :LogAt(LogLevelEnum:DEBUG)

         // formatting filters
         :AddFilter(LoggerFilterRegistry:ABL_SUBSTITUTE_FORMAT)
         :AddFilter(LoggerFilterRegistry:ERROR_FORMAT)
         :AddFilter(LoggerFilterRegistry:FULL_TEXT_FORMAT)
         // writer filter
         :AddFilter(LoggerFilterRegistry:NAMED_FILE_WRITER)

         // gimme the logger
         :Logger.
```

Filters

A log event is passed into a set of filters, in order

```
interface OpenEdge.Logging.Filter.ILoggerFilter:

    /** Performs implementation-specific filtering for a logger type

        @param LogEvent The log event to filter. */
    method public void ExecuteFilter(input poEvent as LogEvent).

end interface.
```

```
OpenEdge.Logging.Format.

   ABLSubstituteFormat
   AnonymizedTokenFormat
   ErrorFormat
   FullTextFormat
   LogManagerFormat
   MDCTokenFormat
   ResolvedTokenFormat
   StackWriterFormat
   TokenContextFormat
```

```
OpenEdge.Logging.Writer.

   LogManagerWriter
   MessageStatementWriter
   NamedFileWriter
   VoidWriter
```

Log event

1. What level was the message logged at?
2. Who logged the message?
3. When was it logged?
4. Where was it logged?
5. Is there an associated error?

```
class OpenEdge.Logging.LogEvent serializable:
  // The logger that initiated this event
  define public property Logger as ILogWriter no-undo get. set.
  // The name of the logger
  define public property LoggerName as character no-undo get. set.
  // The level of this event
1 define public property LogLevel as LogLevelEnum no-undo get.
  // The more-or-less exact time when the log event occurred
3 define public property TimeStamp as datetime-tz no-undo get.
  // The log message
  define public property Message as LogMessage no-undo get.
  // An error to log
5 define public property Error as Progress.Lang.Error no-undo get.
  // The current stack trace, of where the LOG event occurred.
4 define public property CallStack as character extent no-undo get.
  // The user logging this event
2 define public property LoggedBy as handle no-undo get. set.
  // The short-name of the logger logging this event.
  define public property LoggerShortName as character no-undo get. set.
  // The short-name-format of the logger logging this event
  define public property ShortNameFormat as character no-undo get. set.
```

Formatting filter: FullTextFormat

```
class OpenEdge.Logging.Format.FullTextFormat implements ILoggerFilter, ISupportFormatting:
  /* Format for the logger name. See the TokenResolve class for more */
  define public property Format as character initial '1C':u no-undo get. set.

  method public void ExecuteFilter( input poEvent as LogEvent ):
    define variable messageGroup as character no-undo.
    if poEvent:Message:GroupName eq '':u then
    do:
      if this-object:Format eq poEvent:ShortNameFormat then
        assign messageGroup = poEvent:LoggerShortName.
      else
        assign messageGroup = TokenResolver:ResolveName(this-object:Format, poEvent:LoggerName).
    end.
    else
      assign messageGroup = poEvent:Message:GroupName.

    assign poEvent:Message:Message = substitute('[&1] &3 &2: &4':U,
                              /*1*/ iso-date(poEvent:TimeStamp),
                              /*2*/ string(poEvent:LogLevel),
                              /*3*/ messageGroup,
                              /*4*/ poEvent:Message:Message).
  end method.
end class.
```

Writing filter: NamedFileWriter

```
method public void ExecuteFilter(input poEvent as LogEvent):
  define variable mData as memptr no-undo.
  define variable msgLen as integer no-undo.

  // We use a MEMPTR to preserve trailing blanks etc
  assign msgLen = length(poEvent:Message:Message, 'raw':u) + 1.
  set-size(mData) = msgLen.
  put-string(mData, 1, msgLen) = poEvent:Message:Message + StringConstant:LF.

  output stream sFileOutput to value(moFileOutputStream:FileName) append.
    export stream sFileOutput mData.
  output stream sFileOutput close.

  finally:
    set-size(mData) = 0.
  end finally.
end method.
```

Internals

Progress®

Customising loggers

Tokens

Write your own filter

Update logging.config

Token-based formatters

Tokens are variables in the log message (and log file names)

```
${<token>}
```

where

```
token = group ["." arg]
group = "session" / "env" / "guid" / "t[ime]"
        / "web" / "ver[sion] " / "cp"
        / "req[uest]" / "name" / "err"
        / "msg" / "mdc"
```

The supported `arg` values depend on the group

Tokens in messages

```
using OpenEdge.Logging.*.

define variable Logger as ILogWriter no-undo.

// Get a reference to a logger
assign Logger = LoggerBuilder:GetLogger('Demo.PUGSA').

// now we write our messages ...
Logger:Info('PUGSA started').

Logger:Warn(substitute('General session speaker &2 started at &1',
                       iso-date(now), get-db-client():user-id).

Logger:Warn('General session speaker ${cp.uid} started at ${t.now}').

// yak yak yak

Logger:Trace('General session end').
```

Tokens in config

```
1.  {
2.    "DEFAULT_LOGGER": "Example.Server.Request",
3.    "logger": {
4.      "Example.Server.Request": {
5.        "logLevel": "ERROR",
6.        "filters": [
7.          {
8.            "name": "TOKEN_FORMAT",
9.            "format": "[${t.iso} ${msg.level} ${req.tpt}] ${msg} logged by ${cp.quid}"
10.         },
11.         {
12.           "name": "NAMED_FILE_WRITER",
13.           "fileName": "${session.temp-dir}/server-${t.today}-${req.id}.log",
14.           "appendTo": true
15.         }
16.       ]
17.     }
18.   }
19. }
```

Format filter: LogManagerFormat

```
1.  method public void ExecuteFilter( input poEvent as LogEvent):
2.          define variable loggerShortName as character no-undo.
3.          // try to avoid resolving the token on each message
4.          if this-object:Format eq poEvent:ShortNameFormat then
5.              assign loggerShortName = poEvent:LoggerShortName.
6.          else
7.              assign loggerShortName = TokenResolver:ResolveName(this-object:Format,
8.                                                                 poEvent:LoggerName).
9.
10.         assign poEvent:Message:Message = substitute('[&3 &1] &2':u,
11.                                          /*1*/ string(poEvent:LogLevel),
12.                                          /*2*/ poEvent:Message:Message,
13.                                          /*3*/ loggerShortName        ).
14.     end method.
```

```
[17/04/12@12:40:26.684-0400] P-022384 T-004468 1 4GL LogMgrWrtr [l.C.U.i.L.NewName INFO]  A new dawn
```

Mapped Diagnostic Context (MDC)

"Mapped Diagnostic Context" is essentially a map maintained by the logging framework where the application code provides key-value pairs which can then be inserted by the logging framework in log messages

https://www.slf4j.org/manual.html

1. Add `${mdc.<mdc-key>}` tokens to messages
2. Set values for MDC keys in code

```
// Set MDC key values somewhere in the application code
OpenEdge.Logging.MDC:Put('myName', 'Michael Caine').
OpenEdge.Logging.MDC:Put('whoAreYou', ?).
// Write log message with MDC tokens
logger:Info('A message with some tokens ${t.now} or even ${mdc.myName} ').



A message with some tokens 2019-02-27T17:00:04.941-05:00 or even Michael Caine
```

Data anonymisation

- GDPR et al don't like private data being made public
- The `ANON_FORMAT` filter one-way hashes ***token values*** in messages
- The anonymised output follows the C crypt format as per https://en.wikipedia.org/wiki/Crypt_(C)

`$<id>$<salt>$<b64-hash>`

The `<id>` value represents the hashing algorithm, and is one of MD5, SHA-1, **SHA-256**, SHA-512

The `<salt>` value is a base64-encoded UUID generated by the AVM

- Default tokens to anonymise are `CP.UID CP.QUID`

Data anonymisation

```
1. { "logger": {
2.     "Example.Custom": {
3.       "filters": [
4.         { "name":          "ANON_FORMAT",
5.           "tokensToAnon": "mdc.myName",
6.           "hashAlgo":     "SHA-512"
7.         }
8.       ]
9.     }
10. }
```

```
logger:Info('A message with some tokens ${t.now} or even ${mdc.myName} ').

A message with some tokens 2019-02-27T17:00:04.941-05:00 or even
$6$jO4MxS3PE5NOFI0VBIW/Tg$8l8LeypBlbzBhbER+I+UC6dyD7wwZRpmcskR/UW/Q1a5675FM3htnhVT5eeB3uFmutURB
i+0siOjjAjuP7rFhA==
```

Adding your own filters

```
1. { "logger": {
2.   "Example.Custom": {
3.     "filters": [
4.       { "name": "TRANSLATION_FILTER",
5.         "toLang": "af",
6.         "serviceURI": "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0",
7.       },
8.       "REVERSED_WORDS_FILTER",
9.       { "name": "ELASTIC_SEARCH_WRITER",
10.        "serviceURI": http://localhost:9200/
11.      }
12.] } },
13.  "filter": {
14.    "REVERSED_WORDS_FILTER": "Example.Filters.ReverseWordsFormat",
15.    "TRANSLATION_FILTER": {
16.      "type":    "Example.Filters.TranslatedMessageFormat",
17.      "builder": "Example.Builders.TranslatedMessageFormatBuilder"
18.    },
19.    "ELASTIC_SEARCH_WRITER": {
20.      "type":    "Example.Filters.ElasticSearchWriter",
21.      "builder": "Example.Builders.ElasticSearchWriterBuilder"
22.} } }
```

Adding your own filters

```
1. { "logger": {
2.   "Example.Custom": {
3.     "filters": [
4.       { "name": "TRANSLATION_FILTER",
5.         "toLang": "af",
6.         "serviceURI": "https://api.cognitive.microsofttra
7.       },
8.       "REVERSED_WORDS_FILTER",
9.       { "name": "ELASTIC_SEARCH_WRITER",
10.        "serviceURI": http://localhost:9200/
11.      }
12.] } },
13.  "filter": {
14.    "REVERSED_WORDS_FILTER": "Example.Filters.ReverseWordsFormat",
15.    "TRANSLATION_FILTER": {
16.      "type":    "Example.Filters.TranslatedMessageFormat",
17.      "builder": "Example.Builders.TranslatedMessageFormatBuilder"
18.    },
19.    "ELASTIC_SEARCH_WRITER": {
20.      "type":    "Example.Filters.ElasticSearchWriter",
21.      "builder": "Example.Builders.ElasticSearchWriterBuilder"
22.} } }
```

1. Write new filters
2. Write new filter builders
3. Add to new code to logging config's `filter` property

Adding your own filters

```
1. { "logger": {
2.   "Example.Custom": {
3.     "filters": [
4.       { "name": "TRANSLATION_FILTER",
5.         "toLang": "af",
6.         "serviceURI": "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0",
7.       },
8.       "REVERSED_WORDS_FILTER",
9.       { "name": "ELASTIC_SEARCH_WRITER",
10.        "serviceURI": http://localhost:9200/
11.      }
12.] } },
13.  "filter": {
14.    "REVERSED_WORDS_FILTER": "Example.Filters.ReverseWordsFormat",
15.    "TRANSLATION_FILTER": {
16.      "type":    "Example.Filters.TranslatedMessageFormat",
17.      "builder": "Example.Builders.TranslatedMessageFormatBuilder"
18.    },
19.    "ELASTIC_SEARCH_WRITER": {
20.      "type":    "Example.Filters.ElasticSearchWriter",
21.      "builder": "Example.Builders.ElasticSearchWriterBuilder"
22.} } }
```

Adding your own filters

```
1. { "logger": {
2.   "Example.Custom": {
3.     "filters": [
4.       { "name": "TRANSLATION_FILTER",
5.         "toLang": "af",
6.         "serviceURI": "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0",
7.       },
8.       "REVERSED_WORDS_FILTER",
9.       { "name": "ELASTIC_SEARCH_WRITER",
10.        "serviceURI": http://localhost:9200/
11.      }
12. ] } },
13.   "filter": {
14.     "REVERSED_WORDS_FILTER": "Example.Filters.ReverseWordsFormat",
15.     "TRANSLATION_FILTER": {
16.       "type":    "Example.Filters.TranslatedMessageFormat",
17.       "builder": "Example.Builders.TranslatedMessageFormatBuilder"
18.     },
19.     "ELASTIC_SEARCH_WRITER": {
20.       "type":    "Example.Filters.ElasticSearchWriter",
21.       "builder": "Example.Builders.ElasticSearchWriterBuilder"
22.} } }
```

Extending & enhancing

- Word reverser
  - Plain ABL
- Translate messages
  - Call to Azure translation
- Cloud enable
  - Send log events to an Elasticsearch instance
  - Visualise log messages in Kibana

Troubleshooting

- Generally, loggers should never throw errors. Ever.

  … which makes finding out what failed hard

- If the logger builder throws an error, the void logger is used
  - The errors are written to either the `LOG-MANAGER` or Agent log or `SESSION:TEMP-DIR/loggerbuilder.log`

- If any one filter throws an error, the filter chain breaks

  (we should fix this so that only that filter is affected)

Fin

- Modern logging separates writing and recording of messages
  - Separation of concerns (between admins & devs)
  - No need to change application code to record more or less log data

- The OpenEdge.Logging component helps with
  - Sane defaults
  - Simple extension and customisation

Questions?

pjudge@progress.com