

■ ABL Logging: From files to the Cloud

A new and improved way of logging

Consultingwerk
software architecture and development



Peter Judge

- Senior Architect at Consultingwerk
- Writing 4GL since 1996, working on a variety of frameworks and applications. More recently have worked on a lot of integration-y stuff: Authentication Gateway, HTTP Client, Web Handlers. Dabble in PASOE migrations.
- Active participator in Progress communities, PUGs and other events



Consultingwerk Software Services Ltd.

- Independent IT consulting organization
- Focusing on **OpenEdge** and **related technology**
- Located in Cologne, Germany, subsidiaries in UK, USA and Romania
- Customers in Europe, North America, Australia and South Africa
- Vendor of developer tools and consulting services
- Specialized in GUI for .NET, Angular, OO, Software Architecture, Application Integration
- Experts in OpenEdge Application Modernization



Services Portfolio, Progress Software

- OpenEdge (ABL, Developer Tools, Database, PASOE, ...)
- Telerik DevCraft (.NET, Kendo UI, Angular, ...), Telerik Reporting
- OpenEdge UltraControls (Infragistics .NET)
- Telerik Sitefinity CMS (incl. integration with OpenEdge applications)
- Kinvey Plattform, NativeScript
- Corticon BRMS
- Whatsup Gold infrastructure-, network- and application monitoring
- Kemp Loadmaster
- ...

Services Portfolio, related products

- Protop Database Monitoring
- Combit List & Label
- Web frameworks, e.g. Angular
- .NET
- Java
- ElasticSearch, Lucene
- Amazon AWS, Azure
- DevOps, Docker, Jenkins, ANT, Gradle, JIRA, ...
- ...

Agenda

- What is logging?
- Using the ABL logging component
- Implementation details
- Customising & extending loggers





Why logging?

Why do we need logging?

1. We need to know when things happened
 - Verification of software operation, traceable technical or business processes, etc.
2. We messed up
 - Maybe someone we depend on messed up

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 US PUG started
 - Metadata INFO 2023-11-13T09:00:00.000+01:00 daniel.van.doorn

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' , 'MYGROUP')
- 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
- OPS / DevOps 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

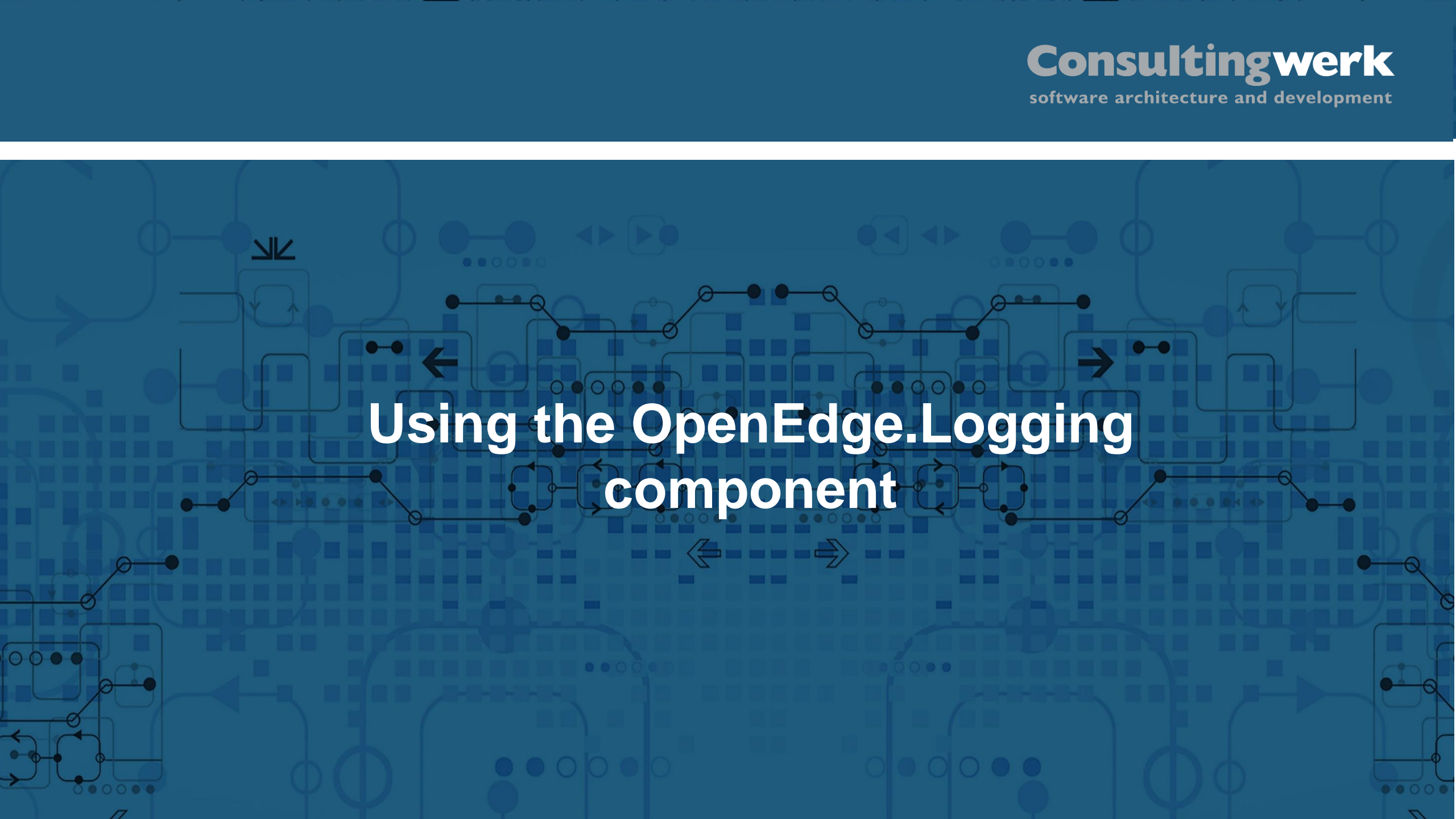
If not LOG-MANAGER then what?

- In OE 11.7.0, Progress released a logging component, as a set of ABL classes in the OpenEdge.Logging namespace
 - Shipped in OpenEdge.Core.pl ; always on PROPATH
- Modelled on 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/>



Using the OpenEdge.Logging component

Using an OpenEdge.Logging logger

```
1. using OpenEdge.Logging.*.  
2.  
3. define variable Logger as ILogWriter no-undo.  
4.  
5. // Get a reference to a logger  
6. assign Logger = LoggerBuilder:GetLogger("Demo.EMEAPUG.NL").  
  
7. // now we write our messages ...  
8. Logger:Info("EMEA PUG started").  
9.  
10. Logger:Warn(substitute("Webinar speaker &2 started at &1",  
11.                       iso-date(now), get-db-client():user-id)).  
12.  
13. // yak yak yak  
14.  
15. Logger:Trace("Webinar session end").
```

Using an OpenEdge.Logging logger

```
1. using OpenEdge.Logging.*.  
2.  
3. define variable Logger as ILogWriter no-undo.  
4.  
5. // Get a reference to a logger  
6. assign Logger = LoggerBuilder:GetLogger(" Demo.EMEAPUG.NL ").  
7.  
8. // now we write our messages ...  
9. Logger:Info("EMEA PUG started").  
10.  
11. Logger:Warn(substitute("Webinar speaker &2 started at &1",  
12.                          iso-date(now), get-db-client():user-id).  
13.  
14. // yak yak yak  
15.  
16. Logger:Trace("Webinar 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 an OpenEdge.Logging logger

```
1. using OpenEdge.Logging.*.  
2.  
3. define variable Logger as ILogWriter no-undo.  
4.  
5. // Get a reference to a logger  
6. assign Logger = LoggerBuilder:GetLogger("Demo.EMEAPUG.NL ").  
7.  
8. // now we write our messages ...  
9. Logger:Info("EMEA PUG started").  
10.  
11. Logger:Warn(substitute("Webinar speaker &2 started at &1",  
12.                        iso-date(now), get-db-client():user-id)).  
13.  
14. // yak yak yak  
15.  
16. Logger:Trace("Webinar session end").
```

Write message
data only

Using an OpenEdge.Logging logger

```
1. using OpenEdge.Logging.*.  
2.  
3. define variable Logger as ILogWriter no-undo.  
4.  
5. // Get a reference to a logger  
6. assign Logger = LoggerBuilder:GetLogger("Demo.PUG.UK").  
7.  
8. // now we write our messages ...  
9. Logger:Info("UK PUG started").  
10.  
11. Logger:Warn(substitute("webinar speaker &2 started at &1",  
12. iso-date(now) get-db-client():user-id).  
13.  
14. // yak yak yak  
15.  
16. Logger:Trace("Webinar 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

```
1. interface OpenEdge.Logging.ILogWriter:
2.     // (mandatory) Name for this logger
3.     define public property Name as character no-undo get.
4.     // (mandatory) The level being logged at
5.     define public property LogLevel as using OpenEdge.Logging.LogLevelEnum no-undo get.
6.     // Event methods for INFO messages
7.     method public void Info(input pcMessage as character).
8.     method public void Info(input pcMessage as character,
9.                             input poError as Progress.Lang.Error).
10.
11.    method public void Info(input poMessage as OpenEdge.Logging.LogMessage).
12.    method public void Info(input poMessage as OpenEdge.Logging.LogMessage,
13.                            input poError as Progress.Lang.Error).
14.
15.    method public void Info(input pcMessageGroup as character, input pcMessage as character).
16.    method public void Info(input pcMessageGroup as character,
17.                            input pcMessage as character,
18.                            input poError as Progress.Lang.Error).
19.
20. // Event methods for ERROR, WARN, FATAL, DEBUG, TRACE messages all have the same
21. // signatures as above
```

Application Developer API: LogMessage

```
1. class OpenEdge.Logging.LogMessage serializable:
2.     /* (mandatory) The group for this log message */
3.     define public property GroupName as character no-undo get. private set.
4.
5.     /* (mandatory) The base text of the message. May contain substitution parameters
6.        like &1 or {} */
7.     define public property BaseText as character no-undo get. private set.
8.
9.     /* (mutable) The formatted message for writing to the logger target */
10.    define public property Message as character no-undo get. set.
11.
12.    /* A set of tokens in the message text. Populated from the constructor.
13.       A set to avoid any duplication. */
14.    define public property Tokens as Progress.Collections.ISet<Token> no-undo
15.        get.
16.        private set.
```

Getting a logger


```
/* Get a reference to a logger */
assign Logger = LoggerBuilder:GetLogger("Demo.PUG.UK").
```

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 well 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.PUG.UK.Talks
 Demo.PUG.UK
Demo
<default>
```




Implementation details

LoggerBuilder:GetLogger(<logger-name>)

- `OpenEdge.Logging.LoggerBuilder:GetLogger(Logger-name)` returns a logger instance
 - *Logger-name* can be a string, handle or `Progress.Lang.Class`
- 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
- Logger instances are cached by the `LoggerBuilder`

```
// 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"
  }
}
```

Build a logger in code

```
1.  logger = LoggerBuilder:Build('com.data.service')
2.      // logging level
3.      :LogAt(LogLevelEnum:DEBUG)
4.
5.      // formatting filters
6.      :AddFilter(LoggerFilterRegistry:ABL_SUBSTITUTE_FORMAT)
7.      :AddFilter(LoggerFilterRegistry:ERROR_FORMAT)
8.      :AddFilter(LoggerFilterRegistry:LOG_MANAGER_FORMAT)
9.
10.     // writer filter
11.     :AddFilter(LoggerFilterRegistry:LOG_MANAGER_WRITER)
12.
13.     // gimme the logger
14.     :Logger.
```

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)

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

```
OpenEdge.Logging.VoidLogger
```

```
OpenEdge.Logging.Logger
```

```
OpenEdge.Logging.ILoggerFilter
```

```
OpenEdge.Logging.LogEvent
```


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.
```

1

```
// The level of this event  
define public property LogLevel as LogLevelEnum no-undo get.
```

3

```
// The more-or-less exact time when the log event occurred  
define public property TimeStamp as datetime-tz no-undo get.
```

★

```
// The log message  
define public property Message as LogMessage no-undo get.
```

5

```
// An error to log  
define public property Error as Progress.Lang.Error no-undo get.
```

4

```
// The current stack trace, of where the LOG event occurred.  
define public property CallStack as character extent no-undo get.
```

2

```
// The user logging this event  
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.
```

Filters

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

```
interface OpenEdge.Logging.Filter.ILoggerFilter:  
  
    /** Performs implementation-specific filtering for a  
        logger type  
        @param LogEvent The log event to filter. */  
    method public void ExecuteFilter(poEvent as LogEvent).  
  
end interface.
```

OpenEdge.Logging.Format.

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

OpenEdge.Logging.Writer.

- JsonLogWriter
- LogManagerWriter
- MessageStatementWriter
- NamedFileWriter
- VoidWriter

Formatting filter: FullTextFormat

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

    method public void ExecuteFilter( input poEvent as LogEvent ):
        define variable messageGroup as character no-undo.
        // Avoid recalculating this on each call
        if this-object:Format eq poEvent:ShortNameFormat or this-object:Format eq '' then
            assign messageGroup = poEvent:LoggerShortName.
        else
            assign messageGroup = TokenResolver:ResolveName(this-object:Format, poEvent:LoggerName).

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

    end method.
end class.
```

Writing filter: NamedFileWriter

```
method public void ExecuteFilter(input poEvent as LogEvent):  
    if not mLogFile:FolderExists and not mLogFile:CreateFolder() then  
        undo, throw new AppError(substitute('Unable to create folder &1', mLogFile:Folder), 0).  
    // can we write to the file?  
    if not mLogFile:CanWriteToFile() then  
        undo, throw new AppError(substitute('Unable to write to log &1', mLogFile:Name), 0).  
  
    // We use a MEMPTR to preserve trailing blanks etc, that are removed by the PUT UNFORMATTED  
    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(this-object:FileName) append.  
        export stream sFileOutput mData.  
    output stream sFileOutput close.  
  
    finally:  
        if not msgLen eq 0 then set-size(mData) = 0.  
    end finally.  
end method.
```


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"
```

Various tokens resolved by `OpenEdge.Core.Util.TokenResolver`
Custom / application token resolution supported via PUB/SUB

Default tokens

Token substitutions are allowed for file names

```
the token format is ${<token>}, where
token = group "." arg
groups = session | env | guid | t[ime] | web | ver[sion]
        cp | req[uest] | name | err
```

If a token cannot be resolved, or resolves to a value of ? (unknown) then the token name is used.

* Group args for SESSION

- any readable attribute on the session handle may be used

* Group args for ENV

- any env var available via OE-GETENV() may be used

* Group args for VERSION

- Correlates to the SESSION:LOCAL-VERSION-INFO
- MAJOR, MINOR, MAINT

* Group args for GUID

- a default-format GUID is used

* Group args for T (T= time).

- Values are taken from the time at which the file name is being built.
- The args are based on <http://en.cppreference.com/w/c/chrono/strftime>

TODAY: An ISO-DATE formatted DATE value is used

NOW) An ISO-DATE formatted DATETIME-TZ value is used

ISO)

HTTP : The datetime formatted as per <https://tools.ietf.org/html/rfc7231#section-7.1.1.1>

YYYY : The current year, incl century

YY : The current year, sans century

BB : The full month name (from the MONTH_LONG property)

B : The shortened month name (from the MONTH_SHORT property)

MM : The integer month value, with leading 0 if needed

M : The integer month value with no leading 0

DD : The integer day value, with leading 0 if needed

D : The integer month value, with no leading 0

HH : The hour value, in 24-hour clock format (ie 18 for 6pm)

H : The hour value, in 12-hour clock format (ie 6 for 6pm)

MMM : The minute value, with leading 0

SS : The second value, with leading 0

SSS : The millisecond value, with leading 0

Z : The timezone (based on the current session), with leading +/-

PP : The AM/PM indicator, as AM or PM

P : The AM/PM indicator, as A or P

AA : The full day of the week, from the WEEKDAY_LONG property

* Group args for REQ (request).

Will return ? if we're not in a request (ie startup event procs). Values are taken from the session:current-request-info

TPT : The adapter type (transport) for this request

CCID : The client context id

ID : The current request id

SESSION : The current session id

THREAD : (PASOE) the current thread id

* Group args for WEB

WEBAPP["." webapp-type]

webapp-type

NAME : the context/webapp name (default)

PATH : the fully-qualified path of the webapp

SVC

any other cgi value

* Group args for CP (client principal)

credential-arg "." db-name

credential-arg

The current user will be used (from the request info or the security-policy)

UID : The current user id

QUID : The qualified user id (user@domain)

DOMAIN : The domain name of the current user

PROP "." property-name : Returns the value of a property in the CP

property-name : The name of a property to return

db-name

An optional logical db name from which to extract the CP. If none is set, use the security-policy

* Group args for NAME

tokenArg = format-expression "." logger-name

- logger-name : a named-hierarchy dotted-name

- named-hierarchy

something like OpenEdge.Net.DataObject.DataObjectHandler (a logger name) will become

O.N.D.DataObjectHandler (default or .1K)

o.n.d.DataObjectHandler (.1L)

OE.N.DO.DataObjectHandler (.1C)

OE.N.DO.DOH (.0C)

- format-expression

keep-expr case-expr

keep-expr:

number of significant entries to keep (from right)

0 : All entries will be trimmed (ie zero kept)

WebHandler tokens

```

Group args for WEB
WEBAPP["." webapp-type]
    webapp-type
        NAME : the context/webapp name (default)
        PATH : the fully-qualified path of the webapp
URI      : The complete request URI
URI.TEMPLATE : The URI template used to select a webhandler
URI.HOST  : The host to which the request was made
URI.PORT  : The port to which the request was made
VERB      :
METHOD    : the http method of the request
SCHEME    : the scheme of the request (HTTP or HTTPS)
TRANSPORT :
TPT       : the transport, including path. So generally /web
SERVICE  :
SVC       : PATH.1 or PathInfo[1]. The service name is, by convention, the first path segment after
           the transport
PATH      : The path after the transport
PATH.PARAM : Comma-separated list of the path parameter names, plus TEMPLATE, FINAL_MATCH_GROUP
PATH.<param> : value for the path parameter named <param>. Blank if the <param> does not exist
PATH.<n>     : value of the nth path segment of the PathInfo (after the transport)
HDR.<name>  : the header value for <name>
COOKIE.<name> : the complete cookie value for <name>
QRY.<name>  : The query string value of <name>
CGI        : a CSV list of the CGI variable names
CGI.<name>  :
CTX.<name>  : The value of the CGI variable
REMOTE"." remote-client-data
    remote-client-data
        HOST      : The hostname of the remote client
        PORT      : The port of the remote client
        ADDRESS   : The IP address of the remote client
LOCAL"." local-server-data These values may differ from the request URI due to proxies or whatnot
    local-server-data
        HOST      : The hostname of this instance
        PORT      : The port of this instance
        ADDRESS   : The IP address of this instance
<value>      : If none of the above, treat as CGI.<name>

```

Application tokens

- Resolve application tokens via PUB/SUB events

```
Consultingwerk.Util.TokenResolver:TokenResolved:Subscribe(TokenResolvedHandler).
```

```
/* telemetry_${telemetry.instanceIdentifier}_${t.YMD}_${t.HMS}_${req.agent}_${req.session}.ndjson */  
cFileName = SUBSTITUTE ("&1/&2":U,  
                        THIS-OBJECT:OutputDirectory,  
                        TokenResolver:Resolve(THIS-OBJECT:FileNamePattern)).
```

```
Consultingwerk.Util.TokenResolver:TokenResolved:Unsubscribe(TokenResolvedHandler).
```


Application tokens

- Event handler provides value(s)

```
/**  
 * Purpose: Event handler for the TokenResolved event of the  
 *         TokenResolver  
 * Notes:  
 * @param sender The reference to the sender of the event  
 * @param e The TokenResolverEventArgs for this event  
 */  
METHOD PROTECTED VOID TokenResolvedHandler (  
                                     sender AS Progress.Lang.Object,  
                                     e AS TokenResolverEventArgs):  
  
    IF e:TokenGroup = "telemetry":U AND  
       e:TokenArg    = "instanceIdentifier":U THEN  
        ASSIGN e:TokenValue = THIS-OBJECT:InstanceIdentifier .  
  
END METHOD.
```

Tokens in messages

```
1. using OpenEdge.Logging.*.  
2.  
3. define variable Logger as ILogWriter no-undo.  
4.  
5. // Get a reference to a logger  
6. assign Logger = LoggerBuilder:GetLogger("Demo.PUG.UK").  
7.  
8. // now we write our messages ...  
9. Logger:Info("UK PUG started").  
10.  
11. Logger:Warn(substitute("Webinar speaker &2 started at &1",  
12.                        iso-date(now), get-db-client():user-id)).  
  
13. Logger:Warn(substitute("Webinar speaker ${cp.uid} started at ${t.now}").  
  
14. // yak yak yak  
15. Logger:Trace("Webinar session end").
```

Uses SECURITY-
POLICY:GET-CLIENT()

Tokens in configuration file

```
{  
  "DEFAULT_LOGGER": "Example.Server.Request",  
  "logger": {  
    "Example.Server.Request": {  
      "logLevel": "ERROR",  
      "filters": [  
        {  
          "name": "TOKEN_FORMAT",  
          "format": "[${t.iso} ${msg.level} ${req.tpt}] ${msg} logged by ${cp.quid}"  
        },  
        {  
          "name": "NAMED_FILE_WRITER",  
          "fileName": "${session.temp-dir}/server-${t.today}-${req.id}.log",  
          "appendTo": true  
        }  
      ]  
    }  
  }  
}
```

Formats all
messages

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').
```

```
// Write log message with MDC tokens  
logger:Info('My name is ${mdc.myName} ').
```

```
My name is 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_\(C\)](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 default `<salt>` value is a base64-encoded UUID generated by the AVM

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

Data anonymisation



```
{ "logger": {  
  "Example.Custom": {  
    "filters": [  
      {  
        "name": "ANON_FORMAT",  
        "tokensToAnon": "mdc.myName",  
        "hashAlgo": "SHA-512"  
      }  
    ]  
  }  
}
```

```
// Set MDC key values somewhere in the application code  
OpenEdge.Logging.MDC:Put('myName', 'Michael Caine').  
// Write log message with MDC tokens  
logger:Info('My name is ${mdc.myName} ').
```

My name is

`6jO4MxS3PE5NOFI0VBIW/Tg$818LeypBlbzBhbER+I+UC6dyD7wwZRpmcskR/UW/Q1a5675FM3htnhVT5eeB3uFmutURB
i+0siOjjAjuP7rFhA==`



Customising loggers

Adding your own filters

```
{ "logger": {  
  "Example.Custom": {  
    "filters": [  
      { "name": "TRANSLATION_FILTER",  
        "toLang": "af",  
        "serviceURI": "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0",  
      },  
      "REVERSED_WORDS_FILTER",  
      { "name": "ELASTIC_SEARCH_WRITER",  
        "serviceURI": "http://localhost:9200"}  
    ] } },  
  "filter": {  
    "REVERSED_WORDS_FILTER": "Example.Filters.ReverseWordsFormat",  
    "TRANSLATION_FILTER": {  
      "type": "Example.Filters.TranslatedMessageFormat",  
      "builder": "Example.Builders.TranslatedMessageFormatBuilder"  
    },  
    "ELASTIC_SEARCH_WRITER": {  
      "type": "Example.Filters.ElasticSearchWriter",  
      "builder": "Example.Builders.ElasticSearchWriterBuilder"  
    }  
  } } }
```


Adding your own filters

```
{ "logger": {  
  "Example.Custom": {  
    "filters": [  
      { "name": "TRANSLATION_FILTER",  
        "toLang": "af",  
        "serviceURI": "https://api.cognitive.microsofttranslator.com/translate"},  
      { "name": "REVERSED_WORDS_FILTER",  
        "serviceURI": "http://localhost:9200"}  
    ],  
    "filter": {  
      "REVERSED_WORDS_FILTER": "Example.Filters.ReverseWordsFormat",  
      "TRANSLATION_FILTER": {  
        "type": "Example.Filters.TranslatedMessageFormat",  
        "builder": "Example.Builders.TranslatedMessageFormatBuilder"  
      },  
      "ELASTIC_SEARCH_WRITER": {  
        "type": "Example.Filters.ElasticSearchWriter",  
        "builder": "Example.Builders.ElasticSearchWriterBuilder"  
      }  
    }  
  }  
}
```

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

Adding your own filters

```
{ "logger": {  
  "Example.Custom": {  
    "filters": [  
      { "name": "TRANSLATION_FILTER",  
        "toLang": "af",  
        "serviceURI": "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0",  
      },  
      "REVERSED_WORDS_FILTER",  
      { "name": "ELASTIC_SEARCH_WRITER",  
        "serviceURI": "http://localhost:9200"}  
    ] } },  
  "filter": {  
    "REVERSED_WORDS_FILTER": "Example.Filters.ReverseWordsFormat",  
    "TRANSLATION_FILTER": {  
      "type": "Example.Filters.TranslatedMessageFormat",  
      "builder": "Example.Builders.TranslatedMessageFormatBuilder"  
    },  
    "ELASTIC_SEARCH_WRITER": {  
      "type": "Example.Filters.ElasticSearchWriter",  
      "builder": "Example.Builders.ElasticSearchWriterBuilder"  
    }  
  } } }
```



Adding your own filters

```
{ "logger": {  
  "Example.Custom": {  
    "filters": [  
      { "name": "TRANSLATION_FILTER",  
        "toLang": "af",  
        "serviceURI": "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0",  
      },  
      "REVERSED_WORDS_FILTER",  
      { "name": "ELASTIC_SEARCH_WRITER",  
        "serviceURI": "http://localhost:9200"}  
    ] } },  
  "filter": {  
    "REVERSED_WORDS_FILTER": "Example.Filters.ReverseWordsFormat",  
    "TRANSLATION_FILTER": {  
      "type": "Example.Filters.TranslatedMessageFormat",  
      "builder": "Example.Builders.TranslatedMessageFormatBuilder"  
    },  
    "ELASTIC_SEARCH_WRITER": {  
      "type": "Example.Filters.ElasticSearchWriter",  
      "builder": "Example.Builders.ElasticSearchWriterBuilder"  
    }  
  } } }
```



Adding your own logger

- The LoggerBuilder has a plug-in design to create logger instances
 - Based on the same named-hierarchy

```
class Example.Builders.TranslationLoggerBuilder inherits LoggerBuilder:  
  /* Constructs the actual logger instance  
   @return ILogWriter A new logger */  
  method override protected ILogWriter GetLoggerInstance():  
    define variable filterList as ILoggerFilter extent 2 no-undo.  
  
    /* Let's say we only want translation for this group of loggers */  
    filterList[1] = new TranslatedMessageFormat().  
    filterList[2] = new NamedFileWriter(session:temp-dir + 'translations-only.log', true).  
  
    return new Logger(this-object:LoggerType, LogLevelEnum:Trace, filterList).  
  end method.  
end class.
```

```
OpenEdge.Logging.LoggerBuilder:Registry:Put('TranslationOnly', get-class(TranslationLoggerBuilder)).
```


Why write a custom logger?

- Wrap existing logging libraries
 - Stable / consistent API
- Simpler configuration
 - Hard-coded filter list / fixed output location / etc



**So, where is that cloud that you
promised us?**

Elastic Search Writer

```
{ "logger": {  
  "Example.Custom": {  
    "filters": [  
      { "name": "TRANSLATION_FILTER",  
        "toLang": "af",  
        "serviceURI": "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0",  
      },  
      "REVERSED_WORDS_FILTER",  
      { "name": "ELASTIC_SEARCH_WRITER",  
        "serviceURI": "http://localhost:9200"}  
    ] } },  
  "filter": {  
    "REVERSED_WORDS_FILTER": "Example.Filters.ReverseWordsFormat",  
    "TRANSLATION_FILTER": {  
      "type": "Example.Filters.TranslatedMessageFormat",  
      "builder": "Example.Builders.TranslatedMessageFormatBuilder"  
    },  
    "ELASTIC_SEARCH_WRITER": {  
      "type": "Example.Filters.ElasticSearchWriter",  
      "builder": "Example.Builders.ElasticSearchWriterBuilder"  
    }  
  } } }
```



Troubleshooting

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
 - Errors written to either the LOG-MANAGER / Agent log or `SESSION:TEMP-DIR/loggerbuilder.log`
- If any one filter throws an error, the filter chain breaks

So you still *always* get a valid object

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
 - Easy API
 - Simple extension and customization
 - In the box since OE 11.7.0

peter.judge@consultingwerk.com



