Embedding CPython in the OpenEdge ABL

Pug Challenge 2017 - Prague
Tijs Wickardt
Panaedra Software
https://github.com/panaedra
Outline

1. (~15 mins)
   Demonstrating capabilities of integrating Python and OpenEdge

2. (~5 mins)
   Two simple .p + Python examples

3. (~15 mins)
   Embedding: Technical caveats

4. (~15 mins)
   Panaedra OO platform makes it easy
But first!

What/who is Panaedra Software?

https://github.com/panaedra
Demo #1/3

Validating data with regular expressions in ABL

Google for "e-mail regex" ->

(?:[a-z0-9!#$%&'*=~/\^_`\{\}\]+(?:\.[a-z0-9!#$%&'*=~/\^_`\{\}\]+)*"(?:[x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f])\\\\[x01-\x09\x0b\x0c\x0e-\x7f]*)*@(?:(?:[a-z0-9]\*[a-z0-9])?(?:[a-z0-9]\*[a-z0-9]?)\|[\[:](?:25[0-5]|2[0-4]\[0-9]|0[1-9]\[0-9]?)\|[0-9][0-9]?)\.|{3}(?:25[0-5]|2[0-4]\[0-9]|0[1-9]\[0-9]?)\|[a-z0-9]\*[a-z0-9]:(?:[x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f])\\\\[x01-\x09\x0b\x0c\x0e-\x7f]+))
Regular expressions definition:

A sequence of characters that define a search pattern. Usually this pattern is then used by string searching algorithms for "find" or "find and replace" operations on strings.

Concept coined in 1950, still very much used today.
Demo #1/3

Regular expressions simple example:

"\bhello\b" searches for "hello" between word boundaries
Demo #1/3

Usage in ABL:

A regular expression (regex) is very useful for validating string data. It is not directly supported in the ABL (although the ABL has support for XSD validation, which has optional regex validation internally)

With Python: very easy and powerful.
Demo #1/3

Running on Linux in an ABL CHUI

```
$ e-mail address
this@looks.alright
this does not look@alright
would_a_wildcard_be_*@valid@check.me
would_braces_be_(valid)@check.me
```

Valid: yes

Valid: no

Valid: yes

Valid: no
Demo #1/3

Running on Windows in an ABL GUI

<table>
<thead>
<tr>
<th>e-mail address</th>
<th>valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>this is not right.com</td>
<td>no</td>
</tr>
<tr>
<td><a href="mailto:this-is@better.com">this-is@better.com</a></td>
<td>yes</td>
</tr>
</tbody>
</table>
Demo #2/3

Showing a Grafana Graph with InfluxDB from ABL
Commonly used Grafana setup:

1. A metrics collector, like Telegraf
2. A time-series database, like InfluxDB
3. A Grafana Webserver

In this demo, we use all above. Next to that, we also write metrics from an ABL session, using the Python Bridge, and an "influxdb" module.
Demo #2/3

Server stats (has nothing to do with the ABL nor our python bridge, uses Telegraf)
Server stats (can focus on _progres processes, using Telegraf)
Now let's use it for our own data...

"sports.db" data from 1993 sounds definitely like it deserves a proper modern visualisation.

We use Order and Order-Line, and show some stats.
Demo #2/3

Order and Order-Line stats:
Demo #3/3

Profiling an ABL fragment using a monotonic clock, and generating an Excel Workbook with the gathered profiling metrics, on Ubuntu (Linux)
Did you say nano-seconds?
Yup, very useful if you need a fine grained analysis.
Profiling and benchmarking should ideally be reproducible at a later date.

If your technical infrastructure changes, or your upgrading a runtime or an OS, you can re-run your benchmark, helping you to predict the enduser impact.
We chose to produce an Excel sheet during profiling, with all relevant data inside the report.

Benchmark data is visualized in graphs, and accompanied by:
1. Environment variables
2. Sourcecode repository versions
3. Operating system details
Demo #3/3

HqTimestamp Benchmark Excel sheet
-End of Demos-

(well, the end of demo-ing the demos, and the start of looking at the sourcecode of the demos)

next:

Two simple .p + Python examples
What are you looking at?

**LAPTOP**

Windows 10

Eclipse Neon w. OpenEdge

Vagrant

Oracle VirtualBox

Linux: Ubuntu Xenial 16.04.3 LTS
sc_mspawnthon:PyEval(
"return 'Hello ABL, here is Python'",
""").

(optional input data)
Simple .p #2/2

tw_pythonbridge_demo_01.p
Debugging

Used to Eclipse debugging?

Using PyDev, you can keep on going, switching between ABL and Python seamlessly.
Debugging

Remote debugging using PyDev

demo: Remote debugging from Linux to your Windows Eclipse IDE, and auto-switch-over from ABL to Python in one process, on Windows.
-End of code samples-

next:

Embedding - Technical Caveats
Technical Caveats

1. **Memory violations**
   Do things right, and they will not occur

2. **IPC**
   Intra-process process communication

3. **OS signal handling**
   What happens with OS signals

4. **Multithreading and signal masks**
   Panaedra has an OpenEdge ABL mask for you
Technical Caveats

Memory Violations

1. Dangerous, especially if connected to OE RDBMS by shared memory
   - Worst case scenario: DB emergency shutdown
2. The Panaedra Python Bridge is solid
   - Used on AIX production server for years
3. Test your new use cases and new modules
   - Especially third party Python modules with binary parts (*.so on Linux/Unix, *.pyd on Win)
1. OS-FILE
   - Pro: safe, performant buffering done by OS
   - Con: leftovers, privacy considerations, clunky

2. MEMPTR
   - Pro: OpenEdge ABL is pretty good at it
   - Con: DIY or no buffering, hard multi-threading

3. FIFO / Named Pipe
   - Pro: Great for multi-threading (tip: use blocked mode),
     OS buffering, OS throttling
   - Con: copy-lob/write-xml=no go, complex

4. (not advised) Sockets (port allocation), SIGPIPE

5. (impossible atm) Developer events (U1, U2,...)
1. OpenEdge ABL runtime Linux signal handlers
   - Are NOT thread safe (makes sense)
2. This is only a concern when multi threading
   - Inline python calls are fine until end of session
3. Panaedra framework ON STOP / ON QUIT
   - You'll want this.
   - Shuts down Python interpreter, which calls Python OO destructors, a.o.
   - Read the next slide...
Technical Caveats

Multithreading and signal masks

1. Very important for multi threading
   - NOT specific to Python nor the Python Bridge

2. You need a signal mask (OS API Call)
   - Or else: each thread will invoke the OpenEdge non-thread-safe ABL runtime signal handler, causing panic!

3. Panaedra framework has one for you
   - Implemented in C
   - Simple and human readable
-End of caveats-

next:

Panaedra OO platform makes it easy
1. Bootstrapping
   - ON STOP, ON QUIT
2. Can I use the python bridge without this?
   - See: panaedra_oe_demo_lowdep
3. What are the main pythonbridge classes?
   - c_mspython_call
     - Inherits: c_mspython_sys_base
   - sc_mspython
     - If needed, for early initialization
1. Multiple methods for exchanging data
   c_mspython_call, precompiled Python plumbing for your own specific call
   - PyRunU() - Unbuffered, longchar in+out
   - PyRunUB() - Longchar in+bytes out
   - PyRunUBB() - Bytes in+out

2. Unbuffered/Buffered: An unbuffered memptr lives in the bridge, and is preferred. Buffered (memptr lives in ABL): not exposed via OO anymore, less performant, more memcpy.
Panaedra platform

GitHub

Panaedra / panaedra_oe_platform_base

Panaedra Progress OpenEdge ABL Platform Base

openedge-abl  progress-openedge  mvvc  n-tier  Manage topics

📅 1,711 commits  🌎 1 branch

Branch: master  New pull request
Questions?
Special Thanks

Panaedra Software thanks:

1. Progress
   - Gary Clink
   - Eduard Smeets
2. Bertus
   - Huibert Kaat
   - Peter Stip
3. Flusso
   - Bert Tukkers
4. ConsultingWerk
   - Mike Fechner
5. EMEA PUG
That’s (almost) all there’s to it 😊

Thanks for listening