Track 1: The Empowered Business

AKIOMA simplifying complexity.

PROGRESS EXCHANGE 2014

"Using Progress OpenEdge, Corticon, Rollbase and Node.js to create a dynamic, rule- and model-driven Web-UI"

» Mike Liewehr, AKIOMA Software

» Frank Hilhorst, Progressive Consulting

Agenda

The Idea

The Solution

Technical overview

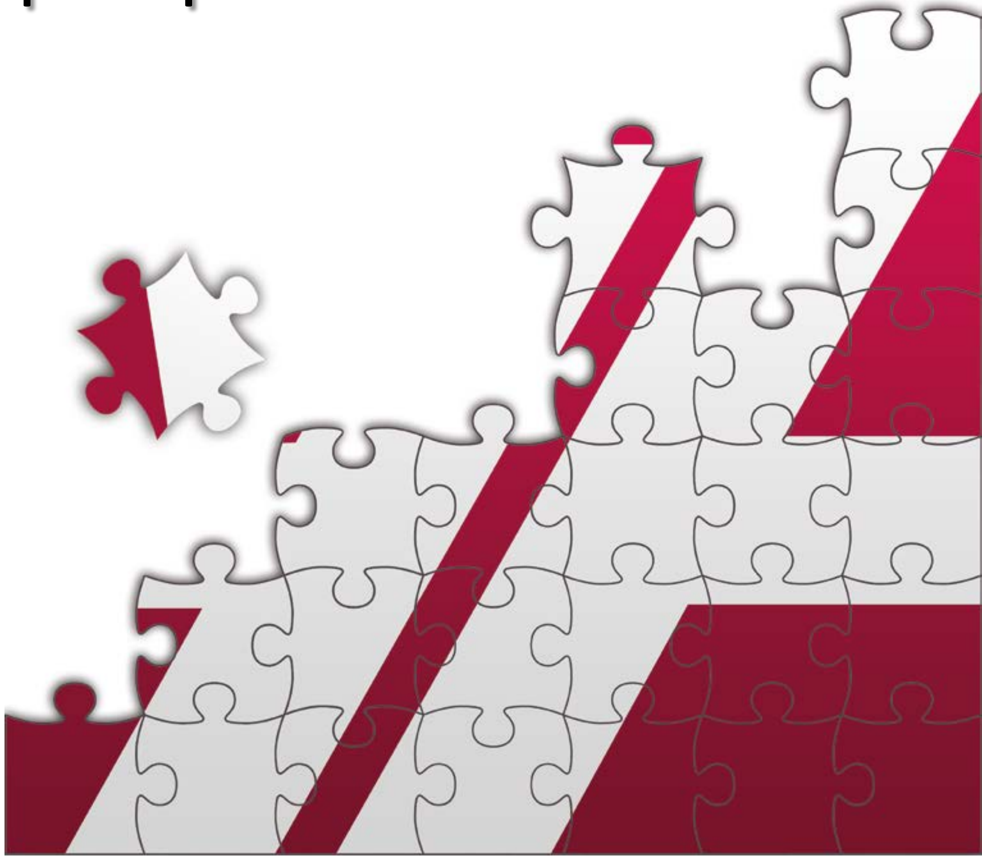
Architecture explained on a live sample

Detailed technical explanation

Q&A

The Idea

"Building web based configuration forms for self service sales of complex products without coding"



The Idea

Build complex, web based configuration forms without coding

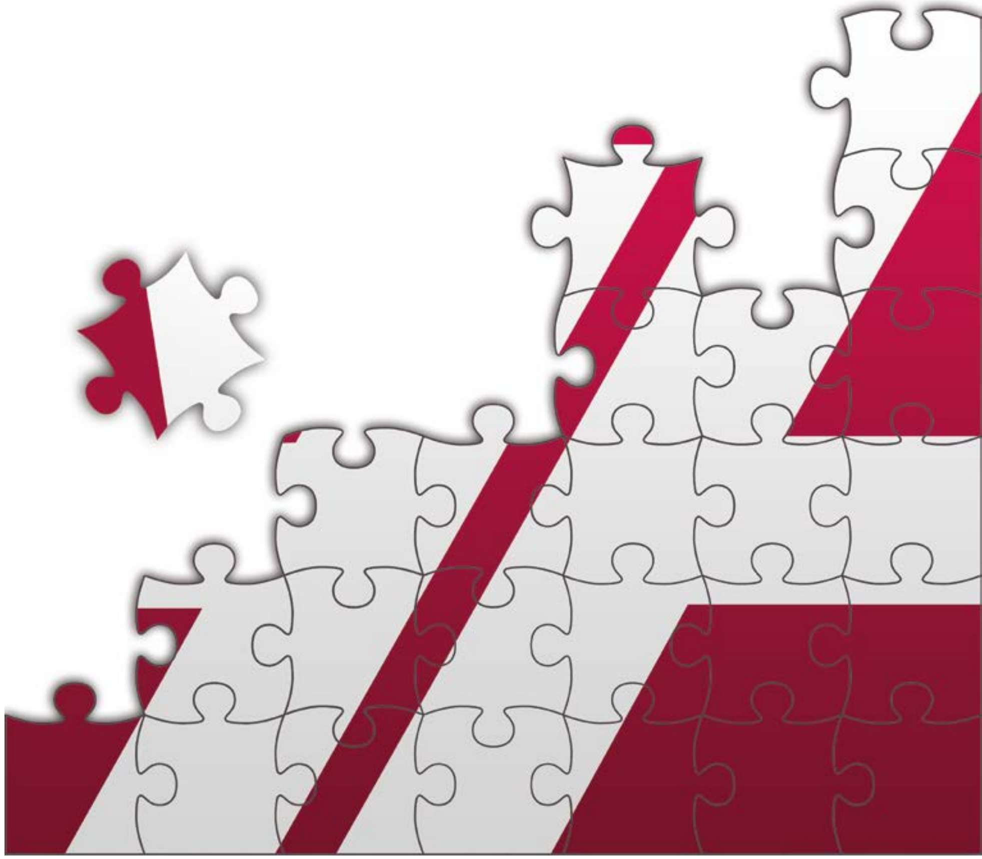
Examples:

…Car configurator

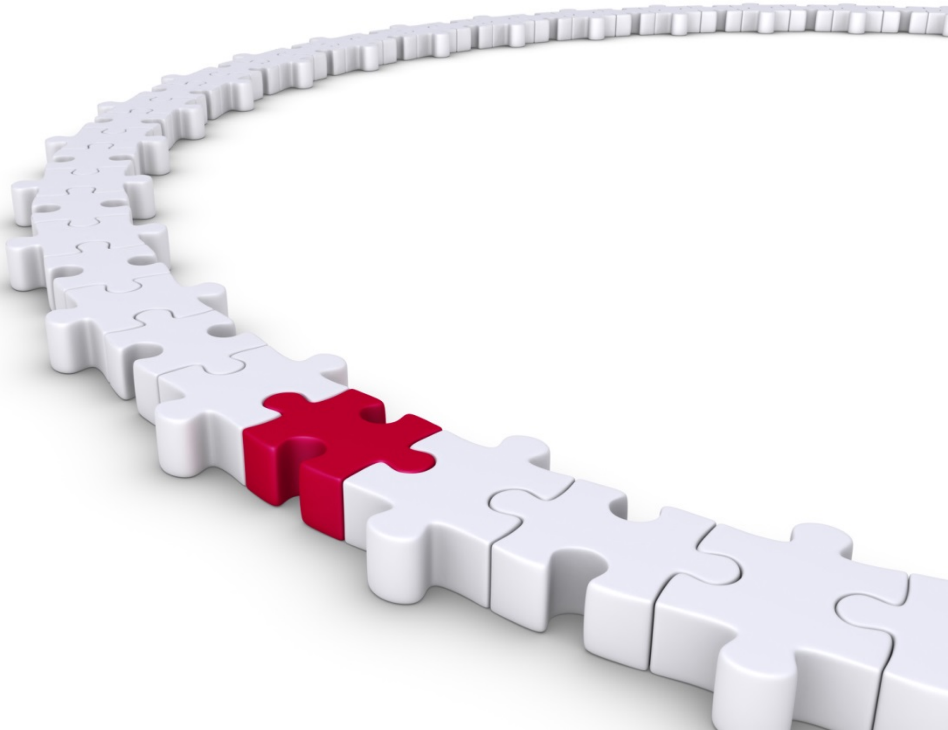
…Complex custom-built machines

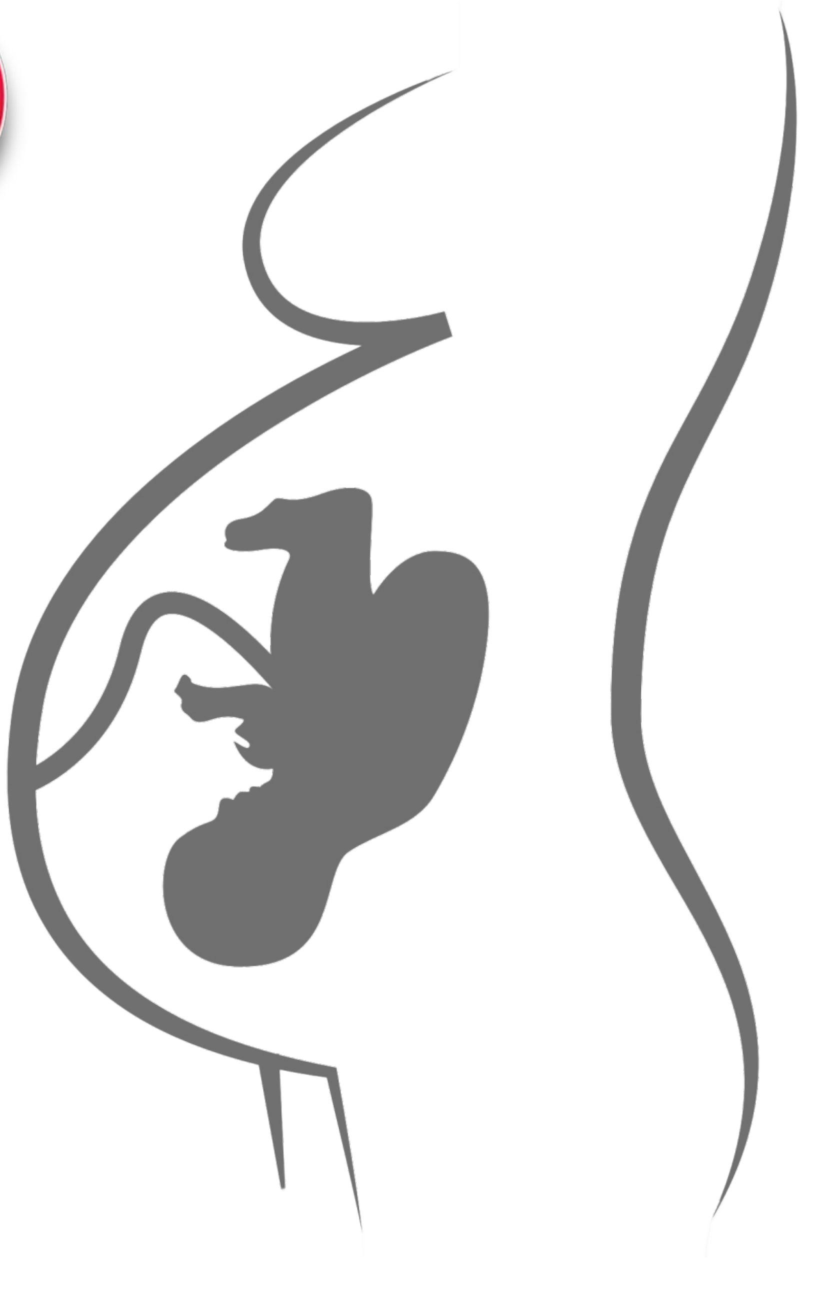
…Online Insurance Enrollment

→ Technically these are all similar



The Concept





…code rules

…code and design UI

…build database structure

AKIOMA
simplifying complexity.

Dramatically reduced time-to-market

Lower costs

No errors

Requirements and challenges

Challenges that came up when building the solution…

Data definitions vary greatly (cars, machines, people, policies…)
- Need a way to define data definition by end-user

Complex rules, that are fully customer driven
- Allow to define complex rules without coding

Rich Browser-based UI
- UI has to be very flexible and dynamic, support for rich components

Performance!
- UI has to adapt in realtime

challenges

Challenges that came up when building the solution…
…and the Products to solve them:

Data definitions vary greatly (cars, machines, people, insurance…)
- Progress Rollbase

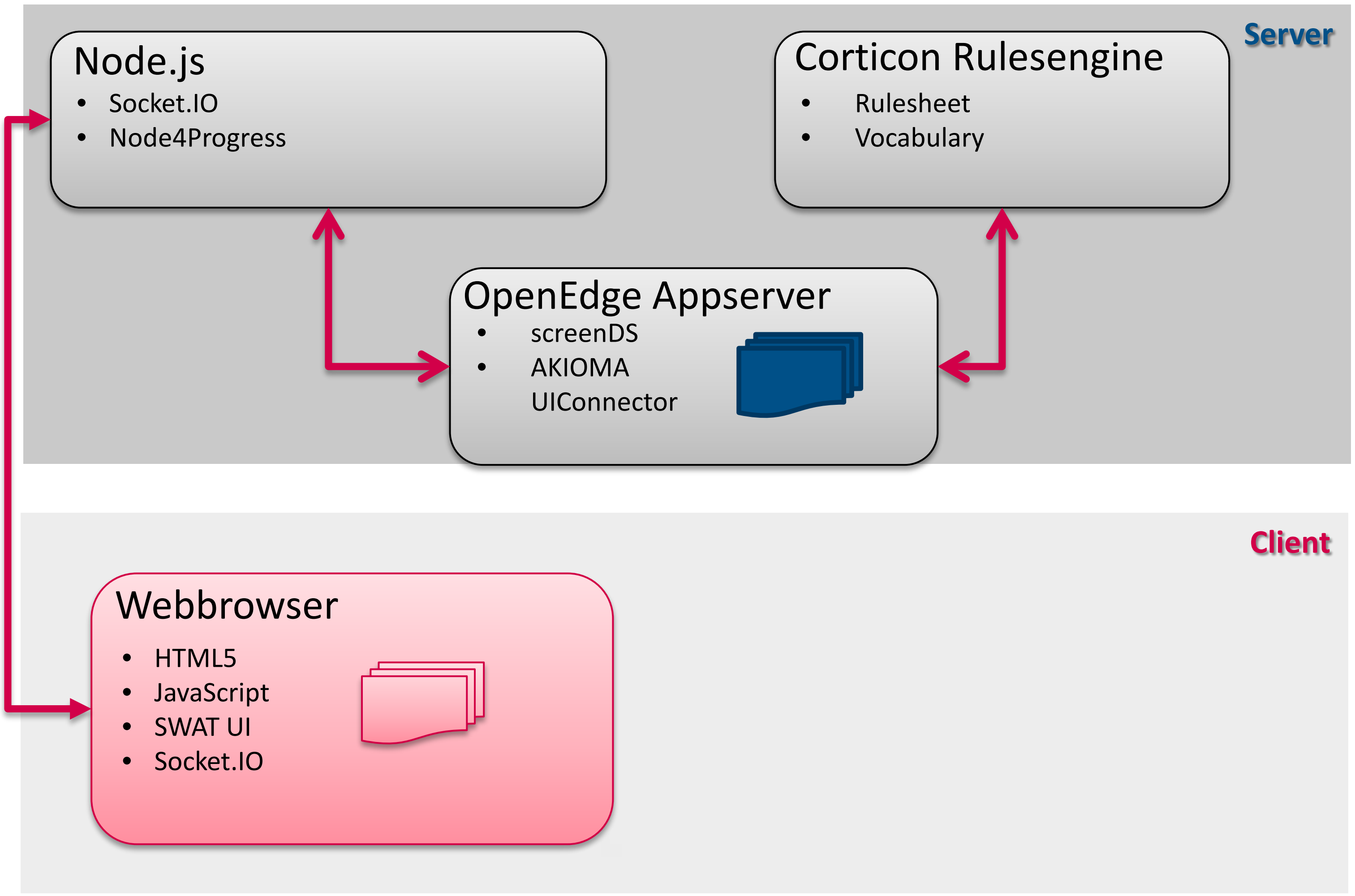
Complex rules, that are fully customer driven
- Progress Corticon

Rich UI which has to be very flexible and dynamic
- AKIOMA HTML5-Client

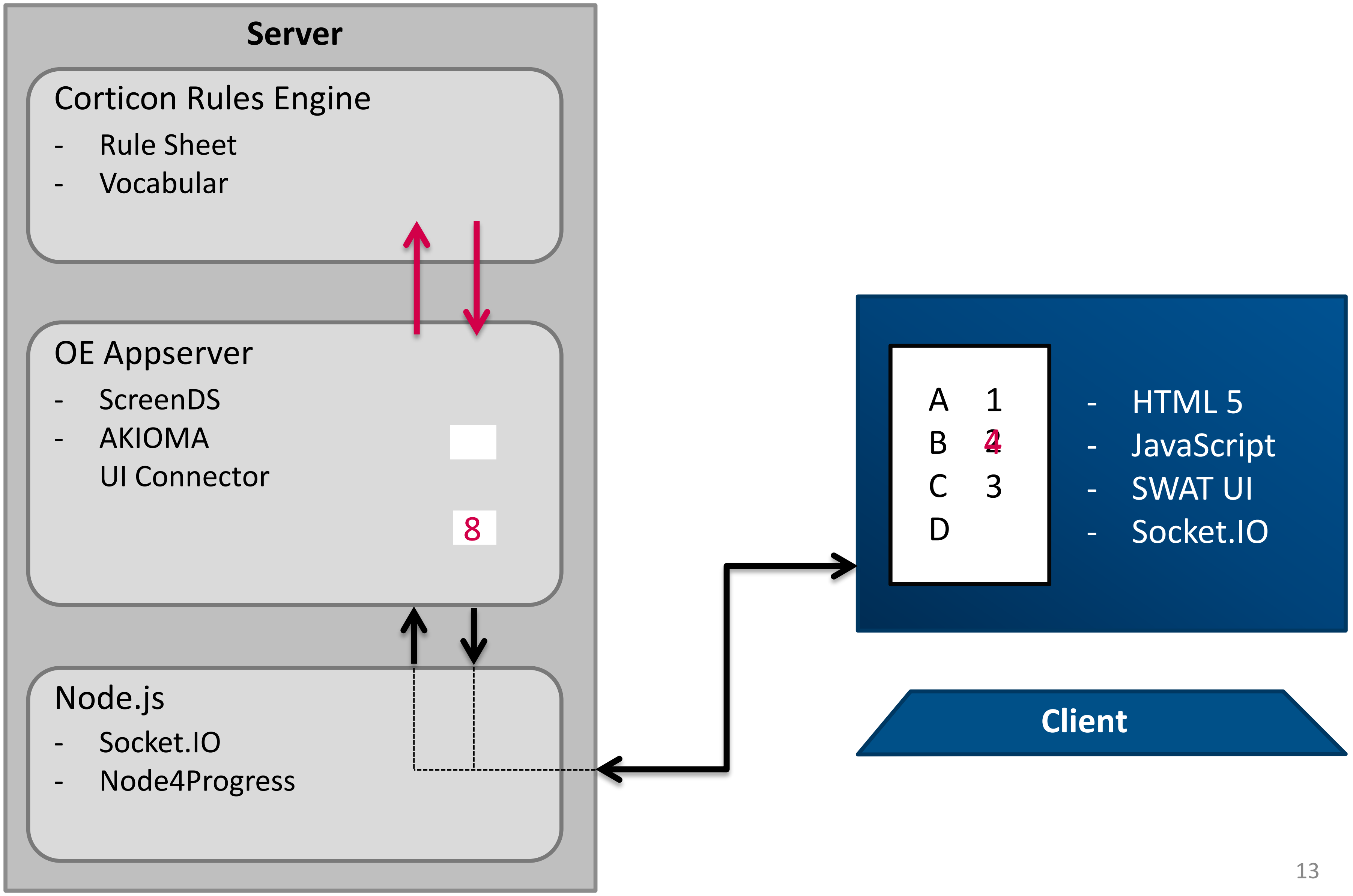
Performance! -> UI has to adapt in realtime
- Node.JS / Socket.IO / node4Progress / ABL (and some magic… ☺)

Solution Live Demo

Technical details



Technical details



Detailed technical explanation

Detailed explanation of the Architecture

Introduction to Node & Node4Progress

- About Node
  - Server side javascript engine
  - Lightweight, scalable and fast
  - Implements non-blocking programming model
    - Involves using callback procedures
  - Allows you to build web applications with features that cannot be accomplsihed any other way
  - Hot technology
    - Microsoft is going Node, Progress is going Node, everybody is going Node
- About node4progress
  - Bridge for accessing business logic on appserver directly from Node

Node4Progress Mission

- Provide flexible and fast way to access business functionality of the appserver
  - Without additional effort to support new appserver procedures
  - Minimizing the number of middleware components needed (e.g. Tomcat)

Node4Progress Features and characteristics

- Small foot print (4MB)
- Fast
- Easy to install
  - Npm install node4progress
- Communicates with appserver through Java OpenClient
- Supports 2 methods for calling an appserver procedure
  - Calling a handler
  - Dynamic appserver call
- Employs dataset object modelled after Progress ProDataset

Converting output parameters of an Appserver procedure into a JSON structure

```
{"output":{
   "OutputParametes-1" : ".....",
   "OutputParametes-2" : ".....",
   "dsCustomer" : {...},
   “ErrMsg" : ""
   },
"error": "“
}
```

- Root level *output* tag contains output parameters parameters from appserver procedure
- Root level *error* tag contains error message if execution of appserver procedure failed

Calling a handler

```
var node4progress = require("node4progress")(null);
var handler    = "handlers/CustomerHandler.p";
var inputPars = "NumCustomersToPull=2";

node4progress.callHandler(handler, inputPars,
                                        function(err,result){
                console.log(result);
});
```

Approach to calling any appserver procedure dynamically

- Define appserver procedure to call
  - Node4progress.setAppSvrProc(....);
- Define each parameter
  - Node4progress.setParameter(......);
- Define callback procedure
- Invoke appserver procedure
  - Node4progress.invoke(callback);

Calling appserver procedure dynamically (example)

```
var node4progress = require("node4progress")(null);
node4progress.setAppsvrProc("handlers/CustomerHandler.p","",false,true);
node4progress.setParameter(
  "InputPars",
  "longchar",
  "input",
  "NumCustomersToPull=5&batchNum=2","");
node4progress.setParameter("OutputPars","character","output","","");
node4progress.setParameter("dsCustomer","dataset-handle","output","","");
node4progress.setParameter("ErrMsg","character","output","","");

node4progress.invoke(function(err,result){
   if(err){
               console.log("ERROR->"+err);
   }else{
               console.log(result);
               result=JSON.parse(result);
   }
});
```

Parameter types supported

- Data-types
  - Character
  - Integer
  - Int64
  - Decimal
  - Longchar
  - Date/Date-time
  - Table-handle
  - Dataset-handle
- Modes
  - Input
  - Input-output
  - Output (append)

Challenges of working with data

- When passing data as input
  - Data needs to be formatted exactly right
    - i.e. In read-json format
    - Working directly with raw JSON structure is error prone
- Solution
  - The node4progress dataset and temp-table objects

The node4progress dataset object

- Getting the data in a dataset object

```
dataset = node4progress.getDataSet(
    “dsCustomer”,
     AppSvrOutputJson);
```

- Objects provided
  - Dataset
  - TempTable
  - Buffer
  - BufferField
- Method provided at the dataset level
  - copyDataset, emptyDataset, writeJson
- Methods provided at the TempTable level
  - forEach, bufferCreate, bufferCopy, bufferDelete, copyTempTable, emptyTempTable, findFirst, findLast, writeJson
- Methods provided at the buffer level
  - Display, writeJson

The node4progress dataset object

- Getting the data in a dataset object

```
dataset = node4progress.getDataSet(
    “dsCustomer”,
     AppSvrOutputJson);
```

- Navigating the data in a temp-table

```
dataset.$(“ttCustomer”).forEach(function(ttCustBuf){
     custNum=ttCustBuf.$(“cust-num”).$(“bufferValue”);
     name=ttCustBuf.$(“name”).$(“bufferValue”);
});
```

- Creating a new record in a temp-table

```
ttCustBuf = dataset.$(“ttCustomer”).bufferCreate().
```

- Populating the fields in a newly created temp-table record

```
ttCustBuf.$(“Cust-num”).bufferValue(100);
ttCustBuf.$(“name”).bufferValue(“frank Hilhorst”);
```

The node4progress dataset object (continued)

- Buffer-copy data into a temp-table buffer

```
data = {“cust-num”: 1180, “name”:”Frank Hilhorst”,”City”:”Miami”};

ttBuf = dataset.$(“ttCustomer”).findFirst();

ttBuf.bufferCopy(data);
```

- Sending updated dataset as input

```
var jsonStr = dataset.writeJson();
node4progress.setAppsvrProc("AppProc.p”."",false,true);
node4progress.setParameter(
    "dsCustomer",
    "dataset-handle",
    "input-output",
    “dsOrderSchema.p”,
    jsonStr);
```

Summary

Progress offers great products

Combined they provide even more value

The whole is more than the sum of its parts

Node.JS is here to stay and provides great value for Progress partners

Questions?

Questions?

PROGRESS EXCHANGE 2014

Visit the Resource Portal

- Get **session details** & presentation **downloads**
- Complete a **survey**
- Access the latest Progress **product literature**

www.progress.com/exchange2014