DYALOG

Belfast 2018

Introduction to Conga

All the basics of Conga and communication

Bjørn Christensen
Adám Brudzewsky

You need Conga to…

- make two Dyalog APL instances speak with each other
- make Dyalog APL speak to networked devices
- let Dyalog provide general services

All of the above may be…

- through a network
  and/or
- on a single local machine

Do not use Conga (directly) to…

- run a website → MiServer
- provide web services → SAWS
- call an APL function over a network → JSONServer
- fetch a file or do other HTTP actions → HttpCommand

What is Conga?

- Layer between APL and OS sockets
- Implementation of protocols
- Multi-thread manager
- Object Oriented APL tool (new in Conga 3.0!)

Sockets

- UDP – User Datagram Protocol
- TCP – Transmission Control Protocol
- Both are on top of the IP layer (Internet Protocol)

- TCP/IP is a tunnel between endpoints

Sockets

OSI Model
Application
Presentation
Session
Transport
Network
Data link
Physical

TCP/IP
Application
Host-to-Host
Internet
Network Interface

TCP/IP Suite

Telnet	FTP	SMTP	Dyalog with Conga	DNS	RIP	SNMP
TCP				UDP		
IP						

Ethernet	Token Ring	Frame Relay	ATM

Sockets: How a connection begins

SERVER	CLIENT
listens on socket bound to address and port	
receives new connection and creates new socket for it	connects to address and port
keeps listening for more connections on old socket	

Sockets: Endpoints

- IPv4 address, e.g. 81.94.205.35 (32 bit)
  or
  IPv6 address, e.g. 2a02:2658:1012::35 (128 bit)
- Port, e.g. 80 (16 bit)

- Server listens on socket bound to address and port
- Client connects to address and port
- Client has ephemeral (unknown) port (≥50000)

Domain Name Service et al.

```
      iConga.Clt 'DYA' 'www.dyalog.com' 'http'
0 DYA
```

Name → IP address

Service → port

```
      iConga.GetProp 'DYA' 'peeraddr'
0
  IPv4 81.94.205.35:80 81 94 205 35 80
```

```
      iConga.GetProp 'DYA' 'localaddr'
0
  IPv4 192.168.17.128:54442 192 168 17 128 54442
```

Loading Conga

Load APL code

```
      )copy conga Conga
```

Load C library

```
      iConga←Conga.Init ''
```

Conga 3.0+: get events instead of errors

```
      iConga.SetProp '.' 'EventMode' 1
```

SERVER Session

```
)copy conga Conga
]load [DYALOG]/Samples/Conga/RPCServices/SimpleReverser -nolink

sr1←Conga.Srv 5000 #.SimpleReverser
sr1.Start
```

```
)wsid SERVER
]caption editor {WSID}    ⍝ Don't do this in RIDE!
```

CLIENT Session

```
)copy Conga
)wsid CLIENT
```

```
iConga←Conga.Init   ''                          ⍝ load lib and create instance
iConga.SetProp  '.'  'eventmode'  1             ⍝ treat errors as events

iConga.Clt      'MyCon'   ''   5000             ⍝ new client connection
iConga.Send     'MyCon.MyCmd'   'repaid desserts'      ⍝ send command
iConga.Wait     'MyCon.MyCmd'   1000            ⍝ wait for response
iConga.Close    'MyCon'                         ⍝ close connection
```

Basic Command Connection

Hands on 1

Follow the cheat sheet

Recap — what did we just do?

```
iConga.Clt 'MyCon' '' 5000
```

Syntax of Wait

Object: ConnectionName.CommandName

```
iConga.Wait 'MyCon.MyCmd' 1000
```

Timeout: How long we'll wait (ms)

Data for event

Event that has occurred

Object: ConnectionName.CommandName

Error code: Zero means "success" — result has this **4**-element format

Error-result of Wait (with 3.0+ setting)

```
iConga.SetProp '.' 'EventMode' 1
iConga.Wait 'MyCon.MyTypo' 1000
```

1010	ERR_OBJECT_NOT_FOUND	

Error description (may be ' ')

Error type

Error code: Non-zero means failure — result has this **3**-element format

SimpleReverser

```
:Class SimpleReverser : #.Conga.Connection

    ∇ MakeN arg
      :Access Public
      :Implements Constructor :Base arg
    ∇

    ∇ onReceive(obj data)
      :Access public override
      _←Respond obj(data(⍎data))
    ∇

:EndClass
```

Modify SimpleReverser

```
:Class SimpleReverserAddr : #.Conga.Connection
    ∇ MakeN arg;iConga;server
      :Access Public
      :Implements Constructor :Base arg

      iConga←srv.LIB
      ⎕←'Mine: ',iConga.GetProp Name'LocalAddr'
      ⎕←'Theirs: ',iConga.GetProp Name'PeerAddr'

      :For server :In iConga.Names '.'
          ⎕←'Server:',server,' Connections: ',iConga.Names server
      :EndFor
    ∇
    ∇ onReceive(obj data)
      :Access public override
      ⎕←'Command: ',obj
      _←Respond obj(data(⍎data))
    ∇
```

SERVER Session

```
]set workdir ,/<mypath>/
]save SimpleReverserAddr /<mypath>/ -makedir

sr1.Stop

sr1a←Conga.Srv 5000 #.SimpleReverserAddr
sr1a.Start
```

CLIENT Session

```
iConga.Close 'MyCon'    ⍝ Did you close it?

(err con)←iConga.Clt 'MyCon' '' 5000

con SendReceive 'repaid desserts'

iConga.Close con
```

Log Server Activity

Hands on 1a

```
∇ result←connection SendReceive text
```

1. Send a request to the server
2. Wait for a result
3. Handle all kinds of results

Keep an eye on the server's session

Don't forget to save your work!

Syntax of Send

Object: ConnectionName.CommandName

```
iConga.Send 'MyCon.MyCmd' 'repaid desserts'
```

Data: Any APL array

0	MyCon.MyCmd

Object: ConnectionName.CommandName

Error code: Zero means "success"

Modify SimpleReverserAddr

```
:Class SimpleReverserAddrPro : #.Conga.Connection

    ∇ onReceive(obj data);i;n
      :Access public override
      ⎕←'Command ',obj
      ⍝ Inform the waiting client on progress
      :For i :In ⍳¯1+n←10
          ⎕DL 4
          _←Progress obj(⎕←(⍕100×i÷n),'% done')
      :EndFor
      _←Respond obj(data(⍎data))
    ∇
```

SERVER Session

```
]save SimpleReverserAddrPro /<mypath>/

sr1a.Stop

sr1b←Conga.Srv 5000 #.SimpleReverserAddrPro
sr1b.Start
```

CLIENT Session

```
iConga.Close 'MyCon'          ⍝ Did you close it?

(err con)←iConga.Clt 'MyCon' '' 5000

con SendReceive 'repaid desserts'
]save SendReceive /<mypath>/

iConga.Names '.'              ⍝ list open connections
iConga.Close con
```

Hands on 1b

```
r←con SendReceive data
(err cmd)←2↑res←iConga.Send con data
⍝ throw error if err is non-zero
:Repeat
    err←⊃res←iConga.Wait cmd 1500
    ⍝ throw error if err is non-zero
    (err obj evt dat)←4↑res
    :Select evt
    ⍝ handle every event type
    :EndSelect
:Until evt≡'Receive'
```

Progress

- You can send any APL array back
- When Receive is called, all remaining Progress reports are discarded

Our SendReceive

```
r←con SendReceive data;cmd;err;obj;evt;dat;res
(err cmd)←2↑res←iConga.Send con data
('Send failed ',⍕⊃res)⎕SIGNAL(err≠0)/999
:Repeat
    err←⊃res←iConga.Wait cmd 1500
    ('Wait failed ',⍕⊃res)⎕SIGNAL(err≠0)/999
    (err obj evt dat)←4↑res
    :Select evt
    :Case 'Timeout'   ⋄ ⎕←'very bored...'
    :Case 'Error'     ⋄ ⎕←'ooooops: ',⍕⊃dat
    :Case 'Receive'   ⋄ r←dat
    :Case 'Progress'  ⋄ ⎕←'so far: ',dat
    :Else             ⋄ r←⍬
    :EndSelect
:Until evt≡'Receive'
```

Mode

- `'Command'` (default, if omitted)
- `'Text'` and `'Raw'`

- `'BlkText'` and `'BlkRaw'` (thin protocol)
- `'http'` (→MiServer/JSONServer/HttpCommand)

- `'Udp'` (broadcasts with no guarantee — WIP)

General Events (can happen in all modes)

- Connect
- Error
- Timeout
- Closed

Command Mode

- Implements an APL specific protocol
- APL at both ends
- Possible events: Receive, Progress
- Object trees:

  Server—⋞Connection—⋞Command—⋞Progress

  Client—⋞Command—⋞Progress

  (You may have multiple children for every —⋞)

'Text' Mode (applies to 'Raw' mode too)

- No protocol
- Can have anything on the other end
- MaxBlockSize
- Both sides use Send
- Possible events: Block, BlockLast
- Object trees:
  Server—≺Connection—≺Block
  Client—≺Block
- Optional: "End Of Message" ('EOM')

SimpleText

```
:Class SimpleText : #.Conga.Connection
    ∇ MakeN arg
      :Access Public
      :Implements Constructor :Base arg
    ∇
    ∇ sa←ServerArgs
      :Access public shared
      sa←('Mode' 'Text')('BufferSize' 1024)
    ∇
    ∇ onBlock(obj data)
      :Access public
      _←Send(⍎data)0
    ∇
    ∇ onBlockLast(obj data)
      :Access public
      Close Name
    ∇
:EndClass
```

SERVER Session

```
]save SimpleText /<mypath>/

sr1b.Stop

st2←Conga.Srv 5001 #.SimpleText
st2.Start
```

CLIENT Session

```
(err con)←iConga.Clt 'MyTxt' '' 5001 'text' 1024

(err blk)←iConga.Send 'MyTxt' 'repaid desserts'

(err obj evt dat)←4↑□←iConga.Wait con 1000


iConga.Close con
```

Basic Text Connection

Hands on 2

Follow the cheat sheet

CLIENT

```
(err con)←iConga.Clt  'MyTxt'  ''  5001  'text'  1024
```

SERVER

```
∇ sa←ServerArgs
  :Access public shared
  sa←('mode' 'text')('BufferSize' 1024)
∇
```

SimpleTextEOM

```
:Class SimpleTextEOM : #.Conga.Connection
    ∇ MakeN arg
      :Access Public
      :Implements Constructor :Base arg
    ∇
    ∇ sa←ServerArgs
      :Access public shared
      sa←('mode' 'text')('BufferSize' 1024)('EOM' '$STOP')
    ∇
    ∇ onBlock(obj data)
      :Access public
      _←Send(⍎data)0
    ∇
    ∇ onBlockLast(obj data)
      :Access public
      Close Name
    ∇
:EndClass
```

SERVER Session

```
]save SimpleTextEOM mypath/

st2.Stop

st2a←Conga.Srv 5001 #.SimpleTextEOM
st2a.Start
```

CLIENT Session

```
(err con)←iConga.Clt 'MyTxt' '' 5001 'text' 1024

(err blk)←iConga.Send 'MyTxt' 'repaid desserts$STOPraw live$STOP'

(err obj evt dat)←4↑⍬←iConga.Wait con 1000
(err obj evt dat)←4↑⍬←iConga.Wait con 1000

iConga.Close con
```

Segmented Messages

Hands on 2a

Follow the cheat sheet

SimpleTextSecure

```
:Class SimpleTextSecure : #.Conga.Connection
    ∇ sa←ServerArgs;iConga;path;file;cert
      :Access public shared
      iConga←#.Conga.Init''

      path←⎕SE.SALTUtils.DYALOG,'/TestCertificates/'
      file←path,'server/localhost'
      cert←⊃iConga.ReadCertFromFile file,'-cert.pem'
      cert.KeyOrigin←'DER' (file,'-key.pem')

      sa←('mode' 'text')('BufferSize' 1024)
      sa,←('X509'cert)('SSLValidation'64)

      _←iConga.SetProp'.' 'RootCertDir'(path,'ca/')
    ∇
```

SERVER Session

```
]save SimpleTextSecure mypath/

st2a.Stop

st2b←Conga.Srv 5005 #.SimpleTextSecure
st2b.Start
```

CLIENT Session

```
anon←⎕NEW iConga.X509Cert

iConga.SetProp '.'  'RootCertDir' (⎕SE.SALTUtils.DYALOG,'/TestCertificates/ca/')

iConga.Clt 'MyTxt'  'localhost' 5005 'text' 1024 ('X509' anon) ('SSLValidation' 64)

iConga.Send 'MyTxt'  'repaid desserts'
iConga.Wait 'MyTxt' 1000

iConga.Close 'MyTxt'
```

Secure Connection

Hands on 2b

Follow the cheat sheet

Summary of Conga Functions

```
Init
Srv
Clt
Send
Wait
Respond
Progress
```

How we used Conga functions

SERVER	CLIENT
	Init
Srv	Clt
Send	Send
Respond	
Progress	
	Wait

Recap — Questions?

1 Basic Command Connection
1a Log Server Activity
1b Report Progress

2 Basic Text Connection
2a Segmented Messages
2b Secure Connection