# New features in Conga 3.0

## One size fit all

We are only built one version of Conga 3.0 that supports both classic and unicode. There is still a 32 bit an 64 bit version of Conga. They are name conga30\_32 and conga30\_64.

I suggest that we use the same version for ride. The only possible downside which I can see is the common use of the gnutls library , so if trace is switched on in Conga trace from Ride will also appear in the log file. (See Multiple Roots)

## Dynamic load

GnuTLS is loaded dynamically when it is needed, The benefit is that we can run plain conga without gnutls, the downside is that it is a bit more tricky to know where the Congassl library is loaded from. ( I may have a task here to try to load the congassl lib from the same directory as the conga lib. I do not know how to do that on NIX)

## GnuTLS

I have upgraded to 3.4.16, and a lot of the certificate handling has changed but I hope it looks the same from the APL Side. One new thing is it is now possible to you certificates in the Microsoft Certificate store for Servers. Added Server Name Indicaction, Session Tickets. No changes on the APL Side.

## Multiple Roots

We have added the possibility to run two roots in one WS that cannot see each other. All is in place in Conga but how the APL side utilize it is a little unclear, I have implemented a classy suggestion.

Eg: DRC.Init ‘’ 🡺 DRC←DRCShared.Init ‘’

## Version

New function to return the version of Conga as an integer 30001334, if an three element 4 bytes interger vector is pass it is filled out with the same numbers.

## Serverside

### Endpoints {Overhaul.dyalog|TestEndpoints}

We cannot restrict incoming connection based on the IP address they are originating from. Endpoints can be specified on the form:

┌→──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐

│ ┌→────────┐ ┌→──────────────────────────────────────────────────────────────────────────────────────────────────────┐ │

│ │Endpoints│ │ ┌→────────────────────────────────────────────────────────────────────┐ ┌→──────────────────────────┐ │ │

│ └─────────┘ │ │ ┌→───┐ ┌→─────────────────────────────────────────────────────────┐ │ │ ┌→───┐ ┌→───────────────┐ │ │ │

│ │ │ │IPv6│ │fe80::d189:fd4:7003:a0a3/120,fe80::9df3:f956:84f5:12ab/120│ │ │ │IPv4│ │192.168.202.1/24│ │ │ │

│ │ │ └────┘ └──────────────────────────────────────────────────────────┘ │ │ └────┘ └────────────────┘ │ │ │

│ │ └─────────────────────────────────────────────────────────────────────┘ └───────────────────────────┘ │ │

│ └───────────────────────────────────────────────────────────────────────────────────────────────────────┘ │

└───────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘

One Ipv6 section and one IPv4 section with a list of ipaddress and size.

We have two sets of Endpoints AllowEndpoints and DenyEndpoints, that can specified when a server is started. Srv. If incoming connection address is present in AllowEndpoints and AllowEndpoints is specified it is allowed in unless DenyEndpoints is specified and the incoming endpoint is in the deny list.

### Pause {Overhaul.dyalog|TestPause}

We can now pause a server to receive new incoming connections.

DRC.SetProp ‘S1’ ‘Pause’ 0|1|2

0 ends pause and resume operation

1 keeps socket open but does not accept new incoming connections.

2 Closes the socket

### FIFOMode {Overhaul.dyalog|TestReadyList & TestReadylist2}

We have implemented a list of items in chronological order, this requires that no Waits are done in the object hieraki, this is not policed but behavior is unpredictable. To enable the list

DRC.SetProp ‘S1’ ‘FIFOMode’ 1

No changes to APL wait loop should be required.

If server is in fifo mode and an attempt to listen on a connection is made error 1142 is returned

### Only Connections {Overhaul.dyalog | TestThreaded}

We have opened up for a different model where you chose to have one APL thread pr active connection and on APL thread for the server receiving new Connections and starting an APL thread to handle the communication on the new Connection. The requires ReadyStrategy 4 and is activated like this

DRC.SetProp ‘S1’ ‘ConnectionOnly’ 1

### Timeout/Close

We have changed the timeout error 100 to be an event and also error 1119, and a Root Property to ensable it has been created.

DRC.SetProp ‘.’ ‘EventMode’ 1

In eventmode error 1119 is returned as a close event and error 100 from DRC.Wait is returned as a timeout event

## Connections

### SendFile{Overhual.Dyalog|TestSendFile & TestSendFileBlk}

It is now possible to send the content of a file in Text Raw , and Blk mode

DRC.Send ‘C1’ (‘’ ‘filename’)

If the data is nested and two elements first is considered data and second a filename.

### SendComplete{Overhaul.dyalog|TestSendCOmpleteCMD&TestSendCompletText}

It is also possible to get an event when the send of a buffer is complete.

DRC.Send ‘C1’ ‘data’ 3

Will cause Conga to queue an event on the connection C1 when the data has completed sending. In command mode a sent event is overridden by the answer on a command, so if you have asked for a sent event and the answer already has arrived when you wait for event you will get the answer.

## New Protocol

We have added new protocols to Conga 3.0. {HTTPTest.dyalog | Run, TestBlockSizes}

### Http

I have implemented a http protocol to move the task of breaking up the data received into to logical protocol blocks. We can run the http protocol both as client and as server.

DRC.Clt ‘C1’ ‘addr’ port ‘Http’ 100000

DRC.Wait on an http connection will return the following events:

HTTPHeader, HTTPBody, HTTPChunk, HTTPTrailer

Data in the events receive is not decoded in any form.

To send data on an http connection you also have to compose valid http messages. The only help Conga is providing is when you sent a file in that case Conga will add the Content-Length to the headers with the right value.

An http Connection can be upgraded to a bi-directional websocket connection. If the WSFeatures are set to 1 for an http Server, it can do the update without intervention, and the server will just be notified that the connection is upgraded with a WSUpgrade. If WSFeatures is not set to 1 the server will get an WSUpgradeReq and has to answer it by Compose an answer and set it as a property (WSAccept ) on the connection.

It looks like most browsers will not upgrade an existing connection but make a new one and upgrade that immediately.

### Websockets{Overhaul.dyalog | TestWebsocket }

When an http connection has been upgraded to websockets, the connection will return events on the form:

WSReceive

Sending data on a websocket connection can be done in two forms:

DRC.Send ‘WS’ (data fin [opcode]) ⍝ where opcode can be 0 Cont 1 Text 2 Bin

DRC.Send ‘WS’ <rawformated websocket message> ⍝ not recommended

DRC.Wait returns the websocket event in the same format event WSReceive (data fin opcode)

### UDP {Overhaul.dyalog|TestUDPSimple}

Not completed yet.