

Clarion Magazine

INVEST
in your own abilities

Clarion
magazine

Please note that Clarion Magazine is temporarily publishing three issues per month rather than four. All subscriptions are automatically extended. Meanwhile the elves are hard at work. Stay tuned for a special announcement later this month...



Data Structures and Algorithms Part XXIII - Dijkstra's Shortest Path Algorithm

Articles: **XML**

News: **XML**

Have you ever needed to work out the shortest path between two locations? Alison Neal shows how to do this in Clarion, using Dijkstra's Shortest Path Algorithm

Posted Monday, October 06, 2003

Clarion 6 Candidate Release 5

Clarion 6 Candidate Release 5 is now available to EA program participants.

Posted Monday, October 06, 2003

Product Review: NetTalk, Part 2

Over the years, Tom Hebenstreit has used various methods to add Internet functionality to his programs, ranging from the built-in Windows WinInet libraries to the SocketTool libraries from Catalyst Corporation. Each had its strengths and, as usual, some weaknesses as well, so Tommy's always on the lookout for simpler, more Clarion-friendly ways of accomplishing those tasks. Enter CapeSoft's NetTalk, a veritable jack of all Internet trades. Part 2 of 2.

Posted Tuesday, October 07, 2003

Weekly PDF for October 5-11, 2003

All articles for October 4-11, 2003 in PDF format.

Posted Wednesday, October 15, 2003



News

[New Download: Parser Class](#)

[CapeSoft C6 Product Status](#)

[Wizard Templates CR6
Compatible](#)

[CPCS C8 Beta For CR-6](#)

[xRuntimeStyle Manager v1.5](#)

[Product Scope 32 PRO
Version 5 Gold](#)

[EasyListPrint 1.06](#)

[Gitano Software Third-party
Vendor Of The Week!](#)

[MAV Direct ODBC 0.01](#)

SURVEY

How often do you purchase products or services online, by credit card?

Frequently

55.9%

Occasionally

26.5%

Only when absolutely necessary

14.7%

Never

2.9%

68 responses

[Previous Surveys](#)

[Clarion 6 First Look: The Examples](#)

There are two ways to get to know a new release of Clarion. One is to read the release notes (always a good idea). The other way is to do some independent investigation. Dave Harms begins a tour of Clarion 6, from the directories on up.

Posted Thursday, October 16, 2003

[Trees, Recursion, and Many to Many Checkboxes](#)

Most of the data you deal with probably fits nicely into a parent/child, relational data structure. But sometimes you have data that works better as a self-join, where data in one table relates back to other data in the same table. Randy Long shows how to use trees, recursion, and many-to-many checkboxes to manage this kind of data.

Posted Friday, October 17, 2003

[Clarion 6 Candidate Release 6 Available](#)

Gold release candidate 6 is now available to Clarion 6 EA program participants.

Posted Tuesday, October 21, 2003

[Weekly PDF for October 12-18, 2003](#)

All articles for October 12-18, 2003 in PDF format.

Posted Wednesday, October 22, 2003

[Clarion 6 First Look: The Source Code](#)

In Part 2 of this series Dave Harms examines the changes in the libsrc directory in Clarion 6, including a number of modified and new classes.

Posted Thursday, October 30, 2003

[Data Structures and Algorithms Part XXIV - Floyd's All Pairs Algorithm](#)

Alison Neal discuss Floyd's All Pairs algorithm, which provides the shortest path between all pairs of nodes in a graph.

Posted Thursday, October 30, 2003

[CR6 Installs For RPM, AFE & PNet](#)

[BoTpl Version 3.2](#)

[How NASA Writes Code](#)

[ClarionFoundry Open To Editors](#)

[xQuickFilter v2.13](#)

[Clarion Programmers By Country](#)

[Clarion Essentials Course For Cape Town](#)

[xRuntimeStyle Manager v1.4](#)

[CPCS Support Unavailable Oct 14-19, 2003](#)

[xTipOfDay 1.9](#)

[SoftVelocity Newsgroup Renamed](#)

[Solid.Software Third-party Vendor Of The Week](#)

[Clarion Developers Challenge Week 6 Winner](#)

[EasyResizeAndSplit 2.01](#)

[EasyExcel 3.02](#)

[EasyAutoEntry 1.01](#)

[Freeware xFunction Library](#)

One Year Ago In CM

[PDF for October, 2002](#)

[Converting The Inventory Example - Calling Stored Procedures \(Part Two\)](#)

[Converting The Inventory Example - Calling Stored Procedures \(Part One\)](#)

[Receiving Email With MAPI \(Part 2\)](#)

[ContainsMatch A to Z](#)

[Data Structures And Algorithms Part X - Going Out Of Your Tree?](#)

[Receiving Email With MAPI \(Part 1\)](#)

[Clarion Challenge Results: ContainsMatch](#)

[Not Having Six Makes Me Cranky](#)

Two Years Ago In CM

[Optimizing DLL Loading - Introduction to Rebasing](#)

[Controlling Printers](#)

[The Cape Town Clarion Essentials Training Course](#)

Danie de Beer reports on Raymond Dummer's Clarion Essentials training course held in Cape Town, South Africa.

Posted Thursday, October 30, 2003

[Book Review: .NET Framework Essentials](#)

.NET represents the first wave of Microsoft's Windows new software development and deployment strategy. Dave Harms reviews a book that provides some basic .NET information.

Posted Friday, October 31, 2003

Looking for more? Check out the [site index](#), or [search the back issues](#). This site now contains more than 700 articles and a total of over a million words of Clarion-related information.

[1.8 C6 Compatibility](#)

[xRuntimeStyle Manager 1.3](#)

[SoftVelocity To Product IP-Based Driver Layer](#)

[SuperTemplates And Clarion 6](#)

[AFE6 for CR5\(ABC\)](#)

[gFileFind Update](#)

[Developers Design Package Special](#)

[New Layout Manager](#)

[Free ErrorClass Template](#)

[comp.lang.clarion Back On SV News Server](#)

[SoftVelocity News Server Update](#)

[CR5 RPM & PNet Installs Available](#)

[SetupBuilder 5 Compiler Output Test](#)

[CPCS C6 Beta For CR-5 Released](#)

[EasyExcel 3.01](#)

[QandA 5 Minute Forum Released](#)

[Clarionfoundry at](#)

[With DevMode \(Part 2\)](#)

[The Clarion Magazine Free Sampler](#)

[Large Table Performance in MySQL](#)

[The Clarion Advisor: Creating A SOAP Client](#)

[New Advertising Rates - Special Deals!](#)

[Clarion Developers Conference 2001 Latin America](#)

[Controlling Printers With DevMode \(Part 1\)](#)

[Weekly PDF for October 14-20, 2001](#)

Three Years Ago In CM

[Give It a Nudge: Adjusting Report Position at Runtime](#)

[Clarion News - October 2000](#)

[The Clarion Advisor: Copying Browse Boxes Between Procedures](#)

[ClarionPublisher.com](#)

[Informatic.Consulting Named
Third-Party Vendor Of The
Week](#)

[Search the news archive](#)

[Web Development
Options: An Overview -
Part 2](#)

[Feature Interview:
Mark Riffey](#)

[Web Development
Options: An Overview -
Part 1](#)

[An Introduction To
Writing Templates:
Part 2](#)

[An Introduction To
Writing Templates:
Part 1](#)

[Template Writing
Made Easier: The
Template Wizard](#)

**Four Years Ago In
CM**

[The Other Way To Use
OLE - Part 3](#)

[Clarion 5.5 Preview](#)

[Stephen Mull's Guide
To Converting To MS-
SQL](#)

[David Bayliss On The
RelationManager - Part
2](#)

[The Novice's Corner:
Understanding Clarion
Code](#)

[The Clarion Advisor:
Detecting Duplicate
Records](#)

[Clarion News -
October 1999](#)

[Product Review:
Clarion Source Search](#)

[Open Source Update](#)

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

Reborn Free**CLARION**
online

Data Structures and Algorithms Part XXIII - Dijkstra's Shortest Path Algorithm

by Alison Neal

Published 2003-10-06

In my [previous article](#) I discussed a Critical Path Analysis algorithm implemented using a graph. The algorithm was of particular use for project management requirements. In this article I will continue my discussion on graphs and algorithms, in particular Dijkstra's Shortest Path Algorithm.

The purpose of Dijkstra's algorithm is to calculate the shortest path between two specific nodes in a graph. This algorithm uses a "greedy" approach, as it searches all possible alternatives within the graph; this can be slow when used with extremely large graphs. The example I will use is a map with weighted edges, indicating the distance from one location to another.

Here is the test file that has been supplied with the sample code (available at the end of this article):

Node1	Node2	Weight
Cgo	LV	1780
Cgo	Mmi	1423
Cgo	NO	948
Cgo	Sea	2060
Ds	LA	1440
Ds	Min	949

Ds	NO	517
Ds	NYC	1614
LA	Ds	1440
LA	LV	272
LA	SF	414
LV	Cgo	1780
LV	LA	272
Mmi	Cgo	1423
Min	Ds	949
Min	NYC	1217
Min	OkC	792
Mwk	Px	1771
Mwk	SF	2257
Mwk	WDC	811
NO	Cgo	948
NO	Ds	517
NYC	Ds	1614

NYC	Min	1217
NYC	WDC	237
OkC	Min	792
Px	Mwk	1771
SF	LA	414
SF	Mwk	2257
SF	Sea	808
Sea	SF	808
Sea	Cgo	2060
WDC	Mwk	811
WDC	NYC	237

Fundamentally this test file depicts an inaccurate (in fact, entirely fictitious) map of the flight distances between various US cities. (If anyone wishes to supply the accurate figures I will be happy to update the source.) Dijkstra's algorithm makes it possible for me to calculate the shortest distance from one city to another, regardless of whether they are directly or indirectly linked.

The graph implementation itself has not really changed from the one I used in previous articles. It does however use a matrix (multidimensional array) of longs to store the edge node addresses. In this instance I have made the matrix static, and dimensioned to 100 x 100, rather than using a dynamic matrix of strings, as I did in my [previous article](#).

```
edgeData    CLASS , TYPE
fromNode    ULONG ( 0 )
toNode      ULONG ( 0 )
weight      ULONG ( 0 )
```



```

        END
GData      LONG(0),DIM(100),DIM(100)
EData      &EdgeData
nSize      LONG(0)
currNode   LONG(-1)
currEdge   LONG(-1),DIM(100)
NodeName   STRING(5),DIM(100)

```

An edge will contain the node numbers of the two nodes that it links together, as well as the weight or distance between those nodes. `nSize` maintains a count of how many nodes have been added to the graph, so that when I need to traverse the graph I don't have to waste processing time by looping through the entire 100 x 100 array. I just need to traverse from 1 to `nSize` positions in the array. The `currNode` variable contains the number of the node, which is currently being examined, and the `currEdge` variable keeps track of the edge numbers, which are currently being examined for each node. The `NodeName` array keeps a list of the names associated with each node added.

The methods used are:

Init	Initialize the Graph
Kill	Clean up
AddGraph	Pre-check for adding nodes to the graph
AddNode	Adds a node if the node hasn't already been added
Add Edge	Add an Edge to the Graph
NumNode	Return the number of nodes in the Graph
NodeNum	Return the node number for a given node name
NodeName	Return the node name for a given node number
First Node	Make the first node in the graph the current node

Next Node	Make the next node in the graph the current node
First Edge	Make the first edge for a given node the current edge for that node.
Next Edge	Make the next edge for a given node the current edge for that node
Dijkstra	Dijkstra's algorithm

The `Init` method doesn't do anything unexpected, it just sets `nSize` to 0 and `Edata` to `NULL`. The `Kill` method traverses the multidimensional array (`Gdata`) and disposes of the memory allocated for all edges.

`AddGraph` adds two nodes to the graph, for each one; it checks first to make sure the node doesn't already exist. Then the edge is added in the appropriate matrix position, dependent on the node numbers that the edge relates to.

```
graph.AddGraph  PROCEDURE(*STRING Node1, *STRING Node2, ULONG Weight)
  CODE
  IF SELF.nodeNum(Node1) < 0 THEN SELF.addNode(Node1).
  IF SELF.nodeNum(Node2) < 0 THEN SELF.addNode(Node2).
  SELF.addEdge(Node1,Node2,Weight)
```

```
graph.addNode  PROCEDURE(*STRING S)
```

```
  i  LONG(0)
  CODE
  ? ASSERT(SELF.nodeNum(s) < 0)
  SELF.nSize += 1
  SELF.nodeName[SELF.nSize] = S
```

```
graph.addEdge  PROCEDURE(*STRING S, *STRING T, LONG W)
```

```
  i  LONG
  j  LONG

  CODE
  i = SELF.nodeNum(S)
  j = SELF.nodeNum(T)
  ? ASSERT(i > 0)
  ? ASSERT(j > 0)
```

```
SELF.addEdge(i, j, W)
```

```
graph.addEdge PROCEDURE(LONG f, LONG t, LONG W)
```

```
CODE
SELF.Edata &= NEW(EdgeData)
? ASSERT(~SELF.EData &= NULL)
SELF.EData.fromNode = f
SELF.EData.toNode = t
SELF.EData.weight = w

SELF.GData[f][t] = ADDRESS(SELF.EData)
SELF.CurrEdge[f] = t
```

This graph implementation is very similar to one I've [already discussed](#), except for the use of the matrix.

Dijkstra's Algorithm

Here's the code for Dijkstra's Algorithm, which searches for the shortest path between two nodes:

```
graph.Dijkstra PROCEDURE(*STRING startNode, *STRING endNode)
```

```
INT_MAX EQUATE(2147483647)
D ULONG(0)
F ULONG(0)
T ULONG(0)
X ULONG(0)
paths LONG(0), DIM(SELF.nSize)
dist ULONG(INT_MAX), DIM(SELF.nSize)
E &EdgeData
edgeAddress LONG(0)
CODE
IF exp.dosopen('export.txt', 1) THEN RETURN -1.
IF exp.setpointer(0, 0) THEN RETURN -1.
SELF.expFile('Find Shortest Path From ' &startNode |
  &' To ' &endNode &'<13,10><13,10>')
X = SELF.nodeNum(startNode)
F = X
T = SELF.nodeNum(endNode)
dist[F] = 0
paths[F] = -1
PQ.Init()
LOOP WHILE x <> T
  IF SELF.FirstEdge(x) > 0
    E &= (SELF.FirstEdge(x))
    LOOP WHILE E.fromNode <> -1
```

```

D = Dist[x] + E.Weight
IF D < Dist[E.ToNode]
  IF PQ.isMember(E.toNode) = TRUE
    PQ.SetWeight(E.toNode, d)
  ELSE
    PQ.insertQ(E.toNode, d)
  END
  dist[E.toNode] = d
  paths[E.toNode] = x
END
edgeAddress = SELF.nextEdge(x)
IF edgeAddress > 0
  E &= (edgeAddress)
ELSE
  BREAK
END
END
END
PQ.remQ(x,d)
END
SELF.expFile('Shortest from is ' &d &' miles <13,10>')
SELF.expFile('Route: ')
SELF.printPath(paths, t)
PQ.Kill()
exp.dosclose()

```

The algorithm works outward from the starting node, analysing each of the edges that lead from it, then analysing each of those, cumulatively calculating a total distance for each possible path as it progresses. As I mentioned at the beginning of the article, this is known as a greedy approach, since the algorithm searches all possible paths.

Variable D is used to calculate the accumulated distance of a given path. F holds the node number for my starting node, the node that I want to leave from. T holds the node number for my target node, the node that I want to go to. X holds the current node number, as the loop traverses through the graph, going from node to node, it calculates the shortest route. By the finish of the algorithm, the paths array will contain a list of the nodes traversed in the shortest route calculation. The dist or distance array is used to calculate and compare the competing distances.

So, lets go.... from DS to CGO.

When the graph is loaded DS is node 6, so F, the starting node variable, and X are initialized to 6. CGO is node 1, so T, my target variable, is initialized to 1. Position 6 (F) in the dist (distances) array is initialized to 0, and position 6 in the paths array is initialized to -1. The starting node (6) doesn't equal the target node so the loop iterates for the first time. The first edge from DS goes to NO (Node 4), and this has a weight of 517, which is assigned to D (517 + 0). The value 517 is less than INT_MAX, so the node that this edge joins NO (4) is added to the [priority queue](#) with its associated weight. The distance and path arrays are then updated with the relevant weight (distance D) and node numbers (6).

The next edge is then analyzed in the embedded loop. This edge runs from DS to LA (node 7) and has an associated weight of 1440 miles. D is assigned the accumulated distance for node 6, which is 1440 for this edge. Note that `dist[x]` has not been updated to date so it is still zero. The value 1440 is less than `INT_MAX`, so a second node is added to the priority queue. The priority queue is ordered on the least distance, making sure that I always examine the lowest cumulative distance first. The queue now looks like this (4,517)->(7,1440). The distance to position 7 in the distance array is then updated with 1440, and the paths array is also updated with the node number 6 (the starting node).

Another edge is analyzed, and the same process continues; this edge is DS to NYC with a weight of 1614, and it is also added to the priority queue as node 9, giving a queue of (4,517)->(7,1440)->(9,1614).

Once all the immediate edges have been analyzed, the algorithm then gets the first node off of the queue, in this example that would be node 4, which had a leading edge weight of 517. All the edges from this node are now analyzed. In this case that includes the edge NO (Node 4) to Chicago (Node 6). While Chicago has been located in the graph, the algorithm will not terminate until after Chicago has been added to the queue, and then removed, thus updating `x` (the reference variable) to equal `t`, and giving me the shortest distance calculated so far for the trip. It is possible for the algorithm to identify more than one path to Chicago, but remember that the Priority Queue is ordered by lowest weight. So if there are two competing paths, the first one returned by the priority queue is always going to be the shortest and therefore the one that I'm looking for.

Summary

Dijkstra's algorithm can be extremely useful in some applications. Being a greedy approach however, on a very large graph it's going to take some time to process.

In my next article I'll be taking a look at Floyd's all-pairs algorithm, which finds the shortest path between all nodes in the graph. Rather than looking for a path between two particular nodes like Dijkstra's algorithm, Floyd's algorithm gets the shortest path between *all* nodes. This means it is possible to only run the algorithm whenever the graph itself changes, rather than every time a distance is required. In the case of distances between cities on a map, one would hope that they don't change very often, so Floyd's algorithm maybe a better solution for that particular application.

[Download the source](#)

Alison Neal has been using Clarion since 2000, whilst working for [Asset Information Systems](#) (AIS) in Auckland, New Zealand. Some years ago (at the tender age of 19) Alison graduated from the Central Institute of Technology in Wellington, New Zealand with a major in Cobol. She also has a BA in English literature and has studied Computer Science, Philosophy and Information Systems. AIS is an independent division of Asset Forestry Ltd, and has a team of five programmers developing almost exclusively in Clarion. AIS also offers web (ClarionNET) and email services for the customer who needs everything. The company has many and varied customers bridging across a wide range of industries including Telecommunications, Forestry & Agriculture, Manufacturers, Military & Government, Legal & Financial, and Retail.

Reader Comments

[Add a comment](#)

Alison, excellent article and great timing since I am...

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

Clarion Magazine

Reborn Free

CLARION
online

Product Review: NetTalk, Part 2

Published 2003-10-07

Welcome to Part 2 of the CapeSoft NetTalk review. In this half of the review I will cover the NetTalk features that use the CapeSoft NetTalk protocol, as well as touch on using NetTalk with Windows Dial-up Networking. I'll also cover documentation and support before I wrap it all up.

By the way, if you haven't read [Part 1](#) yet, I suggest you do so now since this part builds on the topics I already discussed there.

NetTalk Objects Overview

NetTalk Objects are something that seem complicated when you first come across them, then they are simple until you start to use them, then more complicated again until your "Aha!" moment, then you wonder how you ever did any network applications programming without them. I'll do my best here to give a synopsis of what they are, but I also recommend checking out the [CapeSoft website](#) as they have a ton of good information there.

Unlike the NetSimple Objects I covered in Part 1 that use standard Internet communications protocols to talk to other machines, NetTalk Objects use their own special NetTalk protocol to communicate with each other. NetTalk itself still uses regular old TCP (or UDP) to connect and move the packets around, but the protocol on top of that is specific to NetTalk. These NetTalk protocol connections can be across Local Area Networks (LAN), Wide Area Networks (WAN) and yes, across the Internet as well (with a few caveats and some extra effort, covered later).

The core pieces of the NetTalk Objects are the NetClient object and the NetServer object. In a nutshell, NetClient objects initiate requests and NetServer objects respond to requests. Your applications can contain either or both NetServer and NetClient objects, depending on the type of functionality you need. For example, a Time Server application would simply wait for requests from clients to send the current time. It could not initiate any actions on its own, and conversely, the clients could not themselves respond to requests for the time. Something like chat functionality, on the other hand, requires that your application be able to be both a client (connecting and sending text to other machines) and a server (accepting connections and text from other machines).

No matter which type of object it is, the basic point is that the machines can establish a connection and pass packets of information between them using the NetTalk protocol. NetTalk handles all of the low level communications work of connecting, synchronizing, ensuring error free transmissions, disconnecting and so on, while it is up to you to fill those packets with whatever type of information you want to exchange with the other program. You can also optionally have NetTalk automatically encrypt/decrypt your packets using 168-bit DES encryption.

If you wish, you can have NetServer objects automatically announce themselves on the local network as soon as the particular service (i.e., a function such as chat) is started within your program, sending what CapeSoft describes as an "I can do" message to all connected local machines telling them what public (non-private) services it offers. For example, when you open a chat window in one instance of your program, all other instances that have chat open will immediately see you appear in their list of "chat-able" machines (for lack of a better term). In the same way, they all automatically appear in your own chat list. When you close the chat window on your machine, you disappear from everyone else's chat window. Internally, NetTalk is keeping track of any and/or all other public NetTalk servers/services in queues so that it constantly knows who is available and for what purpose.

One final thing to be aware of is how the NetTalk protocol interacts with your proxy servers and firewalls. Communications between your NetTalk protocol applications within your own LAN should be transparent and effortless, since all of the machines are connected to the same subnet and are behind your firewall (one would hope!) In order for NetTalk Objects to communicate outside of your LAN, two things need to happen. First, they must be made aware of each other, since the "I can do" messages are only broadcast locally. (CapeSoft provides two different methods of doing this, and I'll touch on them a bit further on when I talk about NetRemote and NetDIP.) Second, the ports used by the NetTalk protocol must be open in your firewall, or else the outside applications won't be able to establish a connection with those behind the firewall. Once a connection is established, though, remote NetTalk protocol machines are treated just the same as any other local machine.

Using NetTalk Objects

NetTalk (the product) includes several templates to make it easy to implement some of the common uses of the NetTalk Objects. These include:

- **Time Client and Server** - Use these templates to allow one Time Server program/machine to provide date and time services to multiple clients.
- **File Client and Server** - Request and transfer files from one machine to another, without having to bother with setting up FTP servers, or mapping and sharing drives.
- **NetRefresh** - Use this global extension to have all of your browses and forms automatically refresh every time data changes in another instance of your program. Windows that are on inactive threads will also refresh themselves as soon as they become active. This is very handy when you need to ensure that every workstation is displaying the most up-to-date information. Template options include specifying the application name (used to differentiate this program from other NetRefresh enabled

programs), minimum time between refreshes (to avoid having the windows constantly refresh in a heavily used application) and for auto-sending a refresh every time a form is saved or a process completes. If you want, you can trigger a refresh manually in embed code as well. NetRefresh can also be enabled or disabled on a procedure-by-procedure basis using the templates.

- **NetChat** - The chat control templates allow you to add interactive chat to your applications. This is a shared, broadcast type of chat similar to using IRC or the chat component of SoftVelocity Discussions, as opposed to a one-to-one instant messaging type of chat such as ICQ or AOL Instant Messenger. In other words, everyone connected to the chat can see what all other users are saying and join in the discussion.
- **CloseApp** - This allows you to send messages to and remotely shut down all or selected instances of your program on your network. For example, if you needed to do maintenance on your database you could warn all users to exit the program, and then also be able to shutdown any remaining instances remotely (say, if those users had gone home for the day.) As usual, you need to specify the name of the program that will respond to these particular messages and shutdown requests via the template. Remember, you could have many different NetTalk-enabled applications running on your network; for example, you wouldn't want your payroll applications to shut down when you send a shutdown for a tech support application.
- **UseRemoteMachine** and **NetDIP** - These control templates help your NetTalk protocol applications move beyond the boundaries of your local network (LAN). UseRemoteMachine is useful for connecting to external machines with fixed IP addresses. NetDIP, on the other hand, lets you set up a common external machine with a fixed address that other external NetTalk programs can connect to and register themselves. This allows machines with dynamic IP addresses (e.g., connecting through modems so that their Internet IP address is never the same) to make their presence known to local NetTalk applications. Once registered, the external NetTalk programs are treated the same as if they are on the local LAN, i.e., they can send/receive "I can do" broadcasts, use NetTalk services, etc., just like anyone else. NetDIP also has the ability to group connections by what CapeSoft calls "rooms" so that you can have multiple programs using the same NetDIP server, with each only seeing other instances of the same program (i.e., other "users" in the same "room".) You specify the room-name in the templates when you first set up NetDIP in your individual applications.

The provided templates only hint at the flexibility that NetTalk Objects give you for creating networked applications. For example, say I wanted to implement a one-on-one style of messaging rather than the broadcast style of chat. As usual, the first thing I do is look at the NetTalk demo and sample applications. As it turns out, the NetClient and NetServer demos do pretty much just what I want already, showing how to connect to specific machines and transfer text back and forth.

The most important thing to understand about using the NetTalk protocol is that you are certainly not limited to text - you can send any type of binary information you want (data records, images, files, commands, whatever.) NetTalk provides the infrastructure, but it is up to you to decide how to use it. The possibilities are limited only by your own imagination.

NetDUN (Dial-up Networking)

Most of the other NetTalk functions I have discussed prior to this assume that the machine already has an active network connection of some kind. But what do your users do if they are using a laptop, or using a modem? Meet your new pal, NetDUN. This is basically a wrapper around the Dial-Up Networking functions that are built into Windows, but it is really handy to have them encapsulated in an easy to use and Clarion-friendly format. Dial-Up Networking is what Windows uses to connect to the Internet, or to another remote network, via a modem. A control template included with NetDUN lets the user view a list of the currently defined DUN entries and select one to use for the connection. You can also use the NetDUN object methods to manually retrieve the list, dial, hang up, create new entries, delete entries and more.

Additionally, NetDUN lets your application check for an existing connection and use it, rather than creating a new connection. This is useful if another program has already dialed out prior to your own being activated.

What's Missing?

As you've seen, NetTalk provides many, many useful features, but there is one area that it doesn't cover: the ability to use secure web, email and FTP protocols (e.g., HTTPS, SMTPS and FTPS/SFTP). When I asked about them, CapeSoft indicated that at least some of those protocols are on the wish list for future versions, but also that they are not a priority at the moment. If you are required to use one of those secure protocols for your connections you will need to look into using the built-in [Windows WinInet](#) libraries or another third-party product like the Secure Edition of the [Catalyst Development SocketTools](#). Neither solution is even remotely as easy to use as NetTalk, though, so I guess I'll just have to keep whining at CapeSoft and hoping for the future.

Documentation

CapeSoft uses HTML files for their online documentation, so viewing the information requires a web browser. The files are beautifully laid out, easy to read and easy to navigate. Good use is made of popup menus and other browser features, but that also means you will need a reasonably modern browser to take full advantage of them. I tested with both IE 6 and Mozilla Firebird 6.1 and had no problems whatsoever.

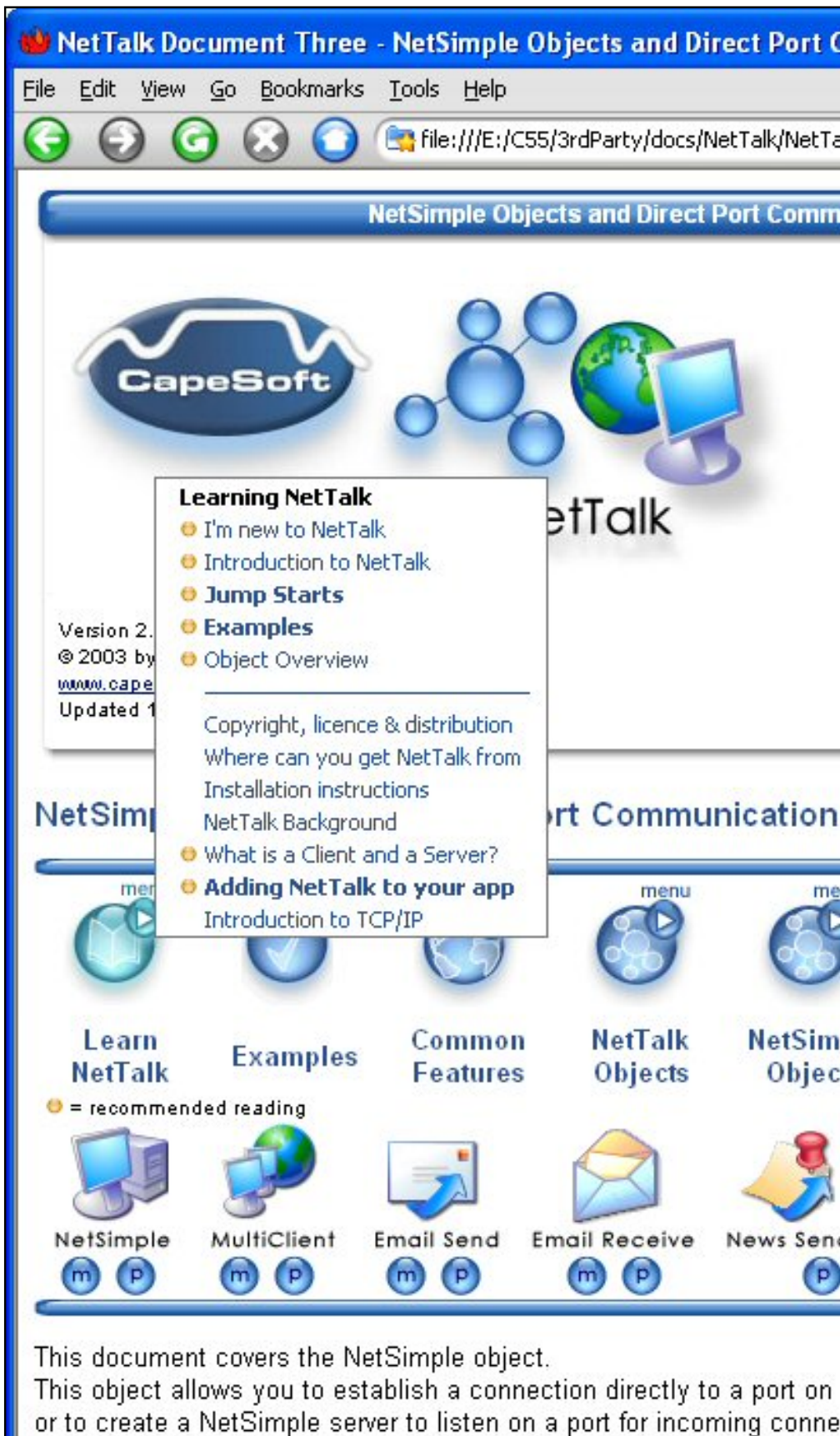


Figure 1. An example of the CapeSoft NetTalk documentation

If printed, the documentation comes to well over 200 pages. I personally found the online version to be perfectly acceptable, and it was easy to cut and paste code snippets.

In addition to the written documentation, NetTalk comes with twenty-six folders of useful demo apps, ranging from simple single feature jump starts to the full-blown comprehensive NetTalk features demo. The majority are ABC based, but a number of Clarion template demos are also included. As I mentioned before, I found it very easy to pick a function, look at the demo and in many cases just cut and paste code directly into my own applications.



Figure 2. The main NetTalk demo application

You can download and try out a compiled version of the NetTalk demo shown in Figure 2 to get a feel for all of the things NetTalk can do. Put it on a couple of machines on your network and have some fun. One word of advice - if you try to use some of the NetTalk protocol functions across the Internet without heeding CapeSoft's admonitions about firewalls, etc., the demo may appear to freeze and flicker for a bit as it tries unsuccessfully to establish or drop a connection. To be fair, they do put the warnings in bright red letters on those windows.

Technical Support

CapeSoft has well-deserved reputation for excellent support, and my experience regarding NetTalk confirmed my own high opinion of the company. Questions that I emailed to the support address were answered quickly and completely within twenty-four hours - quite acceptable seeing as how they are in South Africa and I'm in California. CapeSoft is actively developing and adding new features to NetTalk so, if needed, bug fixes are also available on a pretty regular basis.

Summary


As you can imagine, there is a whole lot going on under the covers in NetTalk. The beauty of NetTalk is that you don't have to worry about all of those sockets, packets, datagrams and

protocols. CapeSoft has wrapped classes and/or templates around most of the commonly needed functions to simplify their use, and the classes themselves are written in a way that gives a clear and consistent feel to the overall product. Once you've mastered one area of NetTalk, you'll find that the next one is already familiar.

As I said, I suspect that many people will be drawn to NetTalk initially by the basic email, FTP and web functions. Don't stop there, though, as NetTalk can do a lot more than that using the NetTalk protocol functions. The ability to have your programs aware of each instance, and able to dynamically talk and share information securely with each other on a real-time basis, is really a mind-bending option.

Bottom line: The combination of easy to use Internet functions and highly flexible network functions make NetTalk *the* tool to use if you need to have your programs communicate with other machines or processes (and who doesn't nowadays?) It is one of those wonderful products that is easy to get started with, but also has the depth to cover almost any networking need that you will come across in the future.

I *like* it!

Overall Product Rating: 	
Ability to do the task	Excellent
Ease of use	Very Good
Ease of Installation	Excellent
Documentation	Excellent
Technical Support	Excellent
Black-Box DLLs/LIBs	Yes

CapeSoft NetTalk is available online at <http://www.clarionshop.com>. At the time of this review, the price was \$299 US. For more information or to download the NetTalk demo, please visit <http://www.capesoft.com>.

Reader Comments

[Add a comment](#)

Tom, I really appreciate you and Dave bringing this...

Tom: Thanks this is a timely reveiw. I need to create...

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

Clarion Magazine



Clarion 6 First Look: The Examples

by David Harms

Published 2003-10-16

Every major release of Clarion brings a number of important new features, some expected, some unexpected. Clarion 6 is no exception, and in fact the list of new features is massively long. They include, per the [list](#) at the SoftVelocity web site: pre-emptive threading (with supporting synchronization classes), BLOB support in all SQL drivers, XML support (choice of native Clarion classes or Clarion wrapper classes for the CenterPoint C++ XML library), IDE enhancements (including smart locators, zoom window for entry fields, single module generation option, new dialogs, and lots more), XP manifest support, client side triggers, business rules, VCR form navigation, ADO support, business graphing, a number of new wizards (including one for labels) plus theme support, report output to XML and PDF, browse sorting by clickable headers, better print preview control, BIND interface in templates, greenbars, ability to use some ABC classes in the Clarion template chain, language and property syntax enhancements, and more.

There is one obvious omission: the IDE is not yet 32 bit. I seriously doubt this will be a deal breaker for anyone, however. If I had the choice between a 32 bit IDE right now and the goodies in C6, I'd happily take the latter. Of course it would be nice to have both, but clearly SoftVelocity plans to do more with a 32 bit IDE than just change the number of bits involved. And we all need at least one thing to complain about in anticipation of the next major release.

As an aside, one of the requests I do see in the newsgroups is for mouse wheel support in the IDE. You may want to try (at your own risk, of course) the [Flywheel mouse driver](#), which reportedly makes the mouse wheel work in the C6 IDE. Mouse wheel support is available in applications you create - in fact you can choose whether you want browses to scroll vertically or horizontally in response to the mouse wheel.

In this series of articles I'll be looking at some of the new features in Clarion 6. To kick things off, here's a view of Clarion you might not have seen before.

C6 under the hood

One of my favorite things to do with any new release of Clarion is to compare the new release's directory tree with the previous release. To do this, I use Scooter Software's [Beyond Compare](#). Figure 1 shows Beyond Compare displaying the top level Clarion directories in C5.5 and C6.

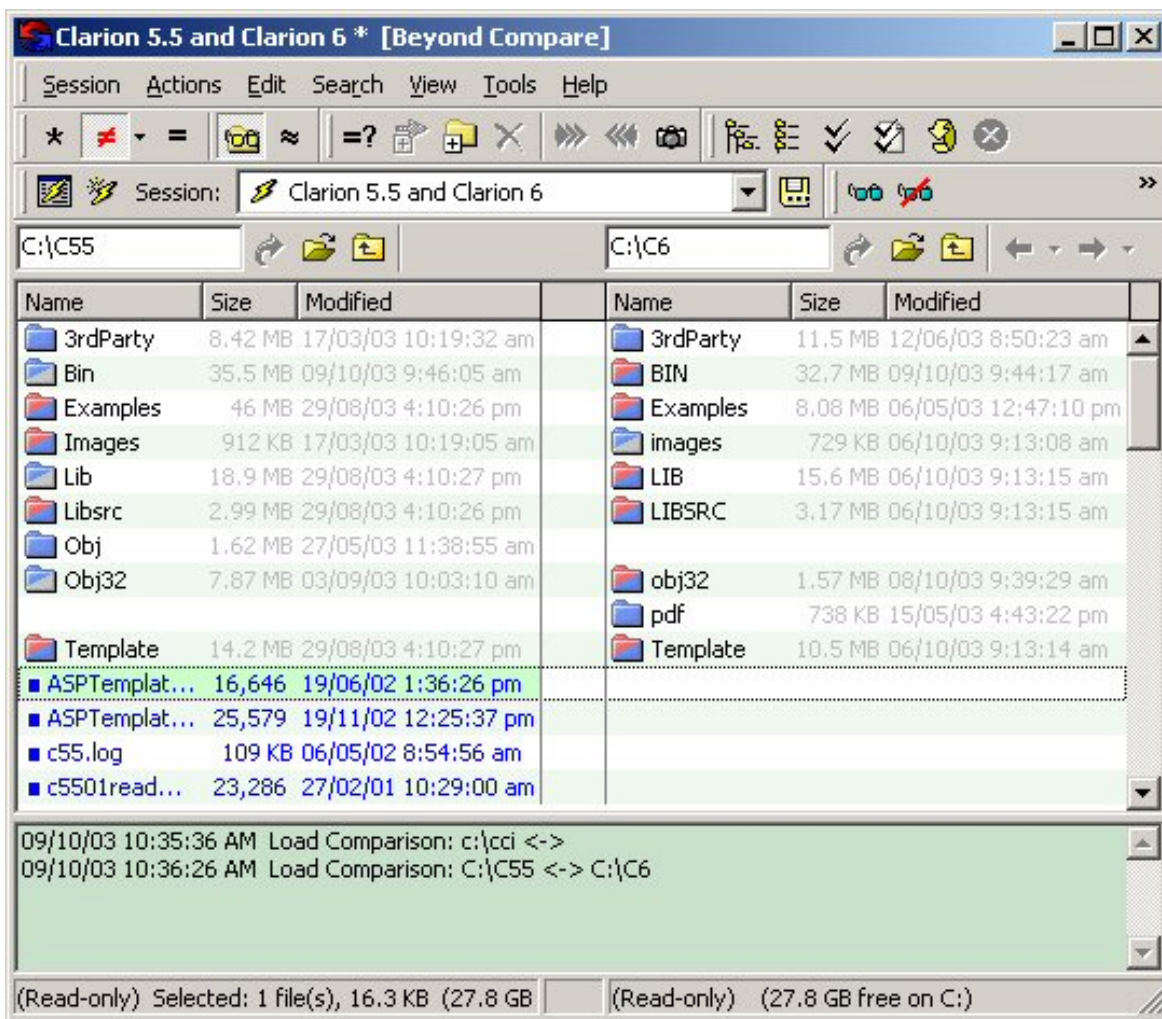


Figure 1. Comparing the C5.5 and C6 directory structures

There isn't that much to note at this level, other than the disappearance of the Obj directory. That's because C6 does not support 16 bit target applications.

The Bin directory

It's hard to get a good comparison of changes in the Bin directory because many of the files have different names (e.g. c55xxxxx becomes c60xxxxx) but there is at least one point to note. New in this directory is TOPMATCH.EXE, which appears to be the TPS equivalent of the old CPD CFIL conversion utility. But overall the bin directory isn't usually a very profitable place to go snooping.

The Examples directory

Now things begin to get interesting. There are a number of new example applications that showcase Clarion 6's features. I'll just provide a quick summary here - look for more detailed coverage in future issues of Clarion Magazine. You can also find a brief discussion of these examples in the C6 help file.

ADO - The Active Data Objects example demonstrates the use of C6's new ADO templates. A typical non-ADO Clarion application uses the database driver(s) and the ABC browse and form classes to model the application's data. ADO is another way to do this, using standard Microsoft technology. In ADO you open a connection to a database, and you issue a query. That query returns a *recordset* object, which is sort of like a view in that it represents the data you've selected. You can then call methods on that recordset to read and write data. The principle advantage of ADO is that you can work much more directly with SQL data than is possible using Clarion views and the ABC classes.

You do have the option of calling the ADO classes directly, or using the ADO templates. There are actually several ADO applications - one is an APP file, which demonstrates ADO features, and there are also two PRJs which show how to connect to an ADO database using hand code.

BizMath - This is actually the same as the old BusMath application, except it's undergone a minor facelift. I haven't examined the source to see if there are any significant changes, but other than the cosmetic changes it looks like the same application.

BizRules - This application demonstrates the use of the Clarion 6 business rules templates. You use a global extension template to define some basic conditions, such the name of the file which will store the definitions (default extension .BRF), and the icon to display beside an entry field when the corresponding rule is broken. Then you create the rules. Figure 2 shows the rule dialog. Clicking on the E button brings up the Expression Editor, which you can use to assist you in building valid Clarion expressions.

The dialog box for defining a rule contains the following elements:

- Rule Name:** City
- Rule Description:** You must enter a City
- Rule Definition:** Evaluation Expression: CLIP(CUS:City) > " E
- Field to link to:** CUS:City F
- Text:** If this expression evaluates to zero, the Rule is broken
- Text:** This Rule will be added to each Procedure where the field is populated as a Control or has been added into the Rules Hot Fields list.
- Checkbox:** Display an error indicator when Rule is broken
- Buttons:** OK, Cancel, Help
- Control List:** Controls

Figure 2. Defining a rule

You can override business rules at the procedure level. And it's easy to incorporate business rules into a multi-DLL application, as the global template lets you specify whether this application is an "original" or a "clone." If it's a clone, all you specify is the name of the business rules file. Here's a portion of the rules file from the BizRules example:

```

RULEFILENAME                ruledemo.brf

ALLRULEBASEDESCRIPTION      Procedure Rules
ERRORINDICATORIMAGE         BRuleNo.ico
RULEOFFSETRIGHT             3

VIEWRULESHEADERICON        BRules.ico
VIEWRULESVALIDRULEICON     BRuleOk.ico
VIEWRULESBROKENRULEICON    BRuleNo.ico

RULEBASES                   1
RULEBASENAME                 CustomerTable
RULEBASEDESCRIPTION          Rules for Customer table

RULES                        1
RULENAME                     City
RULEDESCRIPTION              You must enter a City

```

```

RULEEXPRESSION          CLIP(CUS:City) > ''
RULELINKFIELD           CUS:City
RULESHOWINDICATOR      1
  RULEENABLEDISABLECONTROL ?Ok
  RULEENABLEDISABLECONTROL ?OkButton
  RULEENABLEDISABLECONTROL ?Accept
  RULEENABLEDISABLECONTROL ?Save

RULEBASES                2
RULEBASENAME            OrderTable
RULEBASEDESCRIPTION     Rules for the Orders table

RULES                    1
RULENAME                OrderDate
RULEDESCRIPTION         Ordered date must equal to or greater than today
RULEEXPRESSION          fnCheckDate(ORD:Ordered,Action)
RULELINKFIELD           ORD:Ordered
RULESHOWINDICATOR      1
  RULEENABLEDISABLECONTROL ?Ok
  RULEENABLEDISABLECONTROL ?OkButton
  RULEENABLEDISABLECONTROL ?Accept
  RULEENABLEDISABLECONTROL ?Save

```

There are two applications in the BizRules directory, one for a DLL, the other for an EXE. Make the DLL first, then the EXE. I did it the other way around first, and then got link errors on the EXE after making the DLL. I did a Generate All, recompiled, and everything linked fine.

Crystal - The Crystal reports demo application has been renamed to Crystal6.app, but other than the required minimal changes to the Crystal template I can't see any differences.

DriverTrace - The DriverTrace application is the source code for the Clarion trace.exe utility, which you've probably used in the past to set debug/trace options for the Clarion database drivers.

Graph - The SVGExam application showcases the new business graphing control. You can populate this control on a window or a report, and you can also tie it to a browse control so that each time the browse record changes the graph display also changes. Graph types include scatter, line, area, floating area, column chart, column with accumulation, floating column, bar chart, bar with accumulation, floating bar, and pie chart. Figure 3 shows one example of a bar chart. The mouse tooltip (the yellow text) provides additional detail, and in the application you can double click on the graph to drill down to the underlying list box which holds the data for the graph.

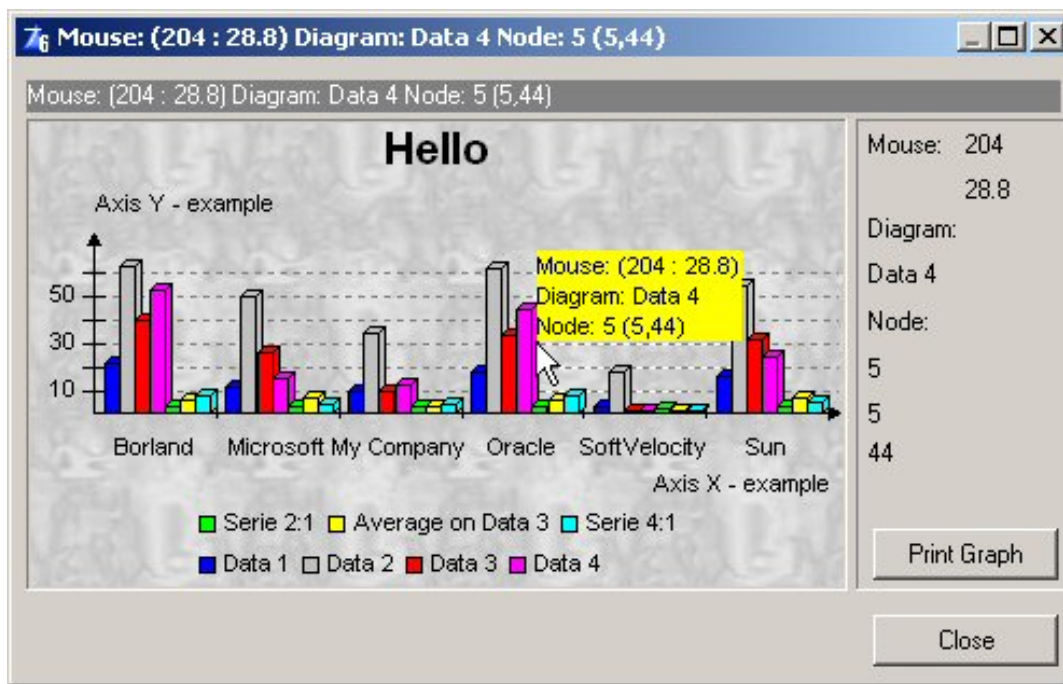


Figure 3. A graphing example

ReportOutput - This application demonstrates some of the new report output options in Clarion 6. Figure 4 shows the report output examples, with varying output options.

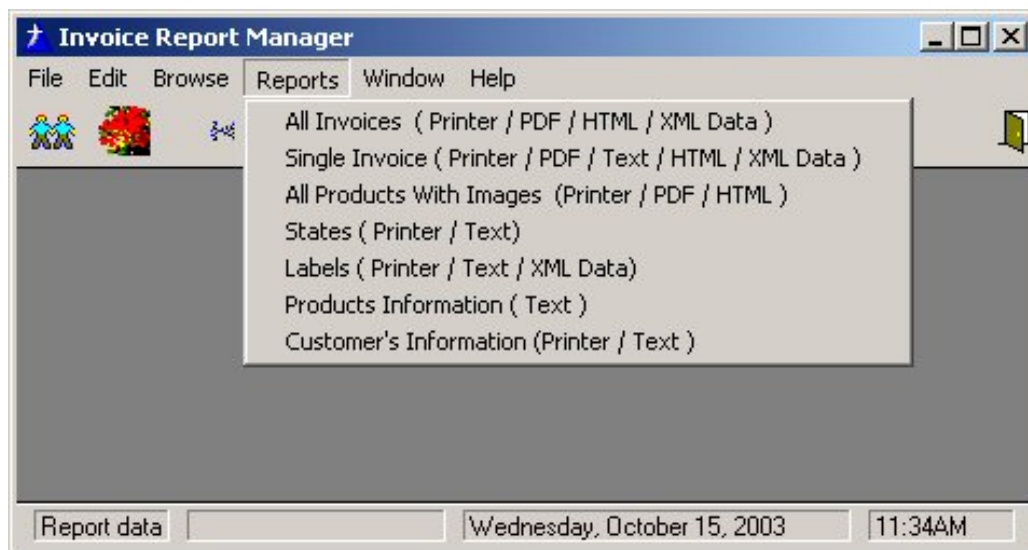


Figure 4. The example report output options

If you choose the Single Invoice option, you'll see the selection dialog in Figure 5. Click OK to get a preview of the report, and from there you can output to the chosen format.



Figure 5. Choosing an output option

All these output options work pretty much as you'd expect, although text output will of course not include images. XML doesn't include images either although presumably this could be achieved using CDATA sections (to what end I'm not sure). The HTML output comes complete with a page drop down navigation list, and First/Prior/Next/Last navigation buttons (all done with JavaScript), which is a nice touch.

RichEdit - This RTF example application is actually another PRJ, so this isn't a demo of the new RTF templates, which are intended to make RTF integration into an existing application a lot easier. On the other hand, the RTFTextControl (which replaces RTFControl) template is easy enough to use that there's no demo needed; just drop in the control, and then optionally drop in the RTFToolbar control (which appears in the control templates list after you've populated the RTFTextControl).

If you want to see how it all works in source code, however, go to the RichEdit example. Figure 6 shows the demo PRJ in action. One curious thing about this example is you need to click on the big RTF Control Test button at the bottom of the window to display the controls.

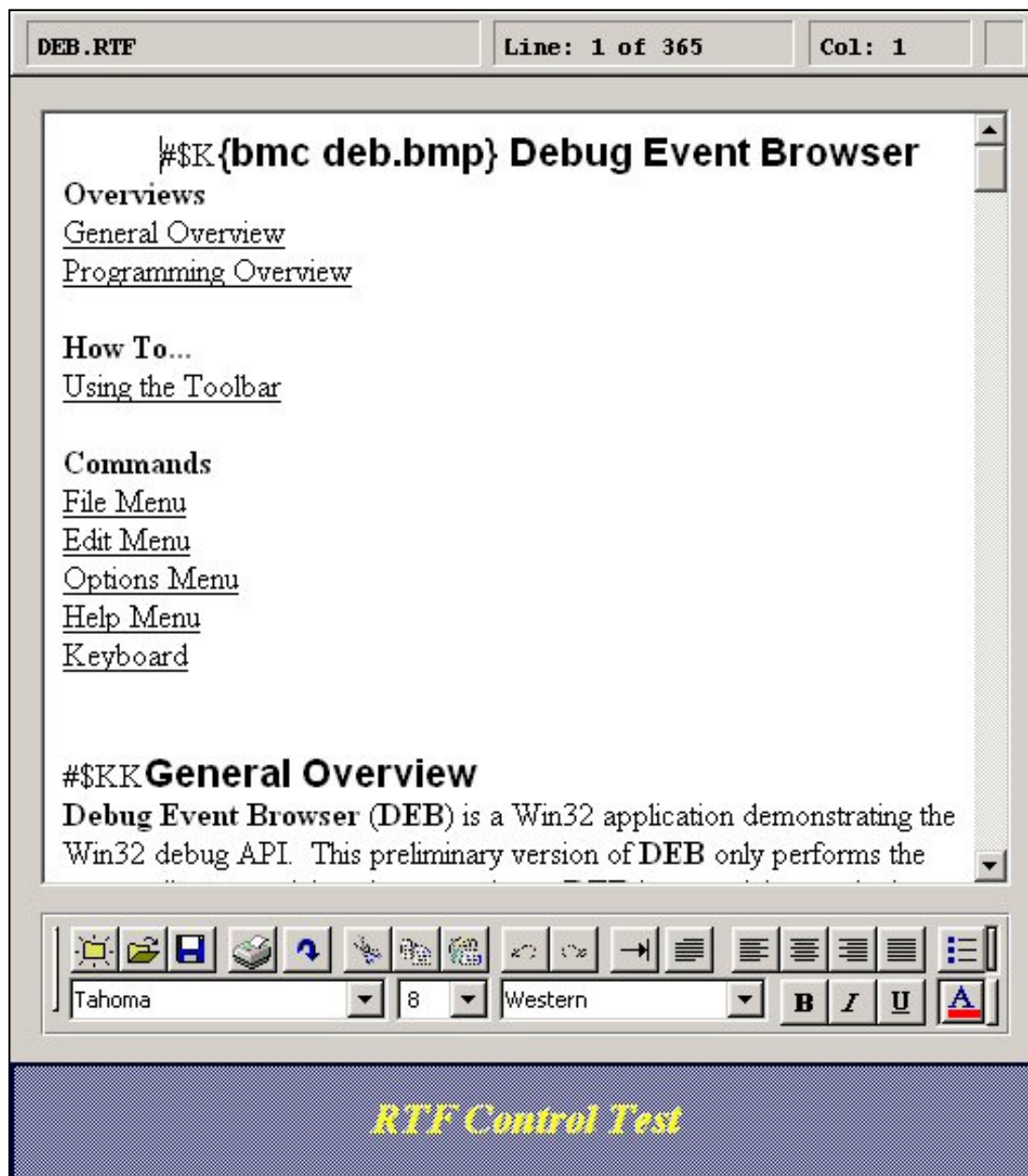
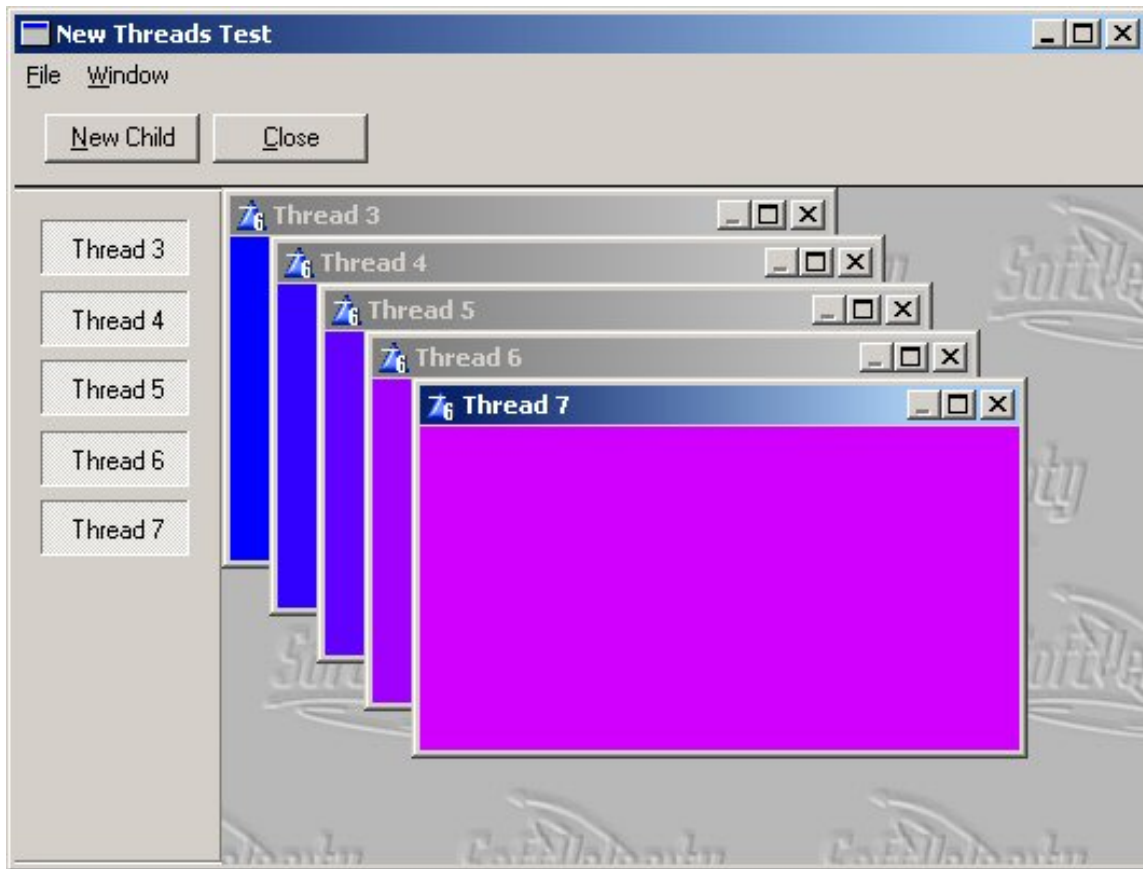


Figure 6. The hand coded RTF example

Threads - This is another PRJ that demonstrates the new threading model in C6. Each time you click on the New Child button, another window appears, and each window cycles through the color palette. Click on one of the flat buttons to disable/enable individual threads (the window's color stops/starts changing accordingly).

**Figure 7. The threading demo**

As the notes in the source code point out, however, this application not only demonstrates what C6 can do, it also demonstrates some of the subtle problems that can arise when you try to take manual control of threads, particularly with MDI windows. There are several ways you can lock up this application. One way is to suspend one of the threads and then click on that window to give it focus. The problem here is that because the app is MDI, Windows begins sending messages to the suspended thread. The program notes also point out that the program will hang if you suspend the currently active window. An easy way to do this is to open at least two windows and suspend them all. (The program will point out the error of your ways if you open just one window and attempt to suspend its thread.) When I reported this "bug" (not having read the program notes) Bob Zaunere set me straight, and commented:

In real world cases the RTL does an absolutely brilliant job of protecting developers from the very real complexities of dealing with pre-emptive threads, to the point where it really trivializes the work that most C++ developers, and in fact most textbooks approach with trepidation. In the end, a developer must realize the axiom: with great power comes great responsibility.

Triggers - This .APP demonstrates the use of client-side triggers, using TPS files. You set up client side triggers in the dictionary, using a tab on the table properties window. You can set up triggers for before or after Insert, Update, and Delete actions. Figure 8 shows the trigger dialog with some code for an After Insert trigger.

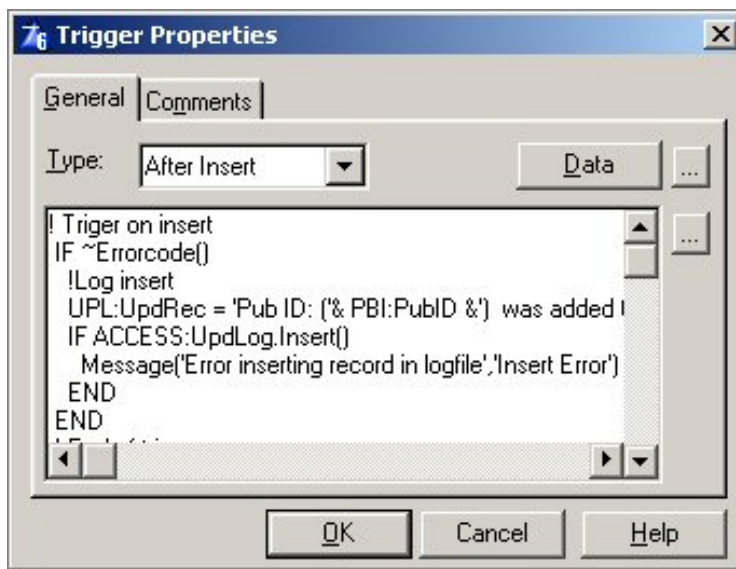


Figure 8. An After Insert trigger

You can expand the code window to full screen, which helps a lot when writing the source. You can also add local data, and that's because this code will be generated as an automatically-called method in the table's FileManager class (look for the *appnamebcn.clw* generated files). The advantage of these client side triggers is that they always execute, no matter which procedure updates the table (assuming, of course, that you use the FileManager or RelationManager methods to do the updating). In the example application, the trigger simply writes some log data out to a text file, such as the following:

```
Pub ID: (9999) was updated in Publisher on 10/15/03 at 12:46PM
Pub ID: (9999) was changed in Pub_Info on 10/15/03 at 12:46PM
```

XMLParse - This is the PEOPLE application modified to export table data in XML format. Figure 9 shows the export buttons on the browse.

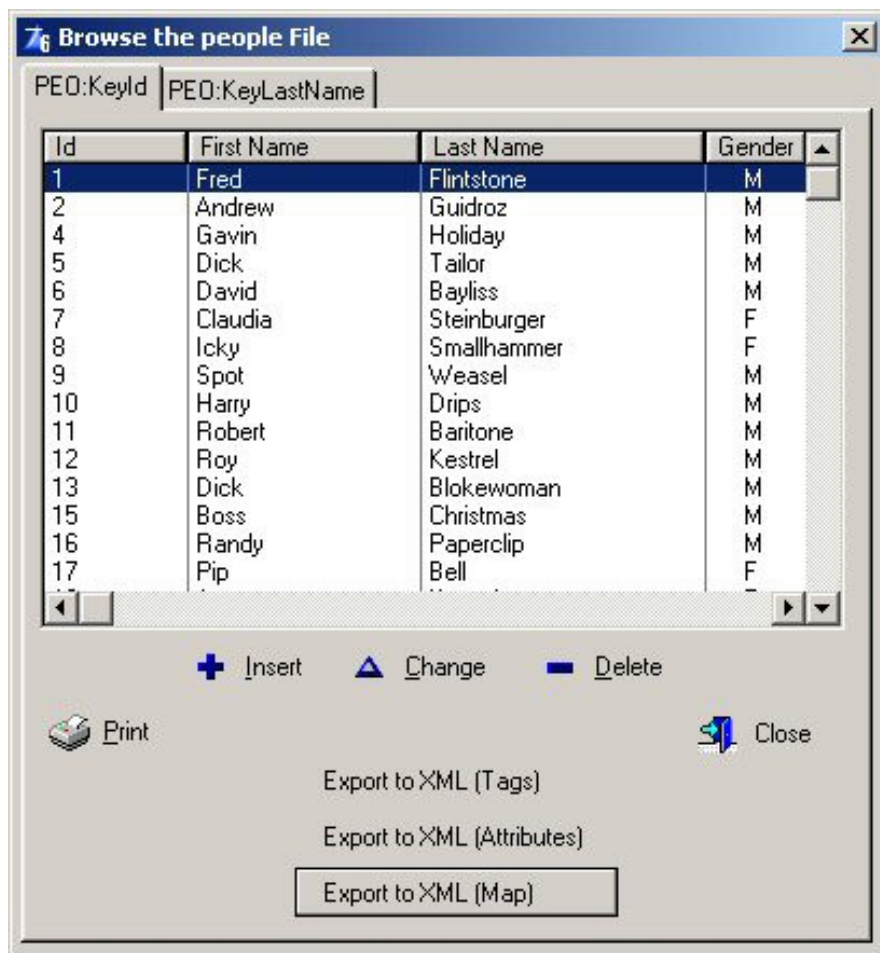


Figure 9. ExportingXML

The three buttons each use a different XML format, and each create a separate XML file. Here's what a portion of the Tags approach creates:

```

<?xml version="1.0" encoding="UTF-8" ?>
- <dataroot>
- <Table>
  <id>1</id>
  <firstname>Fred</firstname>
  <lastname>Flintstone</lastname>
  <gender>M</gender>
</Table>
- <Table>
  <id>2</id>
  <firstname>Andrew</firstname>
  <lastname>Guidroz</lastname>
  <gender>M</gender>
</Table>
- <Table>
  <id>4</id>
  <firstname>Gavin</firstname>
  <lastname>Holiday</lastname>
  <gender>M</gender>
</Table>

```

And here's the same data using Attributes.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <dataroot>
  <row id="1" firstname="Fred" lastname="Flintstone" gender="M" />
  <row id="2" firstname="Andrew" lastname="Guidroz" gender="M" />
  <row id="4" firstname="Gavin" lastname="Holiday" gender="M" />
  <row id="5" firstname="Dick" lastname="Tailor" gender="M" />
  <row id="6" firstname="David" lastname="Bayliss" gender="M" />
  <row id="7" firstname="Claudia" lastname="Steinburger" gender="F" />
  <row id="8" firstname="Icky" lastname="Smallhammer" gender="F" />
  <row id="9" firstname="Spot" lastname="Weasel" gender="M" />
  <row id="10" firstname="Harry" lastname="Drips" gender="M" />
  <row id="11" firstname="Robert" lastname="Baritone" gender="M" />
```

Finally, the data using a Map, which is just a layout you specify that provides alternate tag names:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <dataroot>
  - <Table>
    <Last>Flintstone</Last>
    <First>Fred</First>
    <CustID>1</CustID>
    <Sex>M</Sex>
  </Table>
  - <Table>
    <Last>Guidroz</Last>
    <First>Andrew</First>
    <CustID>2</CustID>
    <Sex>M</Sex>
  </Table>
  - <Table>
    <Last>Holiday</Last>
    <First>Gavin</First>
    <CustID>4</CustID>
    <Sex>M</Sex>
  </Table>
```

Again, it's quite easy to create the XML files - all it takes is filling in a few fields in the `ExportToXML` code template. This XML output is done using the all-Clarion `XMLGenerator` class. If you have industrial strength XML needs, Clarion 6 also ships with wrapper classes for the CenterPoint XML parsers.

Next time...

[Next up](#) on this survey of the C6 changes are my two favorite directories to compare against the previous release: `libsrc` and `template`. There are truly massive changes here, with lots of new code to support features like ADO, XML, PDF, VCR forms, synchronization objects for the new threading model, COM support, and much more. After that I'll turn my attention to IDE improvements and other items not yet covered. Stay tuned!

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross

Santos of Developing Clarion for Windows Applications, published by SAMS (1995). His most recent book is [JSP, Servlets, and MySQL](#), published by HungryMinds Inc. (2001).

Reader Comments

[Add a comment](#)

Flywheel thanks for mentioning it. I had not tried it...

Dave, One other mousewheel solution is RADIS,...

I'm not aware of any problems with the Flywheel driver....

When the trialware expired, I found it impossible to give...

Philip, Clicking on the registration link on the dex...

online sales and delivery
for your applications & tools

Developer **PLUS**

Trees, Recursion, and Many to Many Checkboxes

by Randy Long

Published 2003-10-17

Most of the data I deal with fits nicely into a parent/child, relational data structure. But sometimes I come across data with relationships between data elements of the same kind. In database terms this is sometimes called a self-join, where data in one table relates back to other data in the same table.

For instance, I was recently contracted to maintain a pile of hand coded 2.1 source. As a part of getting things organized, I wanted to be able to trace the calling relationship among the code elements. I wanted to know, for a specific piece of code, which other pieces called it, and which other pieces called each of them. and I settled on a tree display as the most efficient way to present these relationships.

Another more common example of this use of a tree display is the parts explosion. As Figure 1 demonstrates, the widget is composed of the whatnot and the gizmo. The gizmo in turn is composed of the thingamajig and the whoozit. The thingamajig contains another whatnot.



Figure 1. A parts explosion

The common element of both of these examples is that the data can be organized in a list where a row in that list will relate to a set of other rows in the list. Each row in the list potentially "owns" one or more items in other rows of the same list. In parts example, a part might have one or more of the other parts as components, and may also be the component of one or more other parts.

Steve Parker wrote about an inventory application in his [Recursive Lookups](#) article in Clarion Online. His approach was to use a parent-child paradigm, and ALIASes the inventory file to show the child records. That is one approach, but I want to show you a different perspective. I'm interested in managing this type of table using a tree browse to display the iterative data relationships.

The tables and relationships

This application requires two tables and an alias. The first table is the recursive list. It contains primary key column and a name or description column. There is an alias of this table, which will be used to list the related rows.

The second table defines the relationship between a row and a set of rows in the same table. You need the second table because this is a potential [many-to-many relationship](#). It has two columns, a parent and child. In the parts example, these are a part number column and a component column. A key on the Parent column will yield a list of children for each parent. A key on the child column yields a list of parents for each child. A part may have many components, and a part may be a component of a number of other parts. The way this works should become obvious if you compare the component table to the tree representation.

Haven't I been down *this* road before?

Those of you who took time out to review [Steve's article](#) will recognize that this arrangement is the first cousin to the implementation of a standard many-to-many relationship. Fortunately, management of many-to-many relationships has already been discussed in detail in David Harms' [Checkboxes For Many-to-Many Relationships](#) articles. I will use a slight modification of David's code to manage a recursive list.

Back to school

As with David's example, I will enhance the school application to demonstrate the technique. The recursive list will be a list of courses, and their prerequisites. I will modify the school dictionary and create a new application made up of four procedures. I will discuss these four procedures in detail later, but in summary they are:.

preReq_Course - This is the entry procedure. It is a window procedure used to assign prerequisites and display a prerequisite tree for the course prerequisites relationships.

add_Tree_Branch - This procedure navigates a linkage table, such as the component table I mentioned earlier. It identifies each tree branch and level to be added, but does not actually add the branch. This is a generic procedure that can be used to manage any instance of recursive lists. Consequently it will have no knowledge of application specific data or procedures, and all specific information must be passed as parameters.

FormatTreeBranch - This procedure is called by `add_Tree_Branch` to decide if the branch should be added and, if it should, to format the data and add it to the tree. It also notifies `add_Tree_Branch` if the current branch recursion should continue. This procedure is generic, so it can handle any queue format and data required by the application.

check_tree_loop - This is another generic source procedure called by `FormatTreeBranch` to decide if the candidate branch will start a prerequisite loop. For

example, if A is a prerequisite to B, which is a prerequisite to C, which is a prerequisite to A, the relationship cycles back on itself, and the tree would go on forever. `FormatTreeBranch` will tag the branch as a loop in this case and tell `add_Tree_Branch` to stop the recursion process for the current branch.

DataBase changes

To display the data I'll use David's two-browse scheme for managing many-to-many relations, only instead of two tables I'll use the one self-joined table.

The first step is to define an alias for the `Courses` table, which is called `AliasCourses`. Following David's convention, `Courses` is the left table, and `AliasCourses` is the right. I also add a new table, `PreRequisite`, to the dictionary. As with the enrollment example, there are two LONG fields, `PreReq` and `PostReq`. I define a one to many relationship between the `Courses` and `AliasCourses` tables and the `PreRequisite` table. In both relationship, the RI action specified is delete-cascade. See Figure 2.



Figure 2. The table relationships

The preReq_Course procedure

The `preReq_Course` procedure manages prerequisites and displays a prerequisite tree. I make extensive use of David's `cciBrowseClass`. The procedure window is set up with two browses, as defined in [David's article](#). The left browse displays the `Courses` table, and the right the `AliasCourses` table. The local field, `isPreReq`, is used in the right browse for the check box. The right browse uses the `cciBrowseClass` in place of the standard Clarion ABC browse class.

The `cciBrowseClass` initialization in this case is:

```
PreReqBrowse.Init( |
    access:PreRequisite, | ! Linking FileManager
    PreReq:Full_Key, | ! Linking File Key
    PreReq:PreReq, | ! Linking File left field
    PreReq:PostReq, | ! Linking File right field
    COU:Number, | ! Left File Primary key field
    A_Courses:Number, | ! Right File Primary key field
    isPreReq) | ! Local used to show Check Box Icon
```

As anyone who has tried out David's code knows, this is all there is to it. I can now assign prerequisites

to courses.

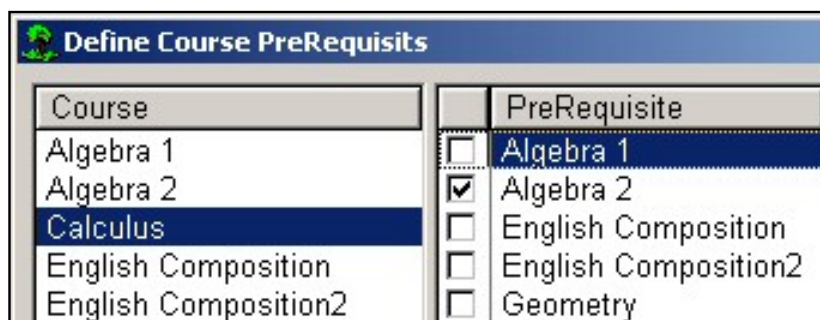


Figure 3. Assigning a prerequisite to a course

I only included the details of this implementation that are unique to this project. Review David's articles and/or the downloaded code for more detail. Incidentally, I'm not using the "[turbo](#)" version David set up to run over a network, but the software he distributed with his [first series](#).

Now, to test the application out, make Algebra 1 a prerequisite for Algebra 2

If Algebra 2 is a prerequisite for Calculus it implies that Algebra 1 is also a prerequisite for Calculus. Remember, I am using a tree control to display this information.

To do this, I now need a third browse to display the tree list. I will be populating a browse without using any existing keys. Some rows, such as the whatnot in the parts example, may be populated more than once. The way to do this is to populate a queue and use a non-template browse control display the queue as a tree. James Cooke in [Handcoding Tree Lists](#) parts [1](#) and [2](#) provides all of the details needed to create a hand coded tree browse. I will not repeat that information, so please review these articles if you aren't comfortable with hand coded tree browses.

Using James Cooke's articles as a reference, I set up a queue to hold the tree information. After the tree fields specified in James' article, I add a fourth LONG type field to hold the ID of the prerequisite row. This will be used to test for a prerequisite loop. For instance, a loop is created if Algebra 2 is a prerequisite for Microcomputers, but Microcomputers is also a prerequisite for Algebra 1 and Algebra 1 is a prerequisite for Algebra 2. In that scenario you'll never be able to sign up for any of these class, and the recursive program, left to its own devices will go on forever.

The add_Tree_Branch procedure

I used Dennis Evans' article [Understanding Recursion](#) as a reference for the recursive programming part of this project. As I mentioned, the procedure, add_Tree_Branch is designed to be generic. This means that all of the information required to do the job, including the procedure to format the tree branch, which is specific to the application, will have to be parameterized and passed to the procedure. Here's the declaration:

```
add_Tree_Branch
  (*QUEUE pTreeQ,          |! The queue for the tree browse
  formatTreeQBranch pFormatBranch, |! proc formatting queue fields
```

```

FileManager pFile,           |! Many-many file
KEY pFileKey,               |! The key for the prereq list
*ANY pNextField,           |! The field w/ next prereq ID
LONG pID,                   |! The ID looking for prereq's
LONG pLevel)                |! The Level of the next branch

```

Here are the guts of the code (you can find the full source in the download at the end of this article):

```

! Get Tree Q level field
TreeLevel  &= WHAT(pTreeQ,2)
! Get the key field to range for subs
KeyField   &= pFile.GetField(pFileKey,1)
! The passed master field to range for subs
pFile.ClearKey(pFileKey)
KeyField = pID
SET(pFileKey,pFileKey)
! Browse link file file for subs
LOOP WHILE pFile.TryNext() = LEVEL:Benign
  IF KeyField ~= pID THEN BREAK.
  NextField = pNextField
  ! Ensure range processing
  FileSaveState = pFile.SaveFile()
  CLEAR(pTreeQ)
  ! Fill passed Level
  TreeLevel = pLevel
  ! Call proc to format branch and
  ! test for Continued recursion
  ContinueRecursion = pFormatBranch()
  IF ContinueRecursion
    add_Tree_Branch(pTreeQ, |
                    pFormatBranch, |
                    pFile, |
                    pFileKey, |
                    pNextField, |
                    NextField, |
                    pLevel + 1)
  END
  ! restore file to insure valid range processing
  pFile.RestoreFile(FileSaveState)
END !LOOP

```

TreeLevel and KeyField are local ANYs which hold references to the passed file fields. The five lines after the KeyField assignment and starting with pFile.ClearKey, are standard code for setting up a range read loop.

NextField is a local LONG, which accepts the value of the passed pNextField for the current record read in the range. Then the tree queue level field is filled with the passed level, and the pFormatBranch function is called to format the branch record and add it to the queue.

`pFormatBranch` also decides if recursion is permissible.

`FormatTreeBranch` uses tables in the dictionary and needs access to the tree queue structure. I provide this access by making the tree queue GLOBAL in scope and THREADED to insure that the queue will not be over-written if the application is compiled in C6. In general terms, the procedure fills the non-level tree queue fields and decides if recursion should continue.

`pLevel` is the level of the current run of records

The LOOP is standard code to read through a set of child records, call `format_Tree_Branch` to format and add the queue record, and notify if recursion should continue. If it should, the procedure calls itself, which makes it recursive, with the current child record passed as the next parent. The file state is saved as the `format_Tree_Branch` could and the recursive call will alter the file state.

The `formatTreeBranch` procedure

This procedure formats the tree branch which, in this case, consists of fetching the name of the course from the courses table. It would be possible to set the branch icon, fill other fields, even insert additional branches.

As this procedure is passed as a parameter to `add_Tree_Branch`, it is prototyped in the Global Map embed point this way (note the TYPE attribute):

```
formatTreeQBranch( ) , BYTE , TYPE
```

This procedure also decides if the branch should be recursed, and signals the calling program with a byte return value. In this case, the decision is based on whether the current branch starts a looping condition. This is determined by a call to `test_tree_loop`.

If the current branch does set up a looping situation, the tree queue description field is tagged [LOOP], and the function returns FALSE to stop further recursion.

The `test_tree_loop` procedure

This generic source procedure accepts a tree queue and a new value, and checks to see if a looping condition is created by the new branch.

To illustrate this, consider the parts explosion example I showed at the beginning of the article. The *whatnot* is in the tree twice but does not cause a looping situation. A looping situation is tested for by tracing the tree back up to the trunk (level = 1), but without testing any other branches to see if the value for the new branch was used between the current node and the trunk.



Figure 4. A parts explosion

You can see, for example, that adding thingamajig, gizmo or widget to the whatnot node would generate a looping situation, but whoozit could be added without any trouble.

Putting it all together

The application displays as a window with three browses, as shown in Figure 5. The first is a list of courses, and the second is another list of courses with a check box in the first column. A checked course in the second is a prerequisite of the course in focus in the first list. Prerequisites may be created and removed by setting and clearing check boxes.

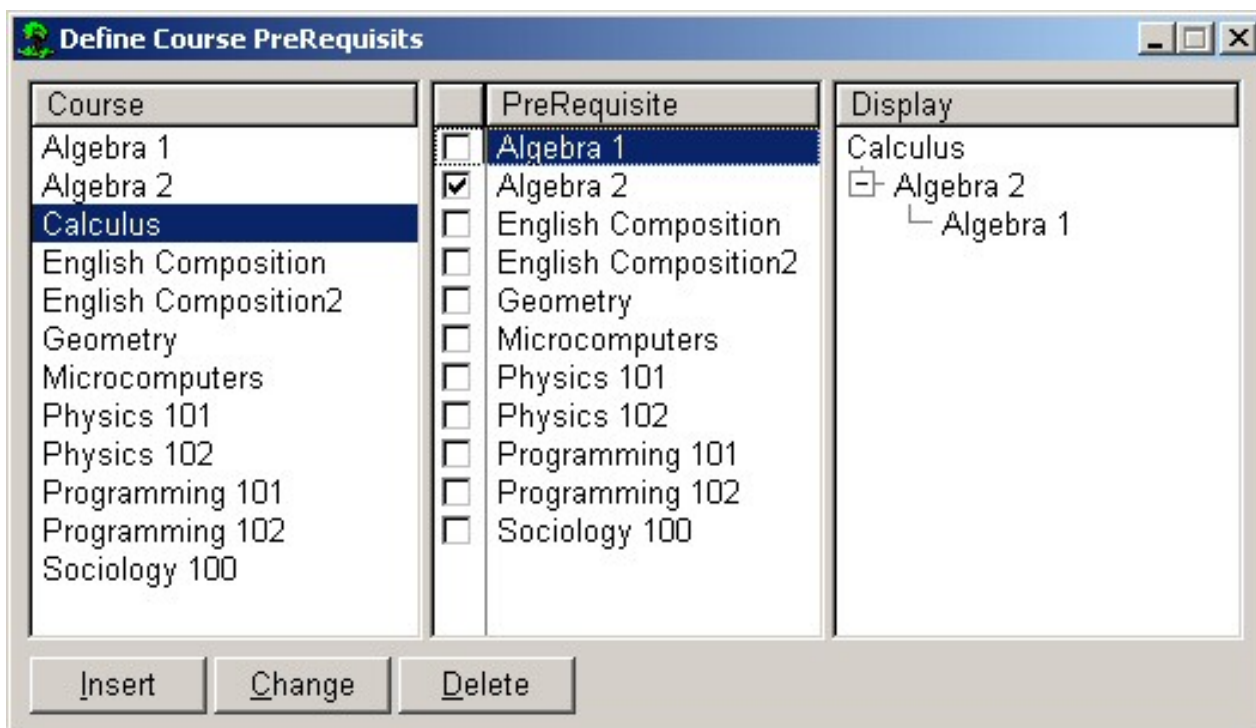


Figure 5. Assigning and displaying prerequisites

The third browse holds the tree browse displaying all generations of prerequisite courses for the course selected in the first browse. As new courses are selected in the left browse, or prerequisites changed in the middle browse, the current prerequisite tree is displayed in the right browse.

While I hope you've picked up a new use of recursive programming, and a new use of the tree browse, I hope you noticed the underlying theme, which is that wheels don't need to be re-invented. This solution makes extensive use of "wheels" generously supplied by other Clarion developers.

Extra credit

If you found this article interesting, here are a couple of ways you can take your education a step further.

1. An obvious project would be to use the generic procedures for another application of recursive lists. This can be done really pretty easily, so that's worth minimal credit.
2. Portable generic code. The generic procedures are embedded in the app, which makes them less than portable. I tried creating a non-dictionary DLL, ran into some problems, and decided the article was taking too long as it was. My "official" reason is that the embedded procedures display better in an educational application! It has been suggested to me that the most efficient way to make them portable is as pure source, which would be included in the module map.
3. Filtering the middle prerequisite. I currently filter this browse so that a course doesn't show up as potentially its own prerequisite. For extra credit, also filter out prerequisites which would generate loops.

[Download the source](#)

Reader Comments

[Add a comment](#)

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

Clarion Magazine

online sales and delivery
for your applications & tools

Developer **PLUS**

Clarion 6 First Look: The Source Code

by David Harms

Published 2003-10-30

This is the second article of a series in which I look at some of the new features in Clarion 6. Many of these are, of course, well-documented in the C6 help, but as I began to show in the [first article](#), sometimes a little direct investigation can reveal things that aren't always obvious in the docs.

In this article I'll continue my directory-by-directory comparison of C5.5 and C6. For this I use [Beyond Compare](#), a nifty utility that not only lets me compare directories, but also files within directories. Last time I looked at the bin and examples directories; now it's time for the libsrc directory.

The libsrc directory

The libsrc directory is where Clarion keeps all of the Clarion source files used in application development, including equate and function definitions, and of course all of the ABC and other class source files. A lot of files have changed here, so I won't describe all the changes or even list all of the changed files. Instead I'll try to hit some of the more interesting highlights. These will be more or less in alphabetical order, by file name.

Changed files

ABAPI.INC/CLW - These files contain Clarion's Windows socket/internet programming functions and classes. Already present in Clarion 5.5, there isn't a whole lot that's new. There are classes here for managing HTTP, FTP and socket connections, and much more. The changes have mainly to do with the new threading capabilities, including an interesting new library function called `AttachThreadToClarion`. This the first line of code in the `EventThread` function, which is a callback function initiated by the Windows API. Without the `AttachThreadToClarion` call the function cannot safely call any Clarion RTL functions.

ABBROWSE.INC/CLW - These files contain the browse-related classes. New in C6 is the `CWEIPManager` class, which is an EIP class that enables you to use the ABC Edit-In-Place classes with the Clarion (legacy) templates or with a hand coded list box.

ABCPHD.INC/CLW - For those who don't want to switch to Clarion 6's default preemptive threading model, there's always the global cooperative threading extension template, which makes use of the `ThreadLocker` and `CooperationClass` classes in these files.

ABDROPS.INC/CLW - The 5.0 and 5.5 version of these files includes the ominous warning: "Unlike the other ABC headers files this one is -not- in its' final form and the interface is liable to change in future releases." In C6 that warning is gone. The code changes are, however quite minor, and none of the method signatures are affected. So if you've been sweating over using `FileDropClass` and `FileDropCombo` class, I guess you can breathe easier now.

ABDST.INC/CLW - The SMTP code gets a minor bug fix or two and adds authentication.

ABEIP.INC/CLW - These files contain the ABC Edit-In-Place classes. A new addition is the `EditCalendarClass`, which makes use of the `CalendarBaseClass` from `ABWINDOW.INC`.

ABERROR.INC/CLW - Relatively minor changes here, except that `ErrorClass` now has `GetError()` and `GetErrorCode()` methods to make it much easier for you to determine the actual cause of a thrown error. New error messages reported by this class include "File could not be closed", messages related to handling Windows Meta Files, several report writer messages, and a "record changed by another station" message for concurrency failure from a browse (previously this message was specific to forms).

ABFILE.INC/CLW - There are a moderate number of changes to the ABC file classes (mainly `FileManager`, `RelationManager`, and `ViewManager`), some of which are directly related to the change in the threading model. Pre-C6, you only had one `FileManager` per file or aliased file, no matter how many threads might use that `FileManager`. ABC used a queue (called `FileThreadQueue`) and the `Thread()` function to keep track of the state of the `FileManager` for each thread. In ABC, the `FileManager` class is threaded, so none of this needs to happen. If you open two copies of a browse in C6, there will be a `FileManager` instance for each thread. Other changes include the new blob handling code. The `RelationManager` now has a `LogoutTimeout` property - previously this value was fixed at 2. Also new is a check to do transactions on aliased files, only if the driver supports this feature. The `ViewManager` adds a number of new methods, including `GetFreeElementAscending` (virtual), `GetFieldAscending`, `GetFieldName`, `GetFirstSortField`, `GetFirstSortFieldName`, `RestoreBuffers` (virtual) and `SaveBuffers` (virtual). There are also changes to the `DbChangeManager` and `DbAuditManager` classes to support the new threading model (by way of critical sections).

ABPOPUP.INC/CLW - The `PopupClass` has a number of changes to reflect the new threading model, and also adds `DeleteMenu` and `GetLastNumberSelection` methods.

ABREPORT.INC/CLW - The report classes changes have mainly to do with support for the PDF, HTML, XML, and Text output options.

ABTOOLBA.INC/CLW - The toolbar classes have been updated to support VCR forms.

ABUTIL.INC/CLW - These files contain utility classes. There are changes to `ConstantClass` to support terminator fields, and to `INIClass` to support storing data in the registry and in tables, as well as INI files (and to make the class thread safe). New in this file is `CalendarBaseClass`, a popup calendar. There are two derived classes, `CalendarClass` and `CalendarSmallClass` which use, respectively, small and large windows. The window structures (two, one small, one large) are in `ABUTILUI.INC`, so if you don't like the look and feel you can, presumably, make sure that your own version of this file will be found first by your redirection file.

ABWINDOW.INC/CLW - The `WindowManager` class is at the heart of any ABC window, and provides all the basic event handling as well as support for form actions. This class now has support for the new `FormVCR` class. New properties include `BatchProcessing`, `SaveControl`, and `DisableCancelButton`; new methods include `AddItem` (for adding a `FormVCRWindowComponent`), and the `TakeDisableButton`, `InsertAction`, `ChangeAction`, `DeleteAction` (all three were formerly routines), `SaveOnChangeAction`, and `SaveOnInsertAction` virtual methods. All but the first virtual method in that list are called via the `TakeCompleted` method.

DEFAULTS.CLW - This source file contains the default window and report structures available to the IDE. There are a number of new reports and windows, and remember that you can also add your own to this file (although you would probably still need to keep the original around for upgrading purposes).

EQUATES.CLW - New equates include: `EVENT:Notify`, `CREATE:rtf`, `BRUSH:Solid`, `BRUSH:Null`, `BRUSH:Hollow`, `BRUSH:Hatched`, `BRUSH:Pattern`, `BRUSH:Indexed`, `BRUSH:DIBPattern`, `MSGMODE:SysModal`, `MSGMODE:CanCopy`, `WINDOW:OK`, `WINDOW:NotOpened`, `WINDOW:BadWindow`, `WINDOW:ClosePending`, `WINDOW:InDestroy`, `TEXT:Field`, `TEXT:File`, `PrintPreviewFileQueue`, `DriverOp:STARTTRAN`, `DriverOp:ENDTRAN`, and a whole bunch of equates for the registry functions.

ERRORS.CLW - This file lists a couple of new errors, including `KeyLengthErr`, `StreamErr`, and `TriggerErr`.

FILECB.INC - This file contains the file callback interfaces and functions. Minor changes to support BLOBs.

New source files

ABPRHTML.INC/CLW, ABPRPDF.INC/CLW, ABPRTEXT.INC/CLW, and ABPRXML.INC/CLW - These classes (`HTMLGenerator`, `HTMLReportGenerator`, `PDFGenerator`, `PDFReportGenerator`, `TextGenerator`, `TextReportGenerator`, `XMLGenerator`, and `XMLReportGenerator`, respectively) handle the conversion of Clarion reports to the specified output format.

ABPRTARG.INC/CLW - TargetGenerator is a class that provides some basic DOS file input/output, in support of the various report generator classes. This one looks to be quite useful as a general purpose file handling class.

ABPRNAME.INC/CLW - The NameGenerator class creates file names based on a mask which can include numbers such as a page number and total pages.

ABRPATMG.INC/CLW - AttributeParser and ReportAttributeManager are specialized classes used (apparently) in the conversion of reports to output formats like XML and HTML.

ABRPPSEL.INC/CLW - The ReportTargetSelector class lets the user choose the report output. Notable here is the AddItem method, which takes as its first parameter any class implementing the IReportGenerator interface. In other words, this class is designed to accept report output plugins.

ABRULE.INC/CLW - The Rule class, and the RulesCollection and RulesManager classes are all part of C6's implementation of business rules.

ABVCRFRM.INC/CLW - The FormVCR class lets you page through and insert/change/delete table records without using a browse.

ABWMFPAR.INC/CLW - The WMFParser and WMFDocumentParser classes are used by the ReportManager to extract report data for conversion to other formats like XML, PDF, etc.

ADOINT.INC, ADOPRCS.CLW, ADOPRCS.INC, ADOPROCC.CLW, ADOPROCC.INC, ADO_SQL.INT, CFILTBASE.INC/CLW, CFILTERLIST.INC/CLW, CFLDPAIR.INC/CLW, CWADO.INC/CLW, QPROCESS.INC/CLW, SVADO.INC/CLW, SVHELPER.INC/CLW, SVMAPPER.INC/CLW - These files contain ADO-related code, including numerous classes, equates and interface definitions.

CPXML.INC/CLW, XMLCLASS.INC, XMLEXCHANGE.CLW, XMLNAMEMAP.CLW, XMLNAVIGATOR.CLW, XMLSCHEMA.CLW, XMLTREEJOIN.CLW, XMLTYPEINFO.INC - CPXML contains the wrapper classes for the CenterPoint SAX and DOM XML parsers, which let you do industry standard, powerful, and complex XML work, using familiar Clarion code. The other files contain classes for converting between XML and Clarion data structures including files, queues, and groups.

CLACVT__.INC/CLW - These files are the source code for the Clarion database conversion utility.

CWSYNCH.INT, CWSYNCHC.INC/CLW, CWSYNCHM.INC - These files contain the thread synchronization interfaces, classes, and functions needed to manage threading issues in C6's true multi-threading environment.

ODBCATTR.CLW - Equates for use with `PROP: StmtAttr` and `PROP: GetInfo`.

RTF.INT, RTFCTL.INC/CLW - These files declare the RTF support classes, used by the `RTFTextControl` template.

SVAPI.INC, SVAPIFNC.INC - A new set of Windows equates and function prototypes.

SVBASE.INC/CLW, SVCOM.INC/CLW, SVCONDEF.INC, SVHELPER.INC/CLW - More COM and ADO base classes.

SVDATE.INC/CLW - SQL/Clarion date conversion code, including the `CDateConverter` and `SystemDateTime` classes.

SVGRAPH.INC/CLW/EQU - The new business graphing classes.

Built in functions

Core Clarion language statements can be grouped into two kinds, those implicitly understood by the compiler (`IF`, `LOOP`, `ACCEPT`, `CASE` etc) and those declared explicitly in `BUILTINS.CLW`. The following are the changes to `BUILTINS.CLW` as of Candidate Release 6:

`GETSTATE` and `RESTORESTATE` - These functions now have a second, omissible byte parameter which defaults to false. If true, the functions will get and restore the state of any `BLOB` fields.

`ARC`, `BOX`, `CHORD`, `ELLIPSE`, `IMAGE`, `LINE`, `PIE`, `POLYGON`, and `ROUNDBOX` - These methods now have an extra omissible parameter for an attribute list. From the help: "A string constant, variable, or `EQUATE` containing an optional type of output document and its associated attributes. Only valid when the target is a `REPORT`."

`BINDEXPRESSION` - A new function that lets you bind mathematical expressions so they don't have to be evaluated every time they are called within an `EVALUATE`.

`DELETEREG`, `GETREG` and `PUTREG` - New functions to update registry values.

`FILEDIALOGA` - This variant of `FILEDIALOG` adds an optional parameter to specify the default extension by position, i.e. a value of 2 would select the second extension listed in the function call as the default extension.

`INSTANCE` - There are four versions of the `INSTANCE` function, for files, keys, queues, and unknown variables. In each case the function returns the address of that instance of the variable on the specified thread. For, um, instance, if you have four threads running, in each thread you could call `INSTANCE` against a global queue with the `THREAD` attribute and get a different address in each thread.

NOTIFY, NOTIFICATION, EVENT:Notify - These functions and equate are intended replacements for the SETTARGET(, thread) syntax. You use NOTIFY to send the message to the receiving thread, where it triggers an EVENT:Notify. The receiving thread then uses NOTIFICATION to retrieve the message. This is now the recommended way to accomplish inter-thread communication.

POPERERRORS and PUSHERRORS - These functions save/retrieve standard error information like ERROR(), ERRORCODE() etc. using a LIFO stack. Each thread has its own stack.

QUOTE and UNQUOTE - If you've ever gone crazy trying to build strings that use single quotes ('), left angle brackets (<), or left braces ({}), all of which have to be doubled because they are special characters, you'll really like these new functions. Respectively, they double and "undouble" these characters.

SUSPEND and RESUME - These functions let you suspend and resume thread execution. As noted in the multi-threading example, you need to use extreme caution when suspending MDI threads, as you can cause your application to lock up. This is because of the design of MDI, not because of any limitation in Clarion's threading implementation.

STATUS - This overloaded function now works on windows, and returns one of the following values: WINDOW:OK, WINDOW:NotOpened, WINDOW:BadWindow, WINDOW:ClosePending, or WINDOW:InDestroy.

But what does it all mean?

There were a few things that surprised me in my survey of the libsrc changes. One is that the new threading model required such minor changes to the ABC library. True, there are a number of new classes to support thread management, but otherwise the migration of the ABC code base seems to have been fairly straightforward. I think that's an encouraging sign for anyone worried about making the transition to C6. In fact, there are few changes overall to the existing ABC classes - most of the work is in new classes.

The changes in the libsrc directory certainly don't tell all the story of what's new and improved, but they are an important indicator. Ultimately, most of what a AppGen application can do depends on these source files. One of the highlights certainly is the work done on new report output formats (XML, text, PDF, HTML). I'm also very pleased to see the new XML classes, especially the CenterPoint wrapper classes. There are significant browse and form enhancements, classes for business rules, and a number of utility classes. I am amazed at the number of ADO-related source files, and I hope the new ADO code gets a lot of exposure and use. As the ADO code is built on COM, it's also clear that Clarion's COM support has come a long way in C6. And of course there are a number of improvements and additions to the built-in functions.

I had really intended to start on some of the template changes this week, but I'm already out of room, so it'll have to wait until next time.

*[David Harms](#) is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of *Developing Clarion for Windows Applications*, published by SAMS (1995). His most recent book is [JSP, Servlets, and MySQL](#), published by HungryMinds Inc. (2001).*

Reader Comments

[Add a comment](#)

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

Reborn Free**CLARION**
online

Data Structures and Algorithms Part XXIV - Floyd's All Pairs Algorithm

by Alison Neal

Published 2003-10-30

In my [last article](#) I discussed Dijkstra's Shortest Path Algorithm and how it can provide the shortest path between two known nodes. In this article I will discuss Floyd's All Pairs algorithm, which provides the shortest path between all pairs of nodes in a graph.

Staying with the same graph implementation and test data as I used in my last article, here is the node list:

Node1	Node2	Weight
Cgo	LV	1780
Cgo	Mmi	1423
Cgo	NO	948
Cgo	Sea	2060
Ds	LA	1440
Ds	Min	949
Ds	NO	517
Ds	NYC	1614
LA	Ds	1440
LA	LV	272
LA	SF	414
LV	Cgo	1780
LV	LA	272
Mmi	Cgo	1423
Min	Ds	949
Min	NYC	1217
Min	OkC	792
Mwk	Px	1771
Mwk	SF	2257
Mwk	WDC	811
NO	Cgo	948
NO	Ds	517
NYC	Ds	1614
NYC	Min	1217

NYC	WDC	237
OkC	Min	792
Px	Mwk	1771
SF	LA	414
SF	Mwk	2257
SF	Sea	808
Sea	SF	808
Sea	Cgo	2060
WDC	Mwk	811
WDC	NYC	237

The output file that I expect to get from the algorithm is this:

	Cgo	LV	Mmi	NO	Sea	Ds	LA	Min	NYC	SF	OkC	Mwk	Px
WDC													
Cgo	0	1780	1423	948	2060	1465	2052	2414	3079	2466	3206	4127	5898
3316													
LV	1780	0	3203	2229	1494	1712	272	2661	3326	686	3453	2943	4714
3563													
Mmi	1423	3203	0	2371	3483	2888	3475	3837	4502	3889	4629	5550	7321
4739													
NO	948	2229	2371	0	3008	517	1957	1466	2131	2371	2258	3179	4950
2368													
Sea	2060	1494	3483	3008	0	2662	1222	3611	4113	808	4403	3065	4836
3876													
Ds	1465	1712	2888	517	2662	0	1440	949	1614	1854	1741	2662	4433
1851													
LA	2052	272	3475	1957	1222	1440	0	2389	3054	414	3181	2671	4442
3291													
Min	2414	2661	3837	1466	3611	949	2389	0	1217	2803	792	2265	4036
1454													
NYC	3079	3326	4502	2131	4113	1614	3054	1217	0	3305	2009	1048	2819
237													
SF	2466	686	3889	2371	808	1854	414	2803	3305	0	3595	2257	4028
3068													
OkC	3206	3453	4629	2258	4403	1741	3181	792	2009	3595	0	3057	4828
2246													
Mwk	4127	2943	5550	3179	3065	2662	2671	2265	1048	2257	3057	0	1771
811													
Px	5898	4714	7321	4950	4836	4433	4442	4036	2819	4028	4828	1771	0
2582													
WDC	3316	3563	4739	2368	3876	1851	3291	1454	237	3068	2246	811	2582
0													

The code for the algorithm itself is as follows:

```
graph.FloydAP          PROCEDURE ( )
!-----
INT_MAX      EQUATE ( 2147483647 )
Dist         ULONG ( 0 )
I            ULONG ( 0 )
J            ULONG ( 0 )
```

```

K          ULONG(0)
E          &EdgeData
edgeAddress LONG(0)
DistArray  ULONG(0),DIM(SELF.nSize),DIM(SELF.nSize)

```

```
CODE
```

```

IF exp.dosopen('export.txt',1) THEN RETURN -1.
IF exp.setpointer(0,0) THEN RETURN -1.
LOOP i = 1 TO SELF.nSize
  SELF.expFile('<9>' &SELF.nodeName(i))
  LOOP J = 1 TO SELF.nSize
    DistArray[i][j] = INT_MAX
  END
  DistArray[i][i] = 0
END
SELF.expFile('<13,10>')

LOOP i = 1 TO SELF.nSize
  IF SELF.FirstEdge(i)
    E &= (SELF.FirstEdge(i))
    LOOP
      DistArray[i][E.toNode] = E.weight
      edgeAddress = SELF.nextEdge(i)
      IF edgeAddress <= 0 THEN BREAK.
      E &= (edgeAddress)
    END
  END
END
LOOP k = 1 TO SELF.nSize
  LOOP i = 1 TO SELF.nSize
    IF DistArray[i][k] < INT_MAX
      LOOP j = 1 TO SELF.nSize
        Dist = DistArray[i][k] + DistArray[k][j]
        IF Dist < DistArray[i][j]
          DistArray[i][j] = Dist
        END
      END
    END
  END
END
LOOP i = 1 TO SELF.nSize
  SELF.expFile(SELF.nodeName(i) &'<9>')
  LOOP j = 1 TO SELF.nSize
    SELF.expFile(DistArray[i][j] &'<9>')
  END
  SELF.expFile('<13,10>')
END
exp.dosclose()
RETURN 0

```

The first thing to happen here is the initialization of a distance array, note that where the start node and end node are the same, then the distance is always going to be zero. Otherwise, as I am looking for the shortest path, each position in the distance array is initialized to a suitably high number, for later comparisons, in this instance I have used `INT_MAX` for this purpose. Thus, the distance array looks like Figure 1 on completion of the first loop

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	CGO	LV	MMI	NO	SEA	DS	LA	MIN	NYC	SF	OKC	MWK	PX	WDC
CGO	0	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
LV	MAX	0	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
MMI	MAX	MAX	0	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
NO	MAX	MAX	MAX	0	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
SEA	MAX	MAX	MAX	MAX	0	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
DS	MAX	MAX	MAX	MAX	MAX	0	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
LA	MAX	MAX	MAX	MAX	MAX	MAX	0	MAX	MAX	MAX	MAX	MAX	MAX	MAX
MIN	MAX	MAX	MAX	MAX	MAX	MAX	MAX	0	MAX	MAX	MAX	MAX	MAX	MAX
NYC	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	0	MAX	MAX	MAX	MAX	MAX
SF	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	0	MAX	MAX	MAX	MAX
OKC	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	0	MAX	MAX	MAX
MWK	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	0	MAX	MAX
PX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	0	MAX
WDC	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	0

Figure 1.

The second loop of the algorithm, calculates the distance for those nodes that are directly connected to each other by a single edge. As you can see from the test file this incorporates such distances as CGO to LV, which has a fictitious distance of 1780 miles. When the second loop finishes, the distance array looks like Figure 2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	CGO	LV	MMI	NO	SEA	DS	LA	MIN	NYC	SF	OKC	MWK	PX	WDC
1 CGO	0	1780	1423	948	2060	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
2 LV	1780	0	MAX	MAX	MAX	MAX	272	MAX	MAX	MAX	MAX	MAX	MAX	MAX
3 MMI	1423	MAX	0	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
4 NO	948	MAX	MAX	0	MAX	517	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
5 SEA	2060	MAX	MAX	MAX	0	MAX	MAX	MAX	MAX	808	MAX	MAX	MAX	MAX
6 DS	MAX	MAX	MAX	517	MAX	0	1440	949	1614	MAX	MAX	MAX	MAX	MAX
7 LA	MAX	272	MAX	MAX	MAX	1440	0	MAX	MAX	414	MAX	MAX	MAX	MAX
8 MIN	MAX	MAX	MAX	MAX	MAX	949	MAX	0	1217	MAX	792	MAX	MAX	MAX
9 NYC	MAX	MAX	MAX	MAX	MAX	1614	MAX	1217	0	MAX	MAX	MAX	MAX	237
10 SF	MAX	MAX	MAX	MAX	808	MAX	414	MAX	MAX	0	MAX	2257	MAX	MAX
11 OKC	MAX	MAX	MAX	MAX	MAX	MAX	MAX	792	MAX	MAX	0	MAX	MAX	MAX
12 MWK	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	2257	MAX	0	1771	811
13 PX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	1771	0	MAX
14 WDC	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	237	MAX	MAX	811	MAX	0

Figure 2.

The triply-nested loop calculates the distance for those edges that are not linked to each other directly. The first position calculated is (1,6) CGO to DS. This is because everything else in column one already contains its lowest value.

When the loops iterate until $k = 4$ and $i = 1$, the code produces the initial value of 948 – position (4,1) in the matrix. When j is equal to 6, the position of (k, j) in the matrix (4,6) the value is 517. That is, there are 948 Miles from CGO (Node 1) to NO (Node 4), and 517 Miles from DS (Node 6) to NO (Node 4), giving a total of 1465 miles in total from CGO to NO, via DS, and this is lower than INT_MAX. Thus position 6 in column 1 is updated to that total as the shortest known indirect, from CGO to DS, so far identified in the algorithm. This yields figure 3.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	CGO	LV	MMI	NO	SEA	DS	LA	MIN	NYC	SF	OKC	MWK	PX	WDC
1 CGO	0	1780	1423	948	2060	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
2 LV	1780	0	MAX	MAX	MAX	MAX	272	MAX	MAX	MAX	MAX	MAX	MAX	MAX
3 MMI	1423	MAX	0	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
4 NO	948	MAX	MAX	0	MAX	517	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
5 SEA	2060	MAX	MAX	MAX	0	MAX	MAX	MAX	MAX	808	MAX	MAX	MAX	MAX
6 DS	1465	MAX	MAX	517	MAX	0	1440	949	1614	MAX	MAX	MAX	MAX	MAX
7 LA	MAX	272	MAX	MAX	MAX	1440	0	MAX	MAX	414	MAX	MAX	MAX	MAX
8 MIN	MAX	MAX	MAX	MAX	MAX	949	MAX	0	1217	MAX	792	MAX	MAX	MAX
9 NYC	MAX	MAX	MAX	MAX	MAX	1614	MAX	1217	0	MAX	MAX	MAX	MAX	237
10 SF	MAX	MAX	MAX	MAX	808	MAX	414	MAX	MAX	0	MAX	2257	MAX	MAX
11 OKC	MAX	MAX	MAX	MAX	MAX	MAX	MAX	792	MAX	MAX	0	MAX	MAX	MAX
12 MWK	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	2257	MAX	0	1771	811
13 PX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	1771	0	MAX
14 WDC	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	237	MAX	MAX	811	MAX	0

Figure 3.

The loop continues until all positions in the matrix have been calculated and the lowest distance has been assigned.

Summary

Floyd's algorithm is a handy piece of code to have around, and not just for calculating the shortest distance. If the edge weights are related to duration, then you can calculate the quickest path to take, not just the shortest; this could be useful for travel applications and project management.

In my next article I will continue with Graph Algorithms by taking a look at the Transitive Closure.

[Download the source](#)

Alison Neal has been using Clarion since 2000, whilst working for [Asset Information Systems](#) (AIS) in Auckland, New Zealand. Some years ago (at the tender age of 19) Alison graduated from the Central Institute of Technology in Wellington, New Zealand with a major in Cobol. She also has a BA in English literature and has studied Computer Science, Philosophy and Information Systems. AIS is an independent division of Asset Forestry Ltd, and has a team of five programmers developing almost exclusively in Clarion. AIS also offers web (ClarionNET) and email services for the customer who needs everything. The company has many and varied customers bridging across a wide range of industries including Telecommunications, Forestry & Agriculture, Manufacturers, Military & Government, Legal & Financial, and Retail.

Reader Comments

[Add a comment](#)

Clarion Magazine



The Cape Town Clarion Essentials Training Course

by **Danie de Beer**

Published 2003-10-30

On 6 October 2003 Raymond Dummer held a 4-day Clarion Essentials Training course at the Green House Faculty of Training Institute in Cape town, South Africa. The course ended with an additional day on object-oriented programming.

The course was attended by nine Clarion Programmers, some as far as Johannesburg (1200km from Cape town) and Arniston (250km from Cape town). Raymond really went out of his way to make everyone feel welcome and relaxed, with a few funny jokes and fine sense of humour.



The really nice about the course was that you didn't have to bring your own PC or Laptop. The course was held at a Professional Training Institute, with a well set up Computer Laboratory.

The Course schedule was a follows:

Daily Routine:

Start 08:30

Break 10:00 (Raymond's brain break)

Lunch 12:30-13:00

Break 15:00

End 16:30

Q & A 16:30-??

Day One:

- Introduction
- The Data Dictionary
- Events and Properties
- New Applications

Day Two

- Dynamic Link Libraries
- Clarion Language Statements
- Introduction to OOP

Day Three

- Source Embeds
- Reports
- Debugger

Day Four

- OLE/OCX
- API

Day Five.

- More About OOP
- Closure and some nice things.

Include in the course was a nice meal everyday, to take care of our empty stomachs, and to get us ready for the final part of the day.



What made the course really interesting and very helpful was the fact that Raymond gave us a lot of extra software (source) examples of the Topics covered in the Essentials manual and Handy Utilities. And as with any course... you really pick up some extra tips and help when it comes to the "smoke break," or as Raymond called it, the "Brain Break." There he gave us some useful tips and shortcuts not really covered in the Essential course, but that come in very handy in every day programming.

The course finished on October 10th, and it was really sad to say goodbye to all the fellow students or Clarion Programmers.



From left to right, front row: Nazla (a.k.a. Nashua), Ernest, Jits , Neil, Robbie. Back row: Danie, Grant, Gerhard, Raymond, Leonard.

The feedback from the participants was very positive, and I felt the course was worth every cent. If you have the opportunity to attend Raymond's courses, jump for it.

For more course information, you can contact Raymond at raymond@soggy.co.za or visit his [web page](#) for course schedules and updates.

Reader Comments

[Add a comment](#)

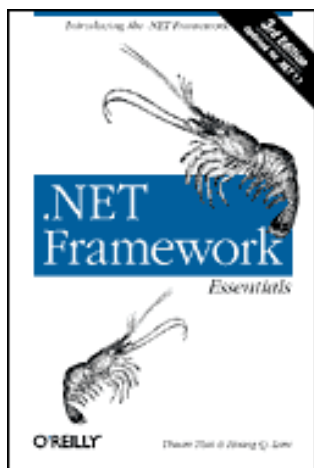
Clarion Magazine



Book Review: .NET Framework Essentials

by David Harms

Published 2003-10-31



[.NET Framework Essentials \(3rd Edition\)](#)

by Thuan Thai & Hoang Q. Lam
published by O'Reilly, 2003
ISBN: 0-596-00505-9
364 pages, \$29.95 US, \$46.95 CA

Yes, I know, this magazine *is* called Clarion Magazine. So what's a .NET book review doing here? Well, in part, it's because Clarion developers create Windows applications, and .NET and the technologies evolving from .NET (such as those unveiled in Microsoft's [Longhorn SDK](#)) are central to Microsoft's future plans. And in part it's because a lot of us don't just use Clarion – we also use other software development technologies where appropriate.

.NET represents a massive revision of not only Microsoft's programming languages, but also of the company's strategy for component based development. And although the .NET moniker suggests that this strategy is aimed largely at Internet-based development, the changes in .NET are profound for desktop and client-server development also.

In part, .NET is an attempt to bring many Windows programming languages under one umbrella, by means of the Common Language Runtime, or CLR, which manages the execution of all code. There are .NET versions of C++, VB, C#, and event J#, among others. Because all these languages adhere to the conventions of the Common Type System (CTS), you can mix and match .NET languages; for instance, you can use a VB.NET class in a C# application.

.NET also provides a core library, called the .NET Framework. This includes classes for tasks such as manipulation of simple data types, system I/O, collections (arrays, lists, hash tables, etc), threading, reflection (dynamic binding and type inspection), security, and networking. There are also classes for ADO, XML, ASP, and Windows Forms (more about this last item later).

Units of code and/or data, which we might call LIBs, DLLs, EXEs, or COM components, are in the .NET world called *assemblies*. An assembly is, to quote the authors of this book, "a basic unit of versioning, deployment, security management, side-by-side execution, sharing, and reuse." That's quite a mouthful, but among other things it means that an .NET assembly knows everything it needs to know about itself. This is in sharp contrast to COM, where an incorrect or missing registry setting can blow your component, and your application, out of the water. There are no registry settings required in the .NET world. In fact, .NET brings back a very old installation tool: XCOPY. That's right – you copy in the files to install your program, and you delete them to uninstall. Mind-boggling. And as a bonus, no more DLL hell where one version of a DLL overwrites another (although recent versions of Windows have done much to address this problem already).

You can learn all about assemblies and other general .NET topics from the book's first four chapters: .NET Overview, The Common Language Runtime, .NET Programming, and Working with .NET Components. The remaining five chapters deal with specific aspects of .NET Programming, and are typically demonstrated with C# examples.

Chapter five, Data and XML, will be of some interest to anyone who is using or considering [Fenix](#), RadVenture's ASP.NET code generator for Clarion. But this chapter is as much about XML as it is about ADO, and although XML is increasingly important to Clarion developers, I think most of us are still a lot more concerned about traditional databases.

As with the chapter on Data and XML, the chapters on Web Services, ASP.NET, and Mobile Devices provide a light briefing, mainly helpful in filling out your understanding of what .NET can do.

I've singled out the chapter on Windows Forms because this area comes closest to what most of us think of as Windows application development. As the authors point out, "classic Windows development is tedious and error-prone." Windows Forms is the latest in a series of efforts by Microsoft to make this process easier. The Microsoft Foundation Classes and the Active Template Library both left much to be desired, and COM/ActiveX components have their own problems. Windows Forms promises easier development, simpler component reuse, and faster integration with services such as ADO.NET. Should a future version of Clarion support .NET (and like other developers, I can only speculate) it seems to me that ABC or an ABC-like library would necessarily make extensive use of Windows Forms. And the massive amount of ADO code in C6 also gives SoftVelocity a basis for future ADO.NET development.

This book really isn't a programming guide, despite the numerous source examples, but it is a good introductory text and reference.

*[David Harms](#) is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of *Developing Clarion for Windows Applications*, published by SAMS (1995). His most recent book is *JSP, Servlets, and MySQL*, published by HungryMinds Inc. (2001).*

Reader Comments

[Add a comment](#)

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.