Clarion Magazine

PDF For January, 2003

All Clarion Magazine articles for January, 2003 in PDF format.

Posted Sunday, February 02, 2003

Interfaces Everywhere

Some time ago Jim Kane created the OLETCL class to use COM objects with a VB-like syntax. OLETCL relied on a quirk in Clarion's COM and thread handling, but with Clarion 6's new threading model OLETCL will no longer work. That prompted Jim to finish a COM interface generator he'd started many years earlier.

Posted Monday, February 03, 2003

Introducing PostgreSQL - Creating Tables And Sequences

Dave Harms resumes his series on the open source PostgreSQL database with a look at table creation, and the intricacies of PostgreSQL's *sequences*, which are used primarily for autonumbering primary keys.

Posted Thursday, February 06, 2003

Weekly PDF For February 2-8, 2003

All ClarionMag articles for February 2-8, 2003 in PDF format.

Posted Monday, February 10, 2003

New Topics Created For Clarion 6

The Clarion Magazine Topical Index now has several topics for Clarion 6 articles, and more will be added as needed.

Posted Monday, February 10, 2003

A Class For Tagging

Steve Parker is on record as not feeling the need to create classes. Yet here he is, creating a class version of his tagging code. It's the end of the world as we know it.

Posted Thursday, February 13, 2003

CONVIC - An Antipodean Clarion Gathering

If you're not an Australian Clarion developer then there's a good chance you haven't heard of ConVic. Does this matter, you ask? Well, ConVic's

News

been on the Australian landscape for five years now and for the last four, it's been the only Clarion conference in Australia. What's more, ConVic 2003 is coming up at the end of March; it's an opportunity not to be missed by Australian developers, and overseas developers who might like to consider combining the conference with a sightseeing trip down under.

Posted Friday, February 14, 2003

Demystifying C6 Threading (Part 1)

Mutexes, semaphores, critical sections, reader/writer locks; all of these things are part of Clarion 6, and they all have to do with the new support for unlocking Clarion threads so they run like real operating system threads. Should you care? If you write any embedded code, yes, you should. Should you worry? That all depends on what kind of embedded code you write. Part 1 of 2.

Posted Saturday, February 15, 2003

Weekly PDF For February 9-15, 2003

All ClarionMag articles for February 9-15, 2003 in PDF format.

Posted Wednesday, February 19, 2003

Demystifying C6 Threading (Part 2)

Mutexes, semaphores, critical sections, reader/writer locks; all of these things are part of Clarion 6, and they all have to do with the new support for unlocking Clarion threads so they run like real operating system threads. Should you care? If you write any embedded code, yes, you should. Should you worry? That all depends on what kind of embedded code you write. Part 2 of 2.

Posted Thursday, February 20, 2003

Data Structures and Algorithms Part XVI - The Huffman Compression Algorithm (Part 1)

In her latest installment, Alison Neal discusses the Huffman compression algorithm, which is the same compression algorithm that is used by PkZip. The algorithm yields approximately 40% compression for text files. The test application included with the article reduces the provided test file from 20kb to 12kb in size, and then decompresses it back to its original state. Part 1 of 2.

Posted Friday, February 21, 2003

Weekly PDF For February 16-22, 2003

All ClarionMag articles for February 16-22, 2003 in PDF format.

Posted Monday, February 24, 2003

Data Structures and Algorithms Part XVI - The Huffman

SkyHI Debug Template

ImageEx 2 Gold Release

Newsletter Service

Win A Free Copy Of cpTracker

cpTracker Silver R2 Available

chSTD Library Version 2.63

EasyVersion 2.01 Released

Icetips Clarion 6 Compatibility

Clarion Third Party Profile Exchange Updated

ABCFree Templates And Tools Updated

CPCS Builds v5.16 (C5b) And v5.57h (C55)

Special Offer From Berthume Software And Epsilon Concepts!

Ingasoftplus EasyResizeAndSplit 1.01

1st Logo Design

C6EA2 Patch Released

EasyResizeAndSplit 1.00

xDigitalClock v1.5

PDF-Tools New Features

Handy Tools FTP/HTTP

ProDomus Updates C55 Translator Plus

gReg Licensing Change

ProDomus PD Universal Drop Edit Controls

SealSoft xFunction Library v1.7

XPMenu Holiday Schedule

HelpMaker Update.

powerRUN Freeware

Clarion 5.5 MS SQL Server Seminar In Johannesburg

INN Bio And News For 28-Jan-2003

Firebird 1.5 Beta 1

cpTracker Silver Edition Released!

Threading Download At RadFusion

Compression Algorithm (Part 2)

In her latest installment, Alison Neal discusses the Huffman compression algorithm, which is the same compression algorithm that is used by PkZip. The algorithm yields approximately 40% compression for text files. The test application included with the article reduces the provided test file from 20kb to 12kb in size, and then decompresses it back to its original state. Part 2 of 2.

Posted Tuesday, February 25, 2003

Debugging Queues With Excel

Like a lot of Clarion programmers, Alan Telford thinks queues are the greatest thing since sliced bread. But it isn't always easy viewing queue data when you're tracking down a queue data-related bug. To solve this problem, Alan created a procedure that makes it easy to export queue data to Excel, which is an excellent tool for viewing tabular data.

Posted Thursday, February 27, 2003

Book Review: PostgreSQL Developer's Handbook

PostgreSQL is one a handful of popular open-source databases, and arguably the most feature-rich. This hefty book from SAMS does cover day to day operations with SQL, but as the title suggests, devotes more pages to the many ways developers can use, and extend, PostgreSQL's advanced features.

Posted Friday, February 28, 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.

Product Scope 32 PRO Sale Ends Feb 3

ZipApp 1.1b Freeware

SealSoft xDigitalClock v1.4

Search the news archive

Interfaces Everywhere

by Jim Kane

Published 2003-02-03

Some time ago I needed to write code to interact with MS Office products, and I wanted a way to do so using syntax closer to Visual Basic syntax. To fill that need, I created the `OLETCL` class, which relied on a quirk in Clarion's COM and thread handling. Unfortunately, with Clarion 6's new threading model `OLETCL` no longer works. When one door closes another opens, or, said differently, necessity is the mother of invention. When I saw `OLETCL` was officially dead, it prompted me to finish a COM interface generator I'd started many years earlier.

There are two ways to call most COM objects. The first is called late binding or `IDispatch`. That is what the Clarion OLE control uses. It's slow, and because it's part of the Clarion language it's sealed and cannot be easily modified or extended. Personally I hate the OLE control on both counts. Fortunately, almost all COM objects are accessible using early binding. Calling a COM object via early binding is accomplished in Clarion by creating an interface. If you not familiar with interfaces read Phil Will's excellent article on the subject.

Calling a COM object via early binding is really no different than calling a Clarion class. Prior to using a Clarion class you need to create an instance of the class, which may implement one or more interfaces. COM objects work much the same way. To create a COM object and receive the address of an interface on that COM object, you call the Win32 API `CoCreateInstance` function, and then you pass the COM class instance, plus interface identifiers (CLSID and IIDs), to `CoCreateInstance`. After creating the class or object, you can call any method on one or more of its interfaces.

Each call to an interface method is very much like calling a Win32 API function. The steps involved, whether calling any Win32 API function or a COM method, are: prototype it;

convert the input data to the format expected by the call; make the call; and convert the output to Clarion data types.

The purpose of this article is to introduce a utility I call **RTLib** that creates the prototypes for interface methods and various equates. In addition, a new class called `StdCom2Cl` eases tasks such as calling `CoCreateInstance`, converting data to or from `BString`s, and error handling.

The `StdComCl` which I published earlier in Clarion Magazine is still viable. `StdCom2Cl` is a superset that creates an instance of `StrCl` for conversion of Clarion strings to and from COM `BString`s, and integrates the error handling of `StrCl` and `StdComCl`. It also uses `CallDLLCl` (also previously in ClarionMag) to optionally automatically register COM objects like regsvr32 does. Because of the inclusion of `CallDllCl`, a small assembler file needs to be added to the project or application (**CallA.A**) whenever `StdCom2Cl` is used.

The task

Recently I was asked to import a list of clients into Quick Books Pro. I was aware there was a COM SDK available so I downloaded it from Intuit's developer web site. Once I had downloaded the SDK and read through the documentation, it became apparent the COM code was located in a DLL called **qbfc2.DLL**. Opening that DLL in Oleview showed over 500 interfaces!

The type library displayed by OleView showed all the interfaces, prototypes, and equates, but in typical Microsoft format. What I wanted was the same information in Clarion format. To accomplish that I wrote **RTlib**, which is short for 'read type library.' **RTLib** works by loading a type library with the `LoadTypeLib` API, then calling the `ITypeLib` interface to iterate through the type descriptions. Full source is provided in the accompanying download.

To use this utility, change the first two lines after the `CODE` statement in the source code to specify the input and output, and then compile and run RTLIB.EXE.

```
dumpfilename='qbfc2.inc'
tlibname='c:\program files\common files\Intuit\QuickBooks\qbfc2.dll'
```

When compiled and run, **RTLib** creates a file – **qbfc2.inc** in this case – with a Clarion version of what OleView shows for **qbfc2.dll**. All 543 interfaces and additional constants and equates are included. That's over a megabyte of text! When **RTLib** is done, it displays the count of interfaces generated.

To use the file created add it to a Clarion source module along with `stdcom2cl` like this:

```
include('stdcom2.inc')
include('qbfc2.inc')
```

In order to add clients to QuickBooks you need to connect with QuickBooks, create a request, add the customer information to the request, then send the request to QuickBooks, and process the response. I derived a new class from `Stdcom2cl`, called `QuickBCl`, to perform all these tasks. The complete code needed to add the customer to QuickBooks is as follows:

```
if ~QuickBCl.init(eDebugon, Appid, appname, companyfile, OMDontCare) then
  !do stuff
  if ~QuickBCl.StartRequest(ROEContinue) then
    loop
      !this is where you get creative
      if QuickBCl.CustomerAddRequest() then break.
      !all done so exit
      break
    end
    !process the requests if any; process the ret object
    QuickBCl.ProcessRequest(1)
  end
end
QuickBCl.kill()
Message('All done')
return
```

I'll go through this code line by line so you can see what actually happens.

The Init method

First, the init method initializes COM and creates the QuickBooks COM object:

```
!intialize COM
SELF.initcom(pDebugMode)
!create the quick books session manager COM object
SELF.IQBSessionManager&=SELF.getinterface(|
  address(CLSID:QBSessionManager),address(IID:IQBSessionManager))
```

Notice that the `CLSID` and `IID` constants are generated by **RTLib** and found in **abfc2.inc** along with the `IQBSessionManager` interface definitions.

In the QuickBCl class definition the interface is defined as:

```
IQBSessionManager       &IQBSessionManagerType
```

This compiles because `IQBSessionManagerType` is defined in **QBFC2.Inc** generated by **RTLib**. **RTLib** appends the word '`Type`' to interface definitions. Thus if the COM object you are using **RTLib** on contains an interface called `IDoSomething`, then **RTLib** will generate `IDoSomethingType`, which can be used as follows:

```
IDoSomething  &IDoSomethingType
Code
  IDoSomething&=(address of the IDoSomething interface)
  !Call IDoSomething an IDoSomething method such as DoIt:
  If IDoSomething.DoIT() <0 then
    Message('IDoSomething Error)
  end
!And finally release the interface:
IDoSomething.Release()
```

Every COM interface has a release method which needs to be called when you are done with the interface. All interface methods return a long commonly called a `HResult`. The `StdComCl.TakeHR()` method checks the `HResult` for error, converts the long to a string, and stores the error description in the `StdComCl.ErrorStr` member variable.

The `IQBSessionManager` interface is initially `null`, but `GetInterface` returns the address of the interface which is then stored in `SELF.IQBSessionManager`. In the `kill` method, if the interface is not `null`, it is released. Tracking all interfaces obtained as `IQBSession`, and eventually releasing them, is important. Failure to do so may result in a memory leak.

The `~QuickBCl.init` method receives several strings: `AppID`, the application name, and the QuickBooks Company file name. These must all be converted to `BString`s. Please consult the QuickBooks SDK for details of these constants. `StdCom2Cl` uses an instance of `StrCl` to do the conversion, and in case of error sets the `ErrorStr` member variable of `StdCom2Cl` with an appropriate message:

```
!convert input strings to bstrings
if SELF.tobstr(pappid,lpBstr1) or|
   SELF.tobstr(pappname,lpbstr2) or |
   SELF.tobstr(pCompanyfile,lpBstr3) then
  do procret
end
```

Next, two methods on the `IQBSessionManager` interface are called to connect with QuickBooks:

```
if ~SELF.TakeHR(SELF.IQBSessionManager.OpenConnection(|
  lpBstr1,lpBstr2),'OpenConnection') and |
  ~SELF.TakeHR(SELF.IQBSessionManager.BeginSession(|
  lpbstr3,pShareMode),'BeginSession')
  then  !(companyfile,omDontCare)
    res=return:benign
end
```

Notice no prototyping of the interface is needed since **RTLib** generated the interface prototype. You can view the prototype by searching for the above methods in **QBFC2.INC**.

The TakeHR method of StdCom2Cl does the error handling in case the method call returns an error, with a description stored in StdCom2Cl.Errorstr. If the debug mode is set to true, the error is displayed.

In each method the ProcRet routine does all the freeing of interfaces or BStrings required. All interfaces eventually need to be released by calling the release method (which every interface has). Likewise all BStrings created to be passed into one of the method calls need to be freed by calling the BstrFree() method of StdComCl.

The StartRequest method

The StartRequest method calls the CreateMsgSetRequest method on the IQBSession interface:

```
!create a requestset (IMsgSetRequest)
if SELF.TakeHr(SELF.IQBSessionManager.CreateMsgSetRequest(|
  2,0,lpInterface),'CreateMsgRequest')
  or ~lpInterface then do procret.
SELF.IMsgSetRequest&=(lpInterface)
```

The address returned by this method, lpInterface, is the address of the IMsgSetRequest interface. The line above casts the long into the interface.

The AddRequest method

The CustomerAddRequest method calls the AppendCustomerAddRq method of the IMsgSetRequest interface to get a ICustomerAdd interface. Notice in the generated file (QBFC2.inc) the AppendCustomerAddRq method is shown as:

```
AppendCustomerAddRq Procedure(*LONG lpICustomerAdd),|
  hresult,raw,proc !Function[OUT][RETVAL]
```

I find the comment following every interface method indicating whether the parameter is supplying ([IN]) or returning ([OUT]) very helpful information in understanding how to use the method. In C/C++ things like **void or a pointer to a pointer are allowed. Clarion does not have a direct equivalent but in general a long can be used. The only remaining item to decide is if it should be *long or long, and that is where the [OUT] or [IN] comment can be very helpful. While RTLIB generally gets it right, the [IN] or [OUT] designation is a good final sanity check.

If an interface has a property such as NewMessageID, the Get and Put methods are generated as follows:

```
PutOldMessageSetID Procedure(bstring pOldMessageSetID),|
```

```
  hresult,raw,proc !PropPut[IN]
GetOldMessageSetID Procedure(*bstring OldMessageSetID),|
  hresult,raw,proc !PropGet[OUT]
```

RTLib prepends `Get` or `Put` to the property name. The comment indicates is a property and clearly shows the direction of information flow.

Now back to calling `AppendCustomerAddRq`. As the comment above indicates, this method returns a `long`.

```
!Get the customerAdd interface
if SELF.TakeHR(SELF.IMsgSetRequest.AppendCustomerAddRq(lpInterface),|
  'AppendCustomerAddReq') or ~lpInterface then do procret.
ICustomerAdd &= (lpInterface)
!Now set the customer name
if SELF.TakeHR(ICustomerAdd.GetName(lpInterface),'GetName') |
  or ~lpInterface or SELF.SetStringValue(lpInterface,'TestCustomer1') |
  then do procret.
```

The `procret` routine contains typical code to release the `ICustomerAdd` interface:

```
IF ~ICustomerAdd&=null THEN
  ICustomerAdd.Release()
  ICustomerAdd&=NULL
end
```

The ProcessRequest method

Finally the `ProcessRequest` method sends the request to QuickBooks by passing the `IMsgSetRequest` interface by address:

```
!SEND THE REQUEST
IF SELF.TakeHr(SELF.IQBSessionManager.DoRequests(|
  address(SELF.IMsgSetRequest), lpInterface),|
  'DoRequests') or ~lpInterface then do procret.
IMsgSetResponse&=(lpInterface)
```

QuickBooks has rather elaborate error handling code. The remainder of the code loops through the results returned by QuickBooks and processes any errors reported. After all the error processing, the `ClearOldMessageSetID` class method is called to let QuickBooks know the response has been processed. Without that step QuickBooks stores previous results so they can be reprocessed in case of power loss or GPF during the initial processing. The code presented here does not fully implement the QuickBooks error recovery system, but the Quick Books SDK describes it in detail for those interested.

The Kill method

When all is completed the `Kill` method is called to clean up and uninitialize COM. First the `IMsgSetRequest` interface is released if not null, and then the connection with QuickBooks is closed:

```
!release a pending MsgSetRequest that was
! not processed - IMsgSetRequest would be
! non-null if error setting up the request
! after creating it.
if ~SELF.IMsgSetRequest&=NULL then
  SELF.IMsgSetRequest.Release()
  SELF.IMsgSetRequest&=NULL
end
!close session/connection and release the section manager
if ~SELF.IQBSessionManager&=NULL then
  SELF.TakeHR(SELF.IQBSessionManager.EndSession(),'EndSession')
  SELF.TakeHR(SELF.IQBSessionManager.CloseConnection(),'CloseConnection')
  SELF.IQBSessionManager.Release()
  SELF.IQBSessionManager&=NULL
end
!uninitialize COM
SELF.killCom()
```

Summary

The purpose of this article is not to show how to use the QuickBooks SDK, but to demonstrate the power of **RTLib** to generate all the interface prototypes and equates needed for a complex COM object such as the one for QuickBooks, and to show how to call those interfaces using `StdCom2Cl`. While **RTLib** may not handle every possible data type especially data types not supported by Clarion such as signed bytes and 64 bit integers, it has handled everything I've thrown at it so far. The source is provided in case modifications or extensions are needed.

Download the source

Jim Kane was not born any where near a log cabin. In fact he was born in New York City. After attending college at New York University, he went on to dental school at Harvard University. Troubled by vast numbers of unpaid bills, he accepted a U.S. Air Force Scholarship for dental school, and after graduating served in the US Air Force. He is now retired from the Air Force and writing software for ProDoc Inc., developer of legal document automation systems. In his spare time, he runs a computer consulting service, Productive Software Solutions. He is married to the former Jane Callahan of Cando, North Dakota. Jim and Jane have two children, Thomas and Amy.

Reader Comments

Add a comment

Interesting stuff. Ok, Im dense. What provoked this...
I was born in New York City - not exactly a bucolic...

Introducing PostgreSQL - Creating Tables And Sequences

by David Harms

Published 2003-02-06

In Part 1 of this series I introduced the PostgreSQL open source database, and showed how to install the native Windows beta version. In this article I'll continue with some basic database operations, including creating tables and using sequences, which are PostgreSQL's version of server-side autoincrementing keys.

As the native Windows open source version of PostgreSQL is still in beta, I'll be using a Linux installation for the following discussion. The latest word is that the native Windows version of will be in PostgreSQL 7.4 (the current stable version is 7.3.1). I won't cover Linux installation as there's plenty of documentation available at the PostgreSQL web site.

Setting up the database

In the last episode I ran into a bunch of trouble getting a working user name and password for the Windows version of psql, the command line interface to PostgreSQL (although not everyone who's run the beta has had the same difficulty). On Linux, if you're administering the server, you can log in as root and assume the postgres user's identity with the su command:

```
[root@ns root]# su postgres
bash-2.05$ createuser demo
Shall the new user be allowed to create databases? (y/n) n
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
bash-2.05$ createdb demo
CREATE DATABASE
bash-2.05$
```

If you don't have root access, then whoever does have root access can set up your database access for you.

Notice that in my example above the user and the database have the same name. This isn't absolutely necessary, but psql will default to the currently logged in user name for both the psql user and the database name. This makes for easy management where you want to give each logged in user their own PostgreSQL database.

Connecting

Now that I have created a demo database, and a demo PostgreSQL user, I can either log in to Linux as user demo and execute

```
psql
```

or as another user I can enter:

```
psql –U demo –d demo
```

In either case I'll get the following greeting:

```
Welcome to psql, the PostgreSQL interactive terminal.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

demo=>
```

You can quickly check the version of PostgreSQL you're running by calling the `version()` function:

```
demo=> select version();
                    version
-------------------------------------------------
 PostgreSQL 7.2.1 on i686-pc-linux-gnu, compiled by GCC 2.96
(1 row)
```

As you can see I'm one dot release behind the times. You can also check the `psql` client version with either the `--version` or the `-V` option:

```
# psql --version
psql (PostgreSQL) 7.2.1
contains support for: readline, history, multibyte
Portions Copyright (c) 1996-2001, PostgreSQL Global Development Group
Portions Copyright (c) 1996, Regents of the University of California
Read the file COPYRIGHT or use the command \copyright to see the
usage and distribution terms.
[root@ns admin]#
```

Creating tables

Figure 1 shows the database I'll be creating in this installment. These are the tables that I currently use to store Survey data for Clarion Magazine, using MySQL. The titles are fairly self-explanatory.

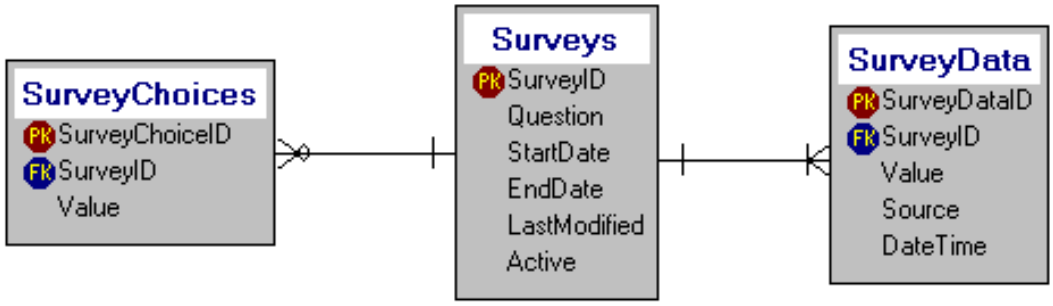


Figure 1. The Surveys database diagram

Here's a `CREATE TABLE` statement for Surveys:

```
CREATE TABLE Surveys(
SurveyID serial NOT NULL PRIMARY KEY,
Question varchar(255),
StartDate date,
EndDate date,
LastModified timestamp,
Active bool DEFAULT False);
```

When I paste this statement into `psql` and execute it, I get the following result:

```
NOTICE:  CREATE TABLE will create implicit sequence
  'surveys_surveyid_seq' for SERIAL column 'surveys.surveyid'
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit
  index 'surveys_pkey' for table 'surveys'
```

Two additional structures have been automatically created, both because of this line in the declaration:

```
SurveyID serial NOT NULL PRIMARY KEY,
```

The second item is the index for the primary key, which is no surprise. But `SurveyID` also is my autoincrementing unique identifier for each row, as indicated by the `SERIAL` data type. `SERIAL` values are stored in eight byte integers, with a maximum value of 9223372036854775807 (big enough for ya?) unless the platform compiler doesn't support eight byte integers, in which case the maximum value is the same as a Clarion LONG. A `SERIAL` column causes PostgreSQL to create a sequence number generator, called simply a

sequence, which is used to automatically increment that value. Here's the description of the table as shown by `psql` after I issued the `CREATE`:

```
demo=> \d surveys;
                        Table "surveys"
    Column    |           Type           |Modifiers
--------------+--------------------------+---------------------------------
 surveyid     | integer                  | not null default nextval(
                                            '"surveys_surveyid_seq"'::text)
 question     | character varying(255)   |
 startdate    | date                     |
 enddate      | date                     |
 lastmodified | timestamp with time zone |
 active       | boolean                  | default 'f'::bool
Primary key: surveys_pkey
```

As you can see the default value for `surveyid` references `surveys_surveyid_seq`. Sequences are actually single-row tables (think control files) with special associated functions: `setval()`, which lets you change the current value in the row; `nextval()`, which increments the current value (and first adds the row if the sequence has never been used before) and returns that value; and `currval()`, which returns the value last set by `nextval()` or `setval()` *for the current process*. This last point is an important one. If, for instance, I add the very first `surveys` record, `currval('surveys_surveyid_seq')` will return 1 even if another process (typically another user) inserts a `surveys` record after I inserted the first one, and before I called `currval()`. (Next time I'll show you to call `currval()` from within a Clarion application to get autoincremented IDs when adding child records.)

The fact that sequences are actually tables gives you some flexibility in autoincrementing. You can create them separately from your tables and with specialized attributes, including reverse order, increment steps other than 1, and wrap-around. Here's a sequence that starts at 4, decrements by two, and when it hits zero starts over at 10 again.

```
CREATE TEMPORARY SEQUENCE increment_test
  INCREMENT -2 MINVALUE 0 MAXVALUE 10
  START 4 CYCLE;
```

Repeatedly calling `SELECT nextval('increment_test')` on this sequence yields the following numbers:

```
4 2 0 10 8 6 4 2 0 10 8
```

… and so on. Also note that I've used the `TEMPORARY` qualifier on this sequence – once the session is closed this sequence will be deleted. You can specify the name of an existing sequence when creating a temporary sequence. In that case the temporary sequence will replace the permanent sequence for the duration of the session. You would, of course, want to

use something like this *very* carefully.

Why would you want a sequence to cycle? As Carl Barnes pointed out to me after reading the draft of this article, one application would be a Job/Order number that you don't want to exceed, say, five digits. Of course that would assume that you're archiving/deleting old orders so they can't cause duplicate key errors.

Because sequences are separate entities from tables, there is one side effect you might not expect. If you subsequently issue a `DROP TABLE` command, as in:

```
DROP TABLE Surveys;
```

the table and any indexes will be deleted, but the sequence will *not* be deleted. If you reissue the `CREATE TABLE` command, you'll get an error like this one:

```
ERROR: Relation 'surveys_surveyid_seq' already exists
```

So what do you do? Issue a `DROP SEQUENCE`:

```
demo=> DROP SEQUENCE surveys_surveyid_seq;
```

Now you can create the table, and its associated sequence will also be created.

Going back to the table description reported by PostgreSQL, you'll notice a `::` operator used in two places. This is the cast operator, and it first shows up in the primary key column that uses the sequence:

```
surveyid | integer | not null default
  nextval('"surveys_surveyid_seq"'::text)
```

Since PostgreSQL supports function overloading, it's conceivable that there could be versions of the `nextval()` function which take parameters of data type other than text. The ::text operator ensures that the parameter to `nextval()` is interpreted as text. Seems a bit paranoid, but there you are.

There is also a cast on the boolean field named active, which is cast to a boolean value (`'f'::bool`). In PostgreSQL, the possible true values for a boolean are `TRUE`, `'t'`, `'true'`, `'y'`, `' yes'`, and `'1'`, while possible false values are `FALSE`, `'f'`, `'false'`, `'n'`, `'no'`, and `'0'`. The cast `'f'::bool` ensures that the default value is in fact boolean.

Because sequences are independent of the table they don't necessarily function the same way as a client-side autonumber, where the code looks at the highest value in the primary key, increments, and tries to add the placeholder record. For instance, imagine a sequence that's

been used just once. You can see the sequence data with a SELECT statement like this:

```
demo=> select * from surveys_surveyid_seq;
    sequence_name     | last_value | increment_by |      max_value
----------------------+------------+--------------+---------------------
 surveys_surveyid_seq |          1 |            1 | 9223372036854775807

| min_value | cache_value | log_cnt | is_cycled | is_called
+-----------+-------------+---------+-----------+-----------
|         1 |           1 |      32 | f         | t
```

The sequence has a last_value of 1 because I've already added a record in the surveys table, without specifying a value for the surveyid column. Here's the resulting data:

```
demo=> select * from Surveys;
 surveyid |   question   | startdate  |  enddate   | lastmodified | active
----------+--------------+------------+------------+--------------+--------
        1 | First survey | 2003-02-03 | 2003-03-05 |              | f
```

Now I add a record specifying a primary key value of 4:

```
INSERT INTO surveys (surveyid,question,startdate,enddate)
values(4,'Fourth survey',now(),now()+30);
```

There are now two records in the table:

```
demo=> select * from surveys;
 surveyid |   question    | startdate  |  enddate   | lastmodified | active
----------+---------------+------------+------------+--------------+--------
        1 | First survey  | 2003-02-03 | 2003-03-05 |              | f
        4 | Fourth survey | 2003-02-03 | 2003-03-05 |              | f
(2 rows)
```

The sequence, however, remains unchanged because a value was supplied for the surveyid field:

```
demo=> select * from surveys_surveyid_seq;
    sequence_name     | last_value | increment_by |      max_value
----------------------+------------+--------------+---------------------
 surveys_surveyid_seq |          1 |            1 | 9223372036854775807

 | min_value | cache_value | log_cnt | is_cycled | is_called
 +-----------+-------------+---------+-----------+-----------
 |         1 |           1 |       0 | f         | t
```

So what happens now? If I continue adding records, I'll end up with a duplicate key error when I hit surveyid 4:

```
demo=> INSERT INTO surveys (question,startdate,enddate)
values('Second survey',now(),now()+30);
```

```
INSERT 506966 1
demo=> INSERT INTO surveys (question,startdate,enddate)
values('Third survey',now(),now()+30);
INSERT 506967 1
demo=> INSERT INTO surveys (question,startdate,enddate)
values('Fourth survey',now(),now()+30);
ERROR:  Cannot insert a duplicate key into
  unique index surveys_pkey
demo=> INSERT INTO surveys (question,startdate,enddate)
values('Fifth survey',now(),now()+30);
INSERT 506969 1
```

Although the attempt to insert the record with `surveyid` 4 failed, the sequence did increment, so subsequent inserts will again work. The moral of the story, however, is that you really have to be careful if you import a bunch of records with existing primary key values. You might want to set the sequence number to a value higher than any of the primary key values you're importing, leaving a window for the existing values. All subsequent calls to `nextval()` will start at that number plus one. You do this with the `setval()` function:

```
demo=> select setval('surveys_surveyid_seq',1000);
 setval
--------
   1000
(1 row)
```

And the result:

```
demo=> select * from surveys_surveyid_seq;
    sequence_name     | last_value | increment_by |      max_value
----------------------+------------+--------------+---------------------
 surveys_surveyid_seq |       1000 |            1 | 9223372036854775807

 | min_value | cache_value | log_cnt | is_cycled | is_called
 +-----------+-------------+---------+-----------+-----------
 |         1 |           1 |       0 | f         | t
```

One curiosity of sequences is that the minimum value of any sequence is 1, so you can't issue a `setval('sequencename',0)`; if you set the current value to 1, then the next available value for any serial data type is 2. That means you can't reset an existing sequence so it will start at 1 – you have to drop the sequence and recreate it. At least that's been my experience.

The SurveyChoices table

As shown in Figure 1. the Surveys database includes a table for survey choices:

```
CREATE TABLE SurveyChoices(
SurveyChoiceID serial NOT NULL PRIMARY KEY,
SurveyID int NOT NULL,
Sequence decimal(5,2) DEFAULT 1,
Value varchar(100),
```

```
FOREIGN KEY (SurveyID) REFERENCES Surveys (SurveyID)
ON DELETE CASCADE
ON UPDATE CASCADE);
```

And just to speed things up, here are a couple of indexes:

```
CREATE INDEX SurveyChoices_Idx_ID
  ON SurveyChoices (SurveyID);
CREATE INDEX SurveyChoices_Idx_ID_Seq
  ON SurveyChoices (SurveyID,Sequence);
```

`SurveyChoices` is linked to the `Surveys` table via the `SurveyID` column. Here's the output from the `CREATE` statement:

```
NOTICE:  CREATE TABLE will create implicit sequence
   'surveychoices_surveychoicei_seq' for SERIAL
   column 'surveychoices.surveychoiceid'
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit
   index 'surveychoices_pkey' for table 'surveychoices'
NOTICE:  CREATE TABLE will create implicit trigger(s)
  for FOREIGN KEY check(s)
```

As before, the `CREATE` statement has resulted in the creation of an implicit sequence for the `SurveyChoiceID` column, as well indexes for the primary key and for the foreign key. The `CREATE` statement also results in the creation of two triggers for the specified `ON DELETE CASCADE` and `ON UPDATE CASCADE` foreign key checks. These mean that if you attempt to delete a `Surveys` record, and there are related child `SurveyChoices` records, those child records will be deleted. Similarly if the parent primary key value changes, these changes will be rippled down to the child records (but that's irrelevant, because you always use values for primary keys that will never need to be changed, right? Right!).

So how do you find out what triggers are in place for an existing table? The answer lies in the system tables, and to get a listing of those you type `\dS`:

```
demo=> \dS
               List of relations
           Name            |   Type   |  Owner
---------------------------+----------+----------
 pg_aggregate              | table    | postgres
 pg_am                     | table    | postgres
 pg_amop                   | table    | postgres
 pg_amproc                 | table    | postgres
 pg_attrdef                | table    | postgres
 ... Approx. 40 tables omitted for brevity
 pg_type                   | table    | postgres
 pg_user                   | view     | postgres
 pg_views                  | view     | postgres
 pg_xactlock               | special  | postgres
```

These system tables store everything from databases to tables to functions, check constraints, data types, and yes, triggers. It would've taken me forever to find out how to get a trigger listing from this database, but happily I found one on the web, posted by Michael Fork in the comp.databases.postgresql.general newsgroup on January 17, 2001:

```
SELECT pg_trigger.tgargs, pg_trigger.tgnargs,
pg_trigger.tgdeferrable, pg_trigger.tginitdeferred,
pg_proc.proname, pg_proc_1.proname FROM pg_class pg_class,
pg_class pg_class_1, pg_class pg_class_2, pg_proc pg_proc,
pg_proc pg_proc_1, pg_trigger pg_trigger, pg_trigger
pg_trigger_1, pg_trigger pg_trigger_2
WHERE pg_trigger.tgconstrrelid = pg_class.oid
AND pg_trigger.tgrelid = pg_class_1.oid
AND pg_trigger_1.tgfoid = pg_proc_1.oid
AND pg_trigger_1.tgconstrrelid = pg_class_1.oid
AND pg_trigger_2.tgconstrrelid = pg_class_2.oid
AND pg_trigger_2.tgfoid = pg_proc.oid
AND pg_class_2.oid = pg_trigger.tgrelid
AND ((pg_class.relname='<<PRIMARY KEY TABLE>>')
AND (pg_proc.proname Like '%upd')
AND (pg_proc_1.proname Like '%del')
AND (pg_trigger_1.tgrelid=pg_trigger.tgconstrrelid)
AND (pg_trigger_2.tgrelid = pg_trigger.tgconstrrelid))
```

Find `<<PRIMARY KEY TABLE>>` in that listing and replace it with the name of your table. For the Surveys table, the output looks like this:

```
tgargs
------------------------------------------------------------------------
\000surveychoices\000surveys\000UNSPECIFIED\000surveyid\000surveyid\000

| tgdeferrable | tginitdeferred |       proname       |       proname
+--------------+----------------+---------------------+---------------------
| f            | f              | RI_FKey_cascade_upd | RI_FKey_cascade_del
```

Fortunately there are tools available to make this kind of database administration easier. I use PostgreSQL Manager from EMS (the same company that produces MySQL Manager, and IB Manager for Interbase/Firebird). I'll say more about PostgreSQL Manager in future articles.

The SurveyData table

Finally, here's the table creation script for the `SurveyData` table, which holds the survey responses:

```
CREATE TABLE SurveyData(
SurveyDataID serial NOT NULL PRIMARY KEY,
SurveyID int NOT NULL,
Value varchar(100) NOT NULL,
Source varchar(30),
DateTime timestamp,
```

```
FOREIGN KEY (SurveyID) REFERENCES Surveys (SurveyID)
  ON DELETE CASCADE ON UPDATE CASCADE);
```

And here's the result:

```
NOTICE:  CREATE TABLE will create implicit sequence
  'surveydata_surveydataid_seq' for SERIAL column
  'surveydata.surveydataid'
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit
  index 'surveydata_pkey' for table 'surveydata'
NOTICE:  CREATE TABLE / UNIQUE will create implicit index
  'surveydata_surveydataid_key' for table 'surveydata'
NOTICE:  CREATE TABLE will create implicit trigger(s)
  for FOREIGN KEY check(s)
```

Add one index to make retrieving the survey data a bit more orderly:

```
CREATE INDEX IDX_SurveyData_Survey_Value ON SurveyData(SurveyID,Value);
```

Summary

Table creation isn't that much different in PostgreSQL as compared to other SQL databases, although its use of sequences to handle autonumbered columns is worth some special attention. Next time I'll provide some demo data, and show how to use the psqlODBC driver to connect to a PostgreSQL database with Clarion.

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

Here's an updated link to installation instructions for...
You can find a commercial Windows version of PostgreSQL at...
While trying to access...
Vaidya, I found an email address at the referring page...
Vaidya, I found an email address at the referring page...

Clarion Magazine

Topical Index

Published 2001-11-12

Topics > CLARION 6

- C6 COM - 1 article(s)
- C6 General Info - 1 article(s)
- C6 Threading - 4 article(s)

Reader Comments

Add a comment

I've been waiting for an index like this. When will you...
Dave, This is great! I looked at all the articles...
Excellent - thank you!
Is there anyway to include a topic about API !!! Coz I am...
I'm trying to create an Invoice Report (1 Parent- 1 child)...
What a PAIN I used a whole afternoon persuing all the...
Dermot, Thanks, I'm delighted to have been the cause of...
Oops - I guess it's no longer undocumented.
Okay-I've just acquired an 8-year established software...
Is there anywhere I can buy clarion book? Thanks
Curtis - Mitten Software sells the Clarion Companion...

A Class For Tagging

by Steven Parker

Published 2003-02-13

> *Oh, how the mighty have fallen! Steve didn't feel the need. He didn't see a reason. Classes offered nothing he couldn't do using tried and true techniques. Now, the good doctor succumbs to the siren call of OOP and makes his first class. Appropriately, perhaps, this article appears 10 years after his first article, "A Better Still Validation Template," in the Clarion Tech Journal for March/April 1993.*

I am on record as feeling that marking, as implemented in the browse template, is massively underwhelming (List Box Marking). Tagging by storing a reference to a record, in a queue or in a file, is and has been my preferred method since forever and is all but bulletproof.

I am also on record as not having felt the need to create classes. In Reports: OOP, ABC and Ignoring Templates (Part 1) I went so far as to state:

> I *use* OOP … but I do not *do* OOP. I have not created a single class, much less a class library. I do have an occasional attack of "I really should learn this stuff." But, that would require doing, and I haven't been *that* tempted….
>
> I just haven't yet felt the need. I have yet to find anything I need to do or want to do that I cannot do quite easily with my tried and true techniques of incorporating frequently used procedures in DLLs and INCLUDE-ing blocks of frequently called code.

Well, "I really should learn this stuff" finally got the better of me. I finally "felt the need" and I converted the tagging code presented in List Box Marking to a class.

What finally motivated me to take this drastic, nay, radical step? Well, that's easy. In fact, it was simple laziness.

In the past, I compiled my tagging and time routines into LIBs and DLLs. This required an EXP file in addition to the functional code. Because I was too lazy to figure out how to create the export file manually, I used an .APP. An .APP is very convenient but that linked in all the globals (ABC or Legacy, 16 bit, 32 bit – I was maintaining both 16 and 32 bit for a while). So, I created a set of PRJs and used the source files generated by the .APP.

No wonder I didn't regenerate them very often.

A class offers the opportunity to be rid of export files, project files and multiple LIB/DLL files. As I said, laziness got me to feel the need to try to write a class.

The INC File

While creating a class (albeit from code that I've been using for almost 10 years), I realized a couple of interesting things about the INC file.

First, a class' INC file is just a prototype file, not significantly different than INC files in CDD (often used with DLLs). The main difference is that these INC files can also contain data declarations (inside or outside the class) in addition to method (procedure) prototypes.

Second, unless you want to jump through *all* kinds of hoops, you need to use the form of prototype that specifies the data type and label together.

The type of tagging I do is queue based, so the INC file includes a queue declaration (before the class proper) and a reference to a queue of that type in the class declaration (explained in CLASSy ASCII File Importing). I still haven't gotten my head around reference variables but, from examining existing class code, I know I need to do this:

```
!ABCIncludeFile
OMIT('_EndOfInclude_',_tagging_)
_tagging_ EQUATE(1)
shpTagQueue   Queue,Type
Ptr             String(1024)
              End
shpTagClass  CLASS,TYPE,MODULE('shpTagging.clw'), ¿
  LINK('shpTagging.clw',_ABCLinkMode_),DLL(_ABCDllMode_)
TagQueue        &shpTagQueue
Construct       Procedure
Destruct        Procedure
IsTagged        Procedure(String pPoint),Byte
MakeTag         Procedure(String pPoint)
ClearTag        Procedure(String pPoint)
ClearAllTags    Procedure
```

```
NumberTagged    Procedure(),Long
             End
_EndOfInclude_
```

As I explained in List Box Marking, I use the queue to store unique information about the tagged record. With this unique information, I can determine if the record has been tagged. I can also use the information to retrieve the actual record. I can even use the unique information as a report, process or browse filter.

Note that my queue has only one element, `Ptr`, a string. Using a long string here allows me a great deal of flexibility.

I can use a string to store `Position()` information. Relying on Clarion's automatic type conversion, I can also use a string to store an auto numbered system ID (usually a long), a `Pointer()`or any other information that uniquely identifies a record. (Note that TopSpeed and Softvelocity are not keen on pointers for TPS files and there are cases where `Pointer(`*`file`*`)` can be unreliable – check the file driver specifications for the file type you are using.).

The remainder of the `INC` file is just the prototypes of functions (class methods) that actually tag and untag records.

Construct and Destruct: As discussed in CLASSy ASCII File Importing, these methods create the queue – because a class cannot contain a structure but can contain a reference to one -- and clean it up on exit (`Free` and `Dispose`).

I don't have to call these methods – Construct and Destruct are "magic" method names; these methods are automatically called whenever an object is created or destroyed, respectively. (Those of you who've been around Clarion a while may remember a Clarion goal, words to the effect of "if it always needs to be done, the developer should never have to do it.")

IsTagged: This function returns True if a record is tagged, False otherwise. This function is used in filters and in conditional list box icons (see the demo app and Icons in List Box).

MakeTag: Tags a record if it is not already tagged (i.e., adds it to the queue).

ClearTag: Untags a record, if it is tagged. ClearTag removes a record reference from the queue.

ClearAllTags: Does just what its label implies.

NumberTagged: Returns how many records are tagged.

Creating the Code File

The code for these functions is somewhat changed from my first presentation of them. Note that return data types *do not* get included in the prototype line (the compiler gets most unhappy if you do).

Because I was using existing, well tested code, I thought my CLW file would be little more than my old code.

Wrong.

Not prepending the class name to each method prototype was deadly. The compiler could not resolve the prototypes without the class name.

Also, note how the queue is referred to by its reference (instantiated) label, not its type declaration. And only dot syntax works in class code:

```
  MEMBER
  omit('***',_c55_)
_ABCDllMode_  EQUATE(0)
_ABCLinkMode_ EQUATE(1)
  ***
   Include('shpTagging.inc')
  Map
  End


shpTagClass.Construct   Procedure
   CODE
   Self.TagQueue &= New(shpTagQueue)

shpTagClass.Destruct     Procedure
   CODE
   Free(Self.Tagqueue)
   Dispose(Self.TagQueue)

shpTagClass.IsTagged     Procedure(String pPoint)!,Byte
   CODE
   Self.TagQueue.Ptr = pPoint
   Get(Self.TagQueue,Self.TagQueue.Ptr)
   If ~ErrorCode()
     Return(True)
   Else
     Return(False)
   End

shpTagClass.MakeTag      Procedure(String pPoint)
   CODE
   Self.TagQueue.Ptr = pPoint
   Get(Self.TagQueue,Self.TagQueue.Ptr)
   If ErrorCode()
```

```
      Add(Self.TagQueue,Self.TagQueue.Ptr)
   End

shpTagClass.ClearTag     Procedure(String pPoint)
   CODE
   Self.TagQueue.Ptr = pPoint
   Get(Self.TagQueue,Self.TagQueue.Ptr)
   If ~ErrorCode()
      Delete(Self.TagQueue)
   End

shpTagClass.ClearAllTags Procedure
   CODE
   Free(Self.TagQueue)

shpTagClass.NumberTagged Procedure()!,Long
   CODE
   Return(Records(Self.TagQueue))
```

If you examine the code, you will not see anything even vaguely resembling rocket science. It's all just standard queue manipulation code. But, like functions in a DLL, it is useable anywhere the object is in scope. See the sample app, where most of these functions are used.

Implementing shpTagClass

I realize that global data is bad and that, in upcoming releases of Clarion, global data is likely to cause an increasing number of problems. However, I do not see the logic in creating a tag queue at anything less than the global level.

If a tag queue is declared at the procedure level, tagging information can only be used in that procedure. But, users are usually tagging records for batch processing (in processes and reports) elsewhere, in other procedures.

Demoting the class instantiation to the module level, forces all potential user-procedures to be in the same module. This would, I think, be a major design restriction.

For the same reason, you will note that neither `shpTagQueue` nor `TagQueue` is threaded. If I put the `Thread` attribute on the queue, any procedure that might use tagged records would have to be on the same thread. I couldn't `Start()` any procedure that *might* use a tag. On its face, this would be another major restriction.

So, in a global data embed, I instantiate `shpTagging` class in the application data area and create a tag object:

```
Include('shpTagging.inc'),Once
Arnold          shpTagClass
```

"Arnold" may seem a somewhat less than descriptive object name (in the sample app, it is used

to tag customer records). But, it illustrates my ability to create multiple tag objects (either on the same file or on different files):

```
Include('shpTagging.inc'),Once
Arnold          shpTagClass
Annie           shpTagClass
```

and I have two sets of tags, two tag queues that know nothing about each other.

In the sample app, there are two browses. Open them, create some records and tag some of them. You will see that the items tagged in one browse are not *ipso facto* tagged in the other.

A more realistic use of this is to create tags for each of your files is:

```
Include('shpTagging.inc'),Once
CustTags        shpTagClass  !Tags for Customer file
InvcTags        shpTagClass  !Tags for Invoices
InvtTags        shpTagClass  !Tags for Inventory
!etc.
```

Neither am I concerned about tags set by multiple users "inheriting" tags set by others, since I'm storing the data in a queue, not in a (potentially) shared file.

Summary

This is hardly a full, or even decent, implementation of tagging. A proper implementation requires a template (for example, to populate the various buttons and the code that makes them work). For this very reason, I have been using Mike Hanson's SuperTemplates (actually, I started using them when they were CPD models). But, not only is the `TagClass` adequate for limited uses, it provides a good test and learning case (especially because you can compare the class to the traditional method of coding, as shown in List Box Marking).

More importantly, I no longer need to recompile my tagging code with new Clarion releases.

Download the source

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since 1993.

Reader Comments

Add a comment

Steve: I like this article. I don't know if this...
Jim could you point out where in the class code a global...
I was confused by the following: I realize that global...
A reference to the class could be passed to a report or...

CONVIC - An Antipodean Clarion Gathering

by Chris Livingstone

Published 2003-02-14

If you're not an Australian Clarion developer then there's a good chance you haven't heard of ConVic. Does this matter, you ask? Well, ConVic's been on the Australian landscape for five years now and for the last four, it's been the only Clarion conference in Australia. What's more, ConVic 2003 is coming up at the end of March – it's an opportunity not to be missed by Australian developers, and overseas developers who might like to consider combining the conference with a sightseeing trip down under.

CONVIC is a Clarion conference held annually in Victoria, Australia's second smallest state in area, the second largest in population. In case you're wondering, CONVIC stands for [Clarion] CONference in VICtoria; of course it's also a sly tip of the hat to Australia's past as a colony for criminals exported from England.

For most of the 1990s, Australian Clarion conferences were held in the heart of Sydney, but the high cost of airfares and accommodation put these Sydney Devcons out of reach for many developers living in other states. The humble aim of the first CONVIC in 1998 was simply to provide an opportunity for all Victorian Clarion developers to get together. The idea was to make the conference as inexpensive and accessible as possible, an idea which spread to developers from South and Western Australia who decided to join us in this venture.

For the last four years, ConVic has been the only Clarion conference in Australia, and so we've had participation from right around the country and even the odd overseas visitor.

CONVICs past

The first CONVIC was held in November 1998 in Ballarat, a large town 100 km west of Melbourne, with its history rooted in the gold rushes of the 1850s. After the success of this

first conference, CONVIC 99 was inevitable. This time we moved right up into the mountains, to Mt Buffalo Chalet in the Victorian Alps. The conference was great, but the fickle spring weather meant that we missed much of the beauty of the mountain. In 2000, we met in Daylesford, a picturesque country town where gentlefolk retired to take the waters of the mineral springs.

Back to Mt Buffalo in March 2001, the end of summer; unfortunately, six months' perfect weather chose to break on the very weekend of our conference! The title "Clarion in the Clouds" was intended to be just a pithy slogan, not a literal description, but thick fog blanketed us the whole weekend limiting visibility outside to a few metres. Luckily there was so much on the program that we had little time for outdoor activities, but those who stayed on an extra day were treated to splendid vistas of the Australian Alps (much of it since devastatingly burnt in the bushfires still burning now).

CONVIC 2001 was great, endorsed by many as the best Clarion conference ever held in Australia. However, anyone who's run a conference of any sort will know just how much work and time is required to make it successful. CONVIC 2001 was no exception and it took its toll. With Clarion use evidently in decline in this country, it seemed that this would be the last of the regional conferences, possibly the last Clarion conference ever in this country. I certainly would never organise another Clarion conference!

But sometimes, you just can’t help yourself, can you?

In this part of the world, Clarion developers are geographically very dispersed and there are very few occasions for catching up with others. Sure we keep in touch over the Internet but it’s not the same as physically getting together.

Someone planned a one-day event in late 2001 but there was considerable disappointment when this was cancelled.

So I relented ever so slightly and decided I'd just "facilitate" something really low-key in 2002. No publicity, no big deal, no organisation, no prizes, no fancy stuff. Just a day for people to get together. All I had to do was tee up a venue and let people get on with it. I'd better organise some food but that’s no big deal. It’d probably be a good idea to get a couple of speakers just to kick things off. Actually, it looks like we’ve got some good stuff here – it’d be a pity to cram it all into one day. OK, so now we’ve got a two-day event. I guess I'd better do something about accommodation. Ah, I've got just the venue. It’s still low-key of course but I'd better work out a bit of a timetable. We’d better have a couple of extra people to fill in some

blank spots. Right, that's it – it's another bloody conference! – CONVIC 2002.

What makes CONVIC special?

Several things make CONVIC special. First of all, it's held "out of town." Deliberately. That mightn't sound like a big deal to overseas developers who live in cities all over the country but down here, in a land of huge empty spaces, most of us live in just a very few big cities. In fact, half of Australia's population lives in just three state capital cities on the eastern sea-board.

Here in Victoria, three-quarters of the population lives in the state capital, Melbourne. However, a significant proportion of our Clarion community lives and works in country and regional areas. So while it makes sense for monthly meetings to be held in the capital city, a weekend conference is a good opportunity to acknowledge and cater for country and regional members by leaving the metropolis. Besides, it's a breath of fresh air for those living in the capital city.

Secondly, our conferences are residential – everyone stays on site. We've eschewed the city hotels and country motels in favour of conference venues which allow participants to relax and socialise. That's why a number of people have taken to referring to our conferences as "retreats," a very appropriate term.

Combining on-site accommodation with an out-of-town location has a further advantage. At the end of the day, participants don't suddenly desert back home to the suburbs, or to their own motels and hotels. Instead, they sit down, relax, go walking, swimming (if the weather's right!) and take the opportunity to catching up with others in an informal environment. Partners join in and become part of the group.

I've always believed strongly that the benefit of conferences goes well beyond the formal content. Of course content is important but equally important is the contact with other developers: sitting round sharing ideas, mixing informally, having a drink, chatting as you walk down to the beach, spending time over lunch, getting to know other developers, get new ideas, make new contacts.

Venues are important. Not just for the conference facilities but also for the environment and that intangible "ambience". If you live and work in the city or the suburbs, it's not much of a

change if you step out of the conference room into bustling city streets and commuter traffic, even if it's a different city. It's much easier to put aside the cares, the pressures, the frustrations, of everyday life when you're in a tranquil rural environment.

Content is obviously central to a conference of any sort. While it's good just coming together every so often, you also need to feel you're getting something useful out of the conference. A good variety of topics is essential but it's also vital to have presenters who can communicate their topic. While I have some leniency with someone who has a lot of technical knowledge to share, I'm always aiming for a professional presentation. From the very first regional conference, CONVIC 98, I ditched the overhead projector and specified PowerPoint for all presenters. A few needed a bit of help at the first conference, but now everyone seems pretty comfortable. It certainly contributes to a much better presentation.

Presentations are also better when they don't go on too long. As you get older, time becomes more precious and we don't like wasting what we have left in this life. (Also, bladders seem to shrink as we get older!) From the start, I made a very strong point of starting and ending sessions on time, even if it meant cutting a speaker short. Feedback shows this has really been appreciated by participants even if speakers aren't always happy. In more recent conferences I've become a little more flexible, but only when there's been time available.

Dave Harms came out as a key speaker for our first conference and established a respect and friendship out here which endures to this day. The experience has not been forgotten by participants nor I suspect by Dave himself. But Dave spoilt us; since then, we've been badly burnt with overseas speakers who've promised but not delivered (or more accurately, they've promised then not turned up). Still, we've proved that we don't need overseas speakers to run a good conference.

Quite a few other things have contributed to the success of the five regional conferences held in this part of the country. One example: seating at meals. This is done by a random chance

allocation. Sorry, but you can't hog a table with all your cronies nor can you wangle it to sit with the keynote speaker at every meal. The flip side is you won't be left out in the cold if you're a newcomer who knows no-one else; you have the same chance as anyone else of sitting with special guests or speakers. Above all, there's every chance that you will meet someone new or discover something different while enjoying your meal.

Is CONVIC successful?

OK, it's different, but is it successful?

If you measure success by attendance, then 30-35 participants probably seems a disappointment. But look at it on a *per capita* basis. Australia is a country with 19 million people (about the same as New York state, a few more than Florida, a few less than Texas) while the US now has around 285 million people. So by this measure, 30-35 to an Australian conference corresponds to 450-500 participants at a US conference. Put like that, you can't complain about attendance. It looks even better when you realise that CONVIC is still primarily a regional conference. It's not bad when you know that you're getting considerably more than 50% of the active Clarion developers in your state!

Feedback from participants leaves no doubt that we've found a winning formula. We haven't had one disappointed participant, no-one who's gone away and said "never again!". Typical reactions are "the best conference so far", "can't wait till next year's conference", plus superlatives enough to embarrass even the most blush-proof.

CONVIC 2002

Despite the low-key intentions, CONVIC 2002, held on March 23-24 2002, proved just as successful as the previous conferences. It was very much a local event with local speakers but that was no disadvantage.

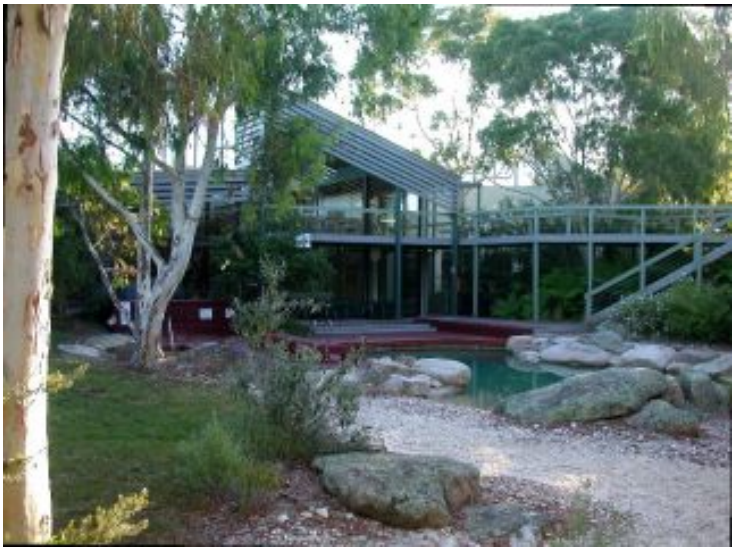
From the weather-cursed mountain tops of the previous conference, I decided to go to the other extreme in 2002, right down to sea-level. To Geelong, a regional centre some 70 km south-west of Melbourne, a waterside city on the very large bay which forms the sea entrance to Melbourne. The conference centre blended in with the surrounding parklands, just five minutes walk from the beach. The change of location worked: we enjoyed fine warm days and balmy evenings. Several people took the opportunity of a swim, most preferring the pool and spa at

the conference centre but more venturesome souls braving the sea water.

Like the previous regional conferences in Victoria, CONVIC 2002 was deliberately residential. After the formal part of the day was over, there was time for people to relax, wander down the beach, take a dip in the pool or spa, use the gym or just sit around chatting, before we all went for a leisurely dinner down by the waterfront. Later in the evening, some played pool or table tennis while the rest enjoyed the casual conversation with other participants and their partners. No distinction between city and country members here – we were nearly all in residence so no-one had to rush off home.

Geelong Conference Centre was a great venue providing good formal conference facilities but also plenty of areas for impromptu gatherings. There was no shortage of food and no one found any need to seek extra sustenance.

Despite the low-key nature of this year's conference, and the emphasis put on people meeting people, the actual content was still substantial. This time we adopted the approach used in some overseas conferences with fewer but longer major presentations.



In fact, there were just three major presentations, in the form of tutorials or "how-to" sessions. The theme became "Clarion + ", or Clarion Plus.

Bruce Cowan had taken on Crystal Reports as part of his work and was therefore an ideal choice for a session on "Clarion + Crystal". Chris Livingstone has spent quite a few months working with ClarioNET and this was the centrepiece of his presentation on "Clarion + 'Net". The third tutorial came out of left field but proved one of the highlights of the conference. Alex McCullie is a long-time Microsoft Access programmer as well as a Clarion developer; in the last couple of years, he has combined these skills by very successfully using Clarion as a front-end to Access and SQL

Server databases. Tutorial number three: "Clarion + MS Access. Each of these sessions led into a wide-ranging discussion on the topic covered with much sharing of ideas and experiences.

We topped up the conference with some less formal but nevertheless valuable contributions from a couple of other stalwarts from this part of the world. Geoff Robinson covered both archiving and BLOBs in his presentation, while we welcomed Simon Brewer back into the fold with the old but still popular favourite review of third-party tools. As a broadening experience, there was also a look at another development environment which has been attracting some interest over the past couple of years. (Perhaps it’s our isolation, more likely just commercial reality, but most developers in this country don’t seem to have a problem recognising there is life outside Clarion.)

In preparation for his session, Simon Brewer contacted ten suppliers of Clarion third-party products, seeking information. I'd like to acknowledge the three who responded; they also generously donated prizes. Thanks to Susan and Arnor at Icetips, Bruce and his team at CapeSoft, and Gus Creces with his rather amazing Handy Tools. So popular was Simon’s first session on the products from these three companies that we had to schedule an extra session.

From every aspect, CONVIC 2002 proved another highly successful conference. The open forum which concluded the conference showed that even with all the Internet communications now available to us, there is still very much a place for face-to-face contact and interaction. The unanimous call for another conference in 2003 in the same place was a very positive affirmation of the success and value of this event; I had little hesitation in booking the dates

and venue for a conference in 2003. "We'll see you next year" was the parting greeting from most participants.

CONVIC 2003

CONVIC 2003 is a three-day conference, from Friday, March 28 to Sunday March 30, 2003, in Geelong, Victoria, Australia. This year we have speakers from all around the country and even a couple of overseas speakers (Gus Creces and Russ Eggen) providing presentations remotely!

If you've never been to a CONVIC, here's your chance. All Australian developers should mark it on their calendars. And we'd be very happy if some overseas Clarion developers could share the experience with us. March-April is a great time to visit this country and with the current exchange rate, it's a great value holiday.

Check us out at www.convic.org.

Chris Livingstone

Grand Poohbah, CONVIC

chris@convic.org

Chris Livingstone has been a teacher for more than half of his working life, and a programmer for some three decades. He began using Clarion in the days of CPD and LPM, and now divides his time between Clarion programming, photography, and other equally challenging interests. Known for writing emails of near Tolstoyan proportions, Chris is possessed of a teacherly spirit regarding written material, user interfaces, pronunciation of kilometre, and sundry other irrelevancies. Has a penchant for organising things (ever since starting the anti-sport league in primary school), a trait which has not always sat well with the anarchic Clarion community in his native Australia. Chris has bossed the Victorian User Group around for three years (making it into a pretty successful group along the way), and has organized yearly Clarion conferences in Victoria since 1998 (CONVIC 2003 will be conference number six).

Reader Comments

Add a comment

Clarion Magazine

Clarion News

Published 2001-11-21

EmailReport Nettalked

EmailReport Nettalked by VividHelp is a specialized report emailing template that works in conjunction with CapeSoft's Nettalk template (not included). With EmailReport Nettalked template you can: Send Clarion, CPCS, RPM, DAS, and TIN reports in the email body; Duplicate report in RTF format as attachment; Email reports as RTF attachment only; Save report to disc without emailing as RTF in silent mode or as RTF, DOC, HTML in manual mode; Send unattended and/or bulk emails; Send and play sound files. Demo available. Available from ClarionShop for $70.

Posted Friday, February 28, 2003

Icons Sets From Ace Icons

Sue and Arnor at IceTips have set up a new website: Ace Icons. At Ace Icons you can get color-matched icon sets, GIFs, backgrounds and buttons. The Ace Icons items are all original creations. As a special sale just for Clarion developer, Ace Icons is offering a free matching imaging set (when it becomes available) for any icon and/or glyph sets you buy before the end of February, 2003. The base set includes 110 different images, and costs $25 (less in a suite). Coming soon: The Ace Imaging Set, a specialty set of icons for use in your imaging apps. These are being designed with the advice of a developer who is using Jens Weierman's ImageEx, but they will work with any imaging needs. Other specialty sets will follow. Background (wallpaper) images are also available, in colors to match the icon sets. The Ace Background sets cost $10 each (less in a suite). Ace Button images work with the Icetips Magic Buttons templates, or on a web site. The Ace Button sets cost $5 each (less in a suite). Icon images are also available as GIF files. The Ace Glyph sets cost $25. Or get everything at once, in a suite.

Posted Friday, February 28, 2003

CPCS Beta Install Files For Clarion6

CPCS has C6 Beta versions of all CPCS Products. Upgrade policy information and install files for all products (except the ClarioNET add-on which should be available a bit later) are on the web site.

Posted Friday, February 28, 2003

xReplacer (TXA Manager) v1.0 beta

SealSoft's xReplacer works with TXA files. Currently in beta, its features include: Move up and downwards on filled embeds; Comment and uncomment all embed lines; Search any text in filled embeds or whole file (from current line or from top of file); Replace any text in filled embeds or whole file (from current line or from top of file); Move up and downwards on changed lines; Edit embeds; Replace in embeds; Go to line by number; Embeds and Buttons List; Go to specific embed; Go to specific button; Open last edited file (options); Save file and SaveAs file. Demo available. During beta period the price is $59, final version will be $89. If you have a SealSoft discount card, use it at ClarionShop (enter the number in the Promotion Number field) and save 10%.

Posted Friday, February 28, 2003

HelpMaker.NET Now Freeware

HelpMaker.NET is a help authoring tool written in C#. There used to be two versions - standard (freeware) and professional (shareware). Now, both versions are now freeware.

Posted Friday, February 28, 2003

INN Bio & News for 25-Feb-2003

A Clarionite in the Pacific Northwest, he's been an expert witness for F. Lee Bailey and won both tennis and photographic competitions. He says about technology: "It marches to a different drummer than I do, I think. Technology marches in double time, and I'm marching behind the horses and have to watch my step." Gotta love him! Be sure not to miss the photos at the end.

Posted Friday, February 28, 2003

EasyResizeAndSplit 1.03

Infasoftplus has released EasyResizeAndSplit version 1.03. Changes include: Small modifications in the templates; Usage of IMM attribute required for the window (rollback); Fixed GPF with controls with a negative coordinates.

Posted Friday, February 28, 2003

[MS] Outlook EmailReport 1.1

Vivid Help has released Outlook EmailReport 1.1 for C5/55 ABC/Legacy. EmailReport lets you send your reports (Clarion, CPCS, RPM, DAS, TIN) as RTF attachments with MS Outlook. You can send emails one-by-one or in batch mode.

Posted Monday, February 24, 2003

Taboga Software Schedule For Feb 24-28, 2003

Edgard Riba will be out of the office until Feb 28 and may not be able to respond to email.

Posted Monday, February 24, 2003

EasyResizeAndSplit 1.02

EasyResizeAndSplit ver 1.02 is now available. Changes include: Allow Resize for Both direction, width only or height only; Different types of the split control - Panel, Invert Transparent; Dynamic resizing for SpilBars and Window (with all controls); Fixed controls of movement for split control; Compatibility with non-Clarion controls (SYSLIST, KSSTBAR etc); Window IMM attribute no longer needed; Fixed incorrect resizing for not MDI windows. A new demo is available. All registered users can download the new version.

Posted Monday, February 24, 2003

BoxSoft SuperSecurity 5.0a

BoxSoft's SuperSecurity Version 5.0 is now available. This release includes some new fields and a new table, so if you're upgrading be sure to read the upgrade notes. All hard-coded windows have been moved into generated Window procedures, to enable better handling by Clarionet, etc. There's a new property called Security.FullAccess. If you want your users to logon, but for some reason you want to allow unlimited access throughout the program, then set this to True. There's a new method called Security.PrepareFilenames. It's an empty virtual procedure that you can override to set your own variable filenames and owner attributes for the security files. The Logon can happen automatically using the network username (as long as it matches a username in SSEC::User). Passwords are no longer restricted to 8 characters. The new default (when importing SECURITY.TXD) is 20 characters, and you can edit the field definition to a maximum of 50 characters. There is also an optional setting for minimum password length. You can set passwords to expire, so that the user must enter a new one. This comes from a global default, but can be changed on a per user basis (including no timeout at all). There's a new "Locked" feature to lock-out users from logging on. When the Security system HALTs the program, you can execute some of your own code on the way out.

Posted Monday, February 24, 2003

HTML Designer 1.04 Beta Update

HTML Designer Version 1.04 Build 12 is available. This is an update to Version 1.03 and cannot be installed on its own. Use the same password as for version 1.03. HTML Designer is a WYSIWYG HTML Help Creation utility that interfaces directly with the Clarion Development environment for HTML Help file Creation in all Windows versions of Clarion up to C55H. (Not tested with version 6 yet). This Update needs version 1.03 (69) to run. Changes include: Updated user interface; more stable DHTML control; Added HTML help and contents control windows; Setting of contents tab tree options now included (change icons, tree style,etc); All images for a project now under Images sub-directory; Selecting an image

from "Anywhere" to be included in a project automatically copies the image to the images sub directory; Check for duplicate image names with options to use existing image or overwrite image; Hotspot map editor updated; and much more. HTML Designer is available through ClarionShop.

Posted Monday, February 24, 2003

PD Lookup And Drop Edit Controls Updated

ProDomus has released updates to PDLookup and DropEdit Controls. Changes include: Parameters can now be passed to the browse lookup procedure (Clarion5 Version 05-05 and C55 Version 55-06); Fix for a class library situation where no edit record is added to the queue which might cause the edit list to appear when a blank record was selected; Fix for a template issue that might occur when two procedures use the same local data definition (C55 Version 55-04).

Posted Monday, February 24, 2003

Winner Of The cpTracker Drawing

The winner of the very first weekly cpTracker drawing is Kell Jørgensen of Proff Consult ApS. Congratulations Kell!

Posted Saturday, February 22, 2003

cpTracker R3

Berthume Software's cpTracker R3 adds the following features: All browse windows are now resizable. Additional columns have been added to all task browses; The Application Procedures option on the Configure menu now has an import option to import all procedure names and descriptions from your Clarion Application TXA files (if applicable); A new report previewer (IceTips) has been added with the ability to generate PDF files (PDF Tools) of your reports which you can then email or archive. More PDF enhancements will be coming such as the ability to email the PDF directly from the previewer.

Posted Saturday, February 22, 2003

Excel Read/Write Utility 1.2

Alexander Ageev has announced version 1.2 of the ERW Excel Reader/Writer utility. This version can read/write Excel files to/from CSV (Tab, basic, etc) files in addition to all previous features. Auto-detect of the most used formats (dates, decimals, reals) is included, along with a number of other features. A new demo is available.

Posted Saturday, February 22, 2003

SkyHI Debug Template

SkyHi Software has released a new debug template which lets you view all the variables that are currently in use at the press of a button. Templates include global and procedure extensions and a breakpoint embed. Available from www.clarionshop.com.

Posted Saturday, February 22, 2003

ImageEx 2 Gold Release

ImageEx 2 is now in gold release. The installation password hasn't changed, so if you're an existing user please use the one sent to you before. New features include: Completely new TIFF engine, now (finally) supporting CCITT group 4 tiff images; Support for Clarion 6 EA 1; New ImageExPcxSaver class for saving uncompressed PCX images; Some minor changes to the bitmp, viewer and panner classes. A new demo is also available.

Posted Saturday, February 22, 2003

Newsletter Service

MyComputerJouranl.com is offering a newsletter service for computer consultants. Each month MyComputerJournal.com will send you a newsletter for you to send to your clients. The newsletter will have computer related articles and tips for your clients with your name and phone number on every page. You have a choice of receiving professionally printed copies of your newsletter or a PDF file. With your subscription, you also receive a free HTML file and link for customers to view your newsletter online.

Posted Thursday, February 13, 2003

Win A Free Copy Of cpTracker

Berthume software is holding a weekly drawing for a free copy of cpTracker. Visit the web site and click on the "Win a Free Copy..." graphic on the lower left.

Posted Thursday, February 13, 2003

cpTracker Silver R2 Available

There is a new version of cpTracker available for immediate download. This version features bulk email capabilities using your MAPI email client, with SMTP support to come shortly.

Posted Thursday, February 13, 2003

chSTD Library Version 2.63

Ingasoftplus's chSTD Library version 2.63 is now available. This release includes changes to date/time processing, reporting, and various miscellaneous functions. New demo available.

Posted Thursday, February 13, 2003

EasyVersion 2.01 Released

Ingasoftplus's EasyVersion ver 2.01 is now available. This release fixes some small bugs in the template.

Posted Thursday, February 13, 2003

Icetips Clarion 6 Compatibility

Icetips Software has made available three installs that are Clarion 6 EA1 compatible. EA2 will be tested shortly. The products which are currently Clarion 6 compatible are: Icetips Magic Buttons; Icetips Magic Entries; Icetips Previewer. IceTips will be working on C6 compatible releases of the other products and will make them available as they get tested for compatibility. Some of the remaining products require recompiles of DLLs etc. and these may not be available publicly until C6 goes gold, but they will be available to beta testers before that. If you are interested in beta testing (and are an existing customer) email clarion6beta@icetips.com.

Posted Thursday, February 13, 2003

Clarion Third Party Profile Exchange Updated

The Clarion Third Party Profile Exchange consists primarily of profiles of third party add-on products and vendors. This includes freeware templates and tools as well. Online and Downloadable Profiles available. Online product profiles include Product Internet URL, Order URL, Dated Price Quote, Grouped by Category, Extended Description and Download Page Reference. Currently, there are 424 product profiles and 355 vendor profiles. You must have Product Scope 32 PRO Version 4.5 or 4.5a to view profiles with data files (downloadable profiles).

Posted Thursday, February 13, 2003

ABCFree Templates And Tools Updated

ABCFree Templates And Tools Version 2.42 is now available: Changes include: Added "Simple Filter" support - user presses CTRL+SHIFT+F and can enter text to search the current list for (string fields only); vsSimpleStringFilter procedure; Template to add call to procedure to all browses; Fixed bug in "SendKeys" logic - the EXTENDEDKEY attribute was being applied to all characters, and on some systems this caused applications to randomly open (dependent on the keyboard driver that was installed); Added support for "Popup" selection when multiple reports could be called using the PrintButton template; Added "Copy/Cut/Paste/Undo" popup menu for normal entry fields (template and class); Added "Prevent MDI Errors" template. The ABCFree Templates and Tools are a set of templates, classes, and utilities for Clarion 5 and 5.5 (ABC template chain).

Posted Thursday, February 13, 2003

CPCS Builds v5.16 (C5b) And v5.57h (C55)

CPCS has posted new builds for v5.16 (for C5b) and v5.57h (for C55) which have been modified as follows: Previewer Page Number spinbox now supports up to 99,999 pages; The Print and Stay toggle can now be set to ON by default (via the previewer INI file). Use your existing codes for these builds to install the new files.

Posted Thursday, February 13, 2003

Special Offer From Berthume Software And Epsilon Concepts!

Get Berthume Software's cpTracker, a complete project management and bug tracking system, for free if you order a service or product from Epsilon Concepts that costs $250 or more! If you spend less than $250, you will receive a 50% off coupon for cpTracker. This is a savings of $80-$159.00.

Posted Thursday, February 13, 2003

Ingasoftplus EasyResizeAndSplit 1.01

Ingasoftplus has released EasyResizeAndSplit 1.01. Now supports non-MDI windows, and has fix for SplitBar control on Windows 98. New demo is also available.

Posted Thursday, February 13, 2003

1st Logo Design

1st Logo Design is having a developer graphics special for $199. Package includes contains an application logo, an application icon, splash screen and product box image (regular price $299). Also included: a 20% off coupon for anything in Gitano Software's product line or custom work in 1st Logo Design.

Posted Thursday, February 13, 2003

C6EA2 Patch Released

A patch for release 2 of the Clarion 6 Early Access release is now available to participants. The honor of the first known EA 2 compatible third party product in binary release goes to Lee White's RPM!

Posted Wednesday, February 12, 2003

EasyResizeAndSplit 1.00

EasyResizeAndSplit (ERS) ver 1.00 has been released. This class and template lets the end user resize window controls and use split bars. ERS generates code to reposition the controls, resize the controls, or both, when the end user resizes the window or moves the split bars (vertical or/and horizontal). Price is $49.

Posted Friday, February 07, 2003

xDigitalClock v1.5

New in xDigitalClock v1.5: The xDigitalClock control now can be used as count-down timer. New methods include: SetTimerOpt; GetTimerResult; SetTimerTime; StopTimer; StartTimer. Code templates for these methods are supplied.

Posted Friday, February 07, 2003

PDF-Tools New Features

This new build of PDF-Tools now allows the auto generation of watermarks (images and text supported) on PDF output (select fonts as outline, foreground/background etc.). Additionally, by completing a few template fields it is now possible to auto-generate nested bookmarks

(outlines) for your reports.
Posted Friday, February 07, 2003

Handy Tools FTP/HTTP

The upcoming update #2 for build O7B2 of the Clarion Handy Tools includes a server/client set of demo applications that illustrate how you can use The Clarion Handy Tools and Clarion to build secure file transfers applications using HTTP over the internet.
Posted Friday, February 07, 2003

ProDomus Updates C55 Translator Plus

A minor update to Translator Plus is now available. Changes include: Legacy Template - fixed posting of non fatal error saying certain include files could not be found (occurred when opening the global extension); Translation Assistant - added preference item to change replacement string font attributes to provide better support of multi-byte character sets; TPSTR.INC - changed definition of pdFontGT to handle styles.
Posted Friday, February 07, 2003

gReg Licensing Change

Gitano Software has changed its licensing policy for gReg effective immediately. All new purchases and upgrades will now include a five license pack and the language modules for Clarion, Visual Basic and Delphi at no additional cost.
Posted Friday, February 07, 2003

ProDomus PD Universal Drop Edit Controls

ProDomus has released the PD "Universal" Drop Edit Controls, a Drop List or Combo with an "Edit" choice that you can populate quickly without having to create a file, browse or form. The library can use INI or database files to save the pick list. Prompts, headers, pictures, and entry mode come from the dictionary or local data. It's quick to implement because: There's no need for a browse; There's no need for a form; There are very few template entries. It relies on the dictionary for headers, prompts, pictures, and entry mode.
Posted Monday, February 03, 2003

SealSoft xFunction Library v1.7

SealSoft's xFunction Library v1.7 is now available. This is a bugfix release.
Posted Monday, February 03, 2003

XPMenu Holiday Schedule

Ronald van Raaphorst will be on holiday from February 3-19, 2003. His colleague will takeover for urgent emails (if any) and sales.
Posted Monday, February 03, 2003

HelpMaker Update.

HelpMaker.net, a help authoring tool that can do WinHelp, HTML-Help can now do HTML generation of help files.

Posted Monday, February 03, 2003

powerRUN Freeware

The free powerRUN utility picks up where RUN() left off. Features include: Launch programs hidden, minimized, in a window or maximized, with or without focus; Run DOS or Window processes in the background, waiting (or not) until termination; Set the Priority in 32bit apps; Load websites and documents using default browser or application via the ShellExecute code template; Pop the client's email window with the email address and subject automatically filled in, CC's listed, BCCs listed, etc. Uses ShellExecute(); Play Wave sound files. powerRUN works in 16 and 32bit, Clarion 2003 and earlier, C4, C5, C5.5 ABC/Legacy and is multi-DLL compliant and DET compatible.

Posted Monday, February 03, 2003

Clarion 5.5 MS SQL Server Seminar In Johannesburg

A Clarion 5.5 seminar on MS SQL Server will be held March 16-18, 2003 in Johannesburg, South Africa. Cost is R3700 per person incl VAT. Requirements: Clarion 5.5, notebook or PC Computer (recommended), SQL Server (eval copy provided if needed), lunch and refreshments provided but you're on your own for dinner. Minimum of 15 needed to make class size up - bookings must be made ASAP. Class Size is limited to 20 so personal attention can be given.

Posted Monday, February 03, 2003

INN Bio And News For 28-Jan-2003

A long term Clarion user, and an Aerospace Engineer, Benjamin is certainly no newcomer to technology. Born in Mexico City in 1963, moved to San Antonio, Texas in 1974 and lived in Israel for 14 years, he has been places and seen things. Now he lives in Denver, Colorado, with his wife, Irit, his two daughters, Vered and Sivan, and works for his own network support company.

Posted Monday, February 03, 2003

Firebird 1.5 Beta 1

Kelvin Chua reports that Firebird 1.5 Beta 1 is now available for download.

Posted Monday, February 03, 2003

cpTracker Silver Edition Released!

cpTracker 2003 Silver Edition is now available for download and purchase. cpTracker is a tool specifically designed for independent software developers, software development departments,

software companies, contract programmers, etc. Keep track of customers, customer product sales, users, products, versions, projects, project tasks, customer projects, beta sites, defects, enhancements, and more. Create custom queries, reports, and spreadsheets, and custom HTML pages for posting to your web site. Includes FTP Wizard for easily uploading and downloading files.

Posted Monday, February 03, 2003

Threading Download At RadFusion

Owen Brunker's template modifications to handle the 5.5 threading issues are now available at the RadFusion web site.

Posted Monday, February 03, 2003

Product Scope 32 PRO Sale Ends Feb 3

In honor of the Super Bowl Champions - Tampa Bay Buccaneers - Encourager Software is extending the sale of Product Scope 32 PRO software at sizeable savings through February 3rd, 2003. Product Scope 32 PRO, Version 4.5 (a) - Spreadsheet Version is USD $15 (normally USD $49); Product Scope 32 PRO, Version 4.5 (a) - Unlimited Editing Version is USD $10 (Normally USD $29).

Posted Monday, February 03, 2003

ZipApp 1.1b Freeware

ZipOption 1.1b, a freeware utility, now has an option to back up TXA (and TXD/TXR) files. Fixes include changes to method calculating Last Incremental to Date Last Run, and a slight file change to show _to_ instead of _at_ on incremental file names.

Posted Monday, February 03, 2003

SealSoft xDigitalClock v1.4

SealSoft's xDigitalClock version 1.4 is now available. Changes include: new SetStopWatchTime(LONG StopWatchTime) method; Some template changes; New demo and install.

Posted Monday, February 03, 2003

Reader Comments

Add a comment

Demystifying C6 Threading (Part 1)

by David Harms and Carl Barnes

Published 2003-02-15

Mutexes, semaphores, critical sections, reader/writer locks; all of these things are part of Clarion 6, and they all have to do with the new support for unlocking Clarion threads so they run like real operating system threads. Should you care? If you write any embedded code, yes, you should. Should you worry? That all depends on what kind of embedded code you write!

The hullabaloo over threading has one root cause: multiple threads can now execute, for all intents and purposes, simultaneously. And there is one potential victim: global data without the THREAD attribute. This article series is about how to recognize situations where global data is in danger, and how to remove that danger.

How threads worked before C6

In versions prior to Clarion 6 it really wasn't true that multiple threads ran at the same time. Although you could launch a number of threads with START, only one was ever active at a time. The ACCEPT statement controlled thread switching in a token-ring fashion so that only one thread was active inside the ACCEPT loop.

Here's what happens in a pre-Clarion 6 application when you create several threads. When you launch the application (in Windows terminology, each instance of an application is also called a *process*) it has one thread of execution, which normally is your application frame, containing the main menu. From the main menu you select, say, a browse procedure, and if you're using the normal Clarion MDI approach there will be some code in that main frame that looks like this:

```
START(BrowseProc1,25000)
```

The runtime library creates a new thread for the browse procedure. Now there are two threads running in the application. Click on the main menu again and you cause a thread context switch back to the application's main thread, and if you choose another browse procedure, or a second instance of the first browse procedure, there is another call to `START()` and another thread begins for that procedure. Now you have three threads. The thread that is active is determined entirely by which window you click on, whether the app frame or one of the browses. (Again, that's assuming you have an MDI application, and also that you're not doing any fancy POSTing of events to cause thread switching.)

The key point is that thread switching is all handled by the `ACCEPT` statement. If you have a tight loop running in one of those threads, clicking on another window won't cause a switch to that other window's thread. First the tight loop has to terminate so the code can reach the `ACCEPT` statement. This is why reports and processes don't just loop through all records in a single sequence; they read *x* number of records at once, then go back to the `ACCEPT` loop and wait for another `TIMER` event. Without this little trick (pre C6) you'd never be able to cancel a report or process! And you could not do anything else in another thread of your application while a long report or process ran. This is a form of thread control known as cooperative multitasking, where threads retain control until they give it back. This was also the only way to do multitasking under 16-bit Windows.

How threads work as of C6

In C6, however, multiple threads can run at exactly the same time. Well, not exactly. Windows, at least in 32-bit versions, use a preemptive multithreading model. On a single-processor computer you can't really have multiple threads running at exactly the same time because there *is* only one processor. The operating system executes a few instructions for a given thread, saves the processor state, switches in some instructions for another thread, and so on. Notice that there is no mention of a process (an application being a process) in that description. Windows schedules, manages and switches threads. A process is just a container to manage memory and thread scope. Windows protects processes from each other, but all of the threads within a process share the virtual memory address space within that process. And every process must have at least one thread.

In a preemptive multithreading system time slices are really very small, which means that it's quite possible for one thread to be halfway through changing, say, a global string variable to a new value, when it gets preempted, i.e. is put to sleep. The variable now contains half new and half old data. If the new thread that gains control reads that entire variable it will get corrupted data (half new and half old).

Chances of corruption become more likely with complicated data structures such as queues, where inserts, deletes, and sorts may involve many instructions. As a result, there are now

Clarion versions of Windows mechanisms which make it possible to control how and when threads run under specific conditions. Those mechanisms (a.k.a. synchronization objects) are the aforementioned mutexes, semaphores, critical sections, and reader/writer locks (and we'll have much more to say about them in Part 2).

Is this kind of data corruption a very likely occurrence? In most business applications, no, not really, because the user generally controls which thread is active at any one time by launching procedures and clicking on active windows. But it could be the cause of intermittent bugs or application crashes. And in an application that does a lot of processing, failure to safeguard access to data could have frequent, serious consequences.

Shared resources – an example

Here’s a real world example of the global data problem, which is typically defined as "synchronizing the use of *shared resources* between threads." Global data is an example of a shared resource.

Consider a single phone line that comes into your house. You can have any number of phone handsets, but they all must share the one line. When you are using the phone in one room nothing prevents someone in another room from picking up the phone and trying to use it. That’s an example of resource contention, and guess what? Your call got corrupted. Anyone with who has used dialup Internet access is familiar with this problem.

Each person in that house is like a thread in an application, potentially contending for a shared resource. If there are only two adults, there may never be any contention for the phone (the shared resource) because they just don’t use the phone that much and tend to notice if someone else is using it. This is analogous to why a typical business app will *probably* not run into shared resource corruption. It’s just not going to get used hard enough.

The cooperative phone sharing situation would never work in a business with multiple phone lines and many handsets. So business phone systems implement synchronization that prevents two callers from interrupting each other by trying to use the same phone line. They also implement features such as conference calling that let multiple callers share the same line. With properly programmed synchronization a Clarion 6 app can run like a business phone system. Without it you’ve got a home phone that could potentially have problems.

One word about terminology. The mechanisms Clarion 6 provides for dealing with shared resource management are called synchronization objects. The word "synchronize" really means to cause events to happen at the same rate or the same time. Synchronization objects like semaphores can be used this way, but more commonly synchronization objects are actually used to serialize access to shared resources so that no two threads use the resource at once. Serialization has another meaning in programming, however (which is to convert an object like

a class into a stream of bytes). Just keep in mind that the programming definition of synchronize is a lot broader than the common, every day definition.

Now you know the nature of the problem. Coming next week: how you can manage access to shared resources using the Clarion synchronization objects.

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).

Carl Barnes is an independent consultant working in the Chicago area. He has been using Clarion since 1990, is a member of Team TopSpeed and a TopSpeed Certified Support Professional. He is the author of the Clarion utilities CW Assistant and Clarion Source Search.

Reader Comments

Add a comment

Looking forward to the rest of this article! Keep up the...
Thanks! Part 2 should be up in a couple of days.
Thanks, good article. I was wondering if you can cover...

Demystifying C6 Threading (Part 2)

by David Harms and Carl Barnes

Published 2003-02-20

Part one of this series introduced the concept of shared resources, and showed how Clarion 6's new threading model introduces the prospect of corrupted global data. That article also introduced the concept of shared resources, of which global data is an example. In this article you'll learn how to use some of Clarion 6's synchronization objects to manage shared resources *when necessary*.

While global data is the most common example of a shared resource, you must be concerned about all *static data* that does not have the thread attribute. This includes module data and local data with the STATIC attribute. Because global data is the most common, that is the term we will use in this article; just remember you must be concerned about *static data*.

You probably have some global data that cannot be threaded, but which you do *not* need to worry about, particularly if you only write that data once while the application is initializing and only read it subsequently. Also any global data that is used to pass information between procedures should still work the desired way if it is changed to have the THREAD attribute. It is only data that must be passed between or used by all threads that requires careful attention.

The first thing you should try to do is not synchronize, but simply get rid of the global unthreaded data. It's like in the movie War Games - the best move is not to play. The easiest way to do that is to try to make it work as threaded data. In many cases simply adding THREAD is all that is required and sha-boom! (that's a technical term often used in OLE class browsing) you're done. A more efficient change would be to make global data into local data and pass it between procedures. The START function allows passing up to three string type by-value parameters to new threads.

Global queues must be handled differently from other data types because buffers are involved. To work on a global queue's record you must use the queue buffer and that is shared by all threads. Two threads might try to get, put or add different queue records at the same time and can corrupt the data.

One possible fix is to put the `THREAD` attribute on the declaration. This will cause the runtime library to create a new copy of the queue for each thread, so that the resource is not shared and queue access code can function properly. The new copy of the queue will be created empty so you may need code to fill the queue with the correct records for each new thread. This can be done easily with a threaded class that has a constructor to populate the queue.

If you require a single global queue shared by all threads then you'll need to wrap all queue access with synchronization code. There is an example of this in the C6 docs – look for the `QueueAccess` class.

Classes are yet different from queues and variables. If you put the `THREAD` attribute on a class declaration, the runtime library will create a new instance of that class each time a thread is launched, and it will also call the class's automatic constructor (you must give this method the name `CONSTRUCT`) on thread creation, and the destructor (the `DESTRUCT` method) on thread termination. You can use the constructor to initialize any data the class needs. Note that in a threaded Class all properties (class data) is implicitly threaded.

If you are reading and writing your global data, then chances are you'll need to do a little more work to ensure your code runs smoothly. The short answer to the problem of data corruption in a multithreaded environment is to synchronize access to the data in question. This simply means identifying each item of the global data that might get corrupted by multiple readers and writers, and then only allowing one thread, at one time, to execute a section of code that reads or writes that data.

Critical sections

The most efficient way to synchronize data access is to use a *critical section*. If you're ambitious, you can do this with the Windows API using the `EnterCriticalSection` and `LeaveCriticalSection` functions, but Clarion 6 provides its own wrapper for this functionality.

To synchronize a shared resource such as global data, you simply use the `NewCriticalSection()` library function to get an instance of `ICriticalSection`. You then call the `Wait()` method to "wait" your turn to continue into the critical section, when `Wait()` returns you execute your data access code, and finally you call the `Kill()` method on `ICriticalSection` to clean up and allow the next thread access:

```
cs   & IcriticalSection
CODE
cs &= NewCriticalSection()
! Get access to the critical section
cs.Wait()
! I now own the CS
! read or write some global data
cs.Release()
cs.Kill()
```

Although this code demonstrates the basic principles of a critical section (create the critical section, call Wait(), execute code, clean up) it isn't that practical a framework for managing global data. Think of how you use your global data now – you declare it, and then you read/write it from various locations in your code. If you want to continue using globals that way, you'll need to make sure you wrap each section of code that uses the globals in a critical section. And what happens if you write some new code that uses the global data and you forget to include a critical section? How do you maintain all of this code?

A better solution is to wrap the global data in a non-threaded global class instance, and make the data private so it can only be accessed using "getter/setter" methods. You then create a critical section (or some other form of synchronization) for the class, and use that critical section whenever you read/write the data. The C6 threading docs include an example of such a class, so we won't go into details here.

There are several other points to note about this code. One is that it synchronizes access to this data only within an application (a process). If you have multiple instances of an application running on one machine, it's possible that two of them will execute the same critical section at the same time. But if you are protecting global data then there is no possible conflict since different instances of a Clarion program do not share global data. Also, and perhaps more importantly, two *different* critical sections can run concurrently. If you have one procedure with its own critical section for reading the data, and another procedure with its own critical section for writing the data, then you can still have data corruption. If, however, you wrap your data access up in a class, you can easily create one critical section in the class constructor and then use it as needed in the class's getter and setter methods.

Remember that you must call the `Wait()` method to get access to (that is, own) the critical section object; if another thread is using it, this method will be blocked until the critical section is released by that thread. Note that a thread can never block itself waiting for a critical section it already owns.

In Clarion versions before C6 the 32-bit RTL `ACCEPT` loop entered a critical section any time an event fired the `ACCEPT`. When control returned to `ACCEPT` the RTL would leave the critical section and then an `ACCEPT` in another thread could own the critical section. If you trace through the `ACCEPT` loop in the debugger you'll see this happen. So you were always in

a critical section and so there was never a risk of corrupting your global data. When you read about critical sections this kind of design is usually frowned up as it creates a lot of contention. C6 enables (and forces) the developer to manage critical sections, thus allowing a program to run better. As a rule, you really do want your critical sections to be as few as possible, and as short as possible.

Critical sections are the fastest and cheapest way to regulate access to shared data, but they only work within a process (i.e. a single instance of an application executing). They are designed for a situation of low contention to protect small blocks of code. They achieve their efficiency by remaining in *user mode* and not making the very expensive switch to kernel mode unless blocked. They also gain efficiency by assuming you will use them correctly and provide almost no protection against improper use. It you release them too many times or fail to release them (abandonment) other threads trying to own the cs will probably hang.

Critical sections also don't give you a way of testing to see if they are available (i.e. `TryWait`) – once you call `Wait()`, you're committed to waiting. (`TryEnterCriticalSection` is available under NT but not under Win9x, so SoftVelocity has chosen not to implement the feature.) You don't have much control over how those threads run – it's all up to Windows. If you need to manage a high level of contention between threads, or you want to regulate access among all processes running on one machine, you need to use a mutex.

Interlocked functions

There is one more Windows mechanism that provides tools for efficient user mode synchronization – the interlocked family of functions. There is no Clarion equivalent for these functions – you'll need to search MSDN for "Interlocked" to get the details. These functions provide *atomic access* to LONG integers. *Atomic* means that no other thread will be able to access the LONG until the function completes. This has the same effect as wrapping the code in a critical section, but is much faster. The interlocked family provide atomic functions to add, subtract, set and, compare-then-set a value.

The following code uses a critical section to see if a thread is already running so the user can be limited to a single thread running the Vendor browse:

```
csThreads.Wait()
IF Glo:ThrdsVendorBrowse=0
   Glo:ThrdsVendorBrowse=THREAD()
   csThreads.Release()
ELSE
   POST(EVENT:GainFocus,,Glo:ThrdsVendorBrowse)
   csThreads.Release()
   RETURN
END
```

You can do the same thing using an interlocked function with the below code. Note that all interlocked functions return the *prior* value. This function call will test that `Glo:ThrdsVendorBrowse` equals zero, and only then will it set it equal to thread(). It returns the prior value which the Clarion `IF` checks if another thread is running.

```
IF InterlockedCompareExchange( |
     Glo:ThrdsVendorBrowse,|
     THREAD(), 0 ) > 0
   POST(EVENT:GainFocus,,Glo:ThrdsVendorBrowse)
   RETURN
END
```

(This code is not as perfect as the critical section code. Only the interlocked function is atomic. In a worst case situation a thread switch could happen after it returns and before the POST. During that time the vendor browse could shut down.) If you have threads doing intense work with global integers you'll want to read about the interlocked functions. Critical sections actually use interlocked functions to provide their user mode locking efficiency.

So much for the API digression. We now return you to your regularly scheduled Clarion synchronization objects article…

Mutexes

A mutex is like a critical section that is managed by the OS kernel. (The name comes from its purpose, which is "mutual exclusion".) You ask for a mutex by specifying an identifying name; if no mutex by that name exists, one will be created. Here's a code example:

```
mt &iMutex
CODE
mt &= NewMutex('MyMutex')
! Get access to the Mutex
mt.Wait()
! read or write some data
mt.Release()
mt.Kill()
```

As with the critical section, the call to `Wait()` will be blocked until any other thread owning the mutex calls the Release() method. Both critical sections and mutexes allow a thread to make multiple calls to `Wait()` and never wait on itself, to allow for their use in recursive calls – just make sure you call `Release()` once for every call to Wait(). This concept of thread ownership is only provided by critical sections and mutexes.

If you don't want to automatically wait (i.e. wait forever) for the mutex to be released, you can call the `TryWait(`*`length of wait in milliseconds`*`)` method. Possible return values are as follows:

`WAIT:OK`	You have the mutex and can proceed
`WAIT:TIMEOUT`	The mutex did not become available before the time passed to TryWait (in milliseconds) expired.
`WAIT:NOHANDLE`	Could not get a handle to the mutex
`WAIT:FAILED`	Attempt to get the mutex failed for some other reason
`WAIT:ABANDONED`	The thread that had control of the mutex ended (or was terminated) without calling Release(). You do own the mutex; this is the same as a Wait:OK return, but there is a chance the data the mutex is protecting was not completely updated by the prior owner.

Only one thread can successfully call the `Wait()` function at once and gain ownership of the mutex. As with critical sections, if thread one calls `Wait()` on a given mutex, and then threads two and three successively call `Wait()` before thread one completes its work and calls `Release()` on that mutex, both threads will be blocked. After thread one calls release, thread two gets the mutex, then when it calls `Release()` thread three gets the mutex. During the time they are waiting the threads are put into a very efficient sleep state and not given any CPU time.

If you want multiple threads to have access to some block of code, or resource, at once, you can use a semaphore instead of a mutex.

Semaphores

Semaphores are *somewhat* like multi-user mutexes. They have a resource count associated with them, and so multiple threads can own the semaphore simultaneously up to the predefined maximum resource count. This would allow you to write a server that might allow five threads to start to service incoming connections; any additional threads would wait. Picture a security check in at an airport with five metal detector lanes and a queue of 50 waiting passengers. The semaphore is the person releasing the passengers from the queue so they can go through the metal detectors.

When you create a new semaphore, you specify its name, and how many threads can use it at once. The following code creates a semaphore that can be used by up to three threads

simultaneously:

```
sem &= new ISemaphore('MySemaphore',0,3)
```

The '0' is the initial release count, the '3' is the maximum count. Often semaphores are created in this *maximum owned* state (initial release count of 0) while the code prepares the data the threads will manage. This causes other threads trying to own the semaphore to wait. When the data is ready a call of `Release(3)` opens the doors to waiting threads. This prevents a race condition when there is a tiny time slot during which corruption could occur.

Semaphores, like mutexes, have `Wait()`, `TryWait()`, `Release()` and `Kill()` methods. You can use a semaphore to limit the number of threads active at any one time, for instance.

Keep in mind that there isn't necessarily a one-to-one correspondence between a thread and a semaphore state. If a thread calls Wait multiple times, it will use up a resource count each time, and eventually could block itself when the maximum is reached. In this respect semaphores are quite different than critical sections and mutexes, where a thread can call Wait() any number of times without blocking itself (although even critical sections and mutexes require one `Release()` for every `Wait()`). For more information see the C6 threading docs or MSDN.

Mutexes and Semphores may be created without a name. This is useful if you need a mutex but do not need access to it from another process (or you have control over that processes and can pass it the mutex's or semaphore's object handle.) You probably are thinking, "Well, if I don't need to access it from another process then I'll use a critical section." Critical sections have a limitation in that they are not "waitable" objects. If you need the functionality of a `TryWait()` method, you're out of luck; there isn't one in the `CriticalSection` interface. Critical sections are designed for efficiency and not features.

There you have some of the basic synchronization mechanizes that Windows implements and SV has wrapped. Next time we'll look at the `ReaderWriter` class and critical procedures.

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).

Carl Barnes is an independent consultant working in the Chicago area. He has been using Clarion since 1990, is a member of Team TopSpeed and a TopSpeed Certified Support Professional. He is the author of the Clarion utilities CW Assistant and Clarion Source Search.

Reader Comments

Add a comment

I am a bit confused with the statement: "Also any...
JC, There really isn't any difference that I can see...

Data Structures and Algorithms Part XVI - The Huffman Compression Algorithm (Part 1)

by Alison Neal

Published 2003-02-21

In this article I will discuss the Huffman compression algorithm, which is the same compression algorithm that is used by PkZip. The algorithm yields approximately 40% compression for text files. The test application included with this article reduces the provided test file from 20kb to 12kb in size, and then decompresses it back to its original state.

The Huffman algorithm fundamentally works by replacing each character with a binary code. It starts by identifying the frequency of each recurring character in the file and then allocating a binary code to each character, with the most frequent characters getting the shortest codes. The file is then read again and the corresponding codes are output in compressed format.

Lets take an example piece of text: *"Steve sells sea shells by the sea shore"*

There are a possible 255 ASCII characters, so I define an array of 255, and then increment the count of the position corresponding to the VAL of the character. For example a lower case ‘a’ has a value of 97, so position 97 in the array would be incremented with every occurrence of the letter ‘a’.

The frequency array for the above text would be as shown in Figure 1:

Chr	Val / Idx	Count
[space]	32	7
S	83	1
a	97	2

b	98	1
e	101	8
h	104	3
l	108	4
o	111	1
r	114	1
s	115	7
t	116	2
v	118	1
y	121	1

Figure 1.

Note that in Figure 1 I have ignored any index value that would be zero.

From the frequency array, you can build a Priority Queue (lowest order first). The Priority Queue is then used to build a Huffman Tree, which is a form of Binary Tree. The most frequent values end up nearest the root, and the least frequent characters end up in the lower levels, the Priority Queue provides this ordering principle.

The Huffman Tree is then used to provide the binary codes associated with each character value. On decompression, the codes will be interpreted as a series of instructions on how to walk the Huffman tree to obtain the characters, one at a time. For the string *"Steve sells sea shells by the sea shore"* the Huffman Tree would look like Figure 2.

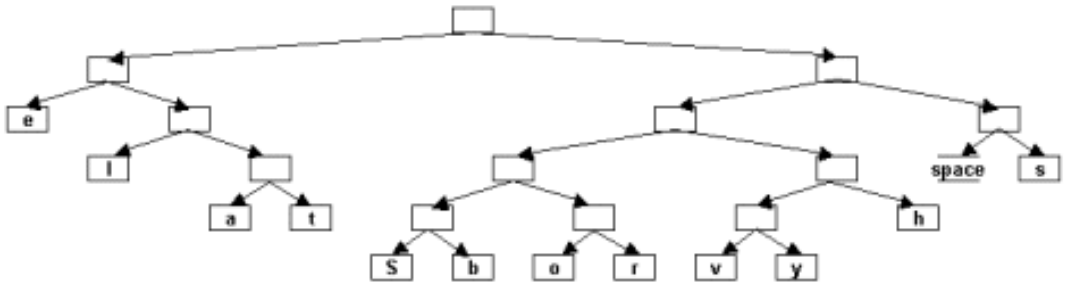


Figure 2.

The codes are then calculated by adding a 0 bit if a left branch is taken, and a 1 bit if a right branch is taken. The following codes are generated (Figure 3):

Letter	Code
‘e’	00
‘l’	010
‘a’	0110
‘t’	0111
‘S’	10000
‘b’	10001
‘o’	10010
‘r’	10011
‘v’	10100
‘y’	10101
‘h’	1011
[space]	110
‘s’	111

Figure 3.

Once the codes are generated the compressed file can be created. Firstly the frequency array is output to the file, so that the decompression algorithm can re-create the Huffman Tree and translate the codes.

Then the file to be compressed is re-read and the appropriate codes written to the compressed file, instead of the character. The compressed file is written one byte at a time, as the codes are squashed together into the byte (8 bit) space which one character would normally fill. For *"Steve sells sea shells by the sea shore"*, given the codes above, the output, one byte at a time, would read something like Figure 4:

Byte 1	10000011	(S and beginning of t)

Byte 2	10010100	(Rest of t, e and v)
Byte 3	00110111	(e, space and s)
Byte 4	00010010	(e, l and l)
Byte 5	11111011	(s, space and beginning of s)
Byte 6	10001101	(end of s, e, a and beginning of space)
Byte 7	10111101	(end of space, s and beginning of h)
Byte 8	10001001	(end of h, e, l, and beginning of second l)
Byte 9	01111101	(end of l, s, space, and beginning of b)
Byte 10	00011010	(end of b and beginning of y)
Byte 11	11100111	(end of y, space, and t)
Byte 12	10110011	(h, e, and beginning of space)
Byte 13	01110001	(end of space, s, e and beginning of a)
Byte 14	10110111	(end of a, space, and s)

Byte 15	10111001	(h and beginning of o)
Byte 16	01001100	(end of o, r and e)

Figure 4

The text has been compressed from 39 bytes (characters) to 16 bytes plus the frequency array, which is also written in compressed fashion. If the output had completed part way through a byte, the remaining bits would have been filled with zeroes and the last number written to the file would say how many zeroes that was.

To successfully complete the compression algorithm the application must complete the following steps:

1. Read the file to be compressed and build a frequency array for each occurrence of a character.
2. Load the frequency array into a Priority Queue and Build a Huffman Tree.
3. Assign a binary code to each character within the tree (shortest code to most frequent character), determined by whether the character is a left child or right child.
4. Write the frequency array in compressed format to the compressed file (so that the decompression algorithm will be able to rebuild the Huffman tree and therefore decode.)
5. Read the file to be compressed again and output the corresponding binary codes in compressed format.

Step 1. Build the frequency array.

Building the frequency array is a reasonably simple affair. It is simply a matter of defining an array of 255 (the maximum ASCII characters), and scanning the input file character by character and incrementing the count in the frequency array for that character.

Note, in the code below I have used a class for file handling, included in the sample code provided with this article. The class uses API calls rather than the DOS driver, but this is purely just a matter of preference and outside the scope of this article.

```
Build:Frequency       ROUTINE
 DATA
i           ULONG
 CODE
 !0 = File Begin & 0 for position
 Cancel = ios_imp.SetPointer(0,0)
```

```
IF ~Cancel
  LOOP i = 1 TO MAXIMUM(dat:freq,1)
    dat:freq[i] = 0
  END
  LOOP
    IF ios_imp.ReadFile(ADDRESS(loc:impLine),1)
          THEN BREAK.
    dat:freq[VAL(loc:impLine[1])] += 1
 END
 IF ~Cancel THEN Cancel = Tr.Init(dat:freq).
END
```

The first thing I do is set the file pointer to the beginning of the file. If this works I initialise the frequency array. Then I loop through the file reading one byte at a time, until the loop breaks when the read fails. I then increment the `VAL(read byte)` position in the frequency array. For a space this is going to be position 32, for a lower case ‘a’ this is going to be position 97.

Step 2. The Priority Queue and Huffman Tree.

I introduced the Priority Queue structure in my last article. I have changed the code only slightly to allow lowest order first rather than highest order first, which simply meant flipping the greater than and less than operators around on the `insert` method. I have also implement the Huffman Tree as an array, or rather as two arrays.

```
huffL       LONG(0),DIM(511)
huffR       LONG(0),DIM(511)
n           LONG(0)
huffRoot    LONG(0)
```

As I am somewhat pedantic about dynamic structures, it may seem odd that I have a fixed length array in my code; however, a binary tree with a maximum of 255 leaves (ASCII characters) should only ever have a maximum of 511 (255 * two possible values + 1) nodes. I feel reasonably secure in the knowledge that the code should not over run the bounds of the array.

Telling the difference between a leaf containing a valid character and a parent node which points (contains the index value) of its two sub-children is reasonably easy. The `HuffL` (left) index position will always be zero on a leaf making the `HuffR` (right) index position contain a valid character, whereas both `HuffL` and `HuffR` will contain values if it is a parent node.

For the example string the tree actually looks like Figures 5 and 6.

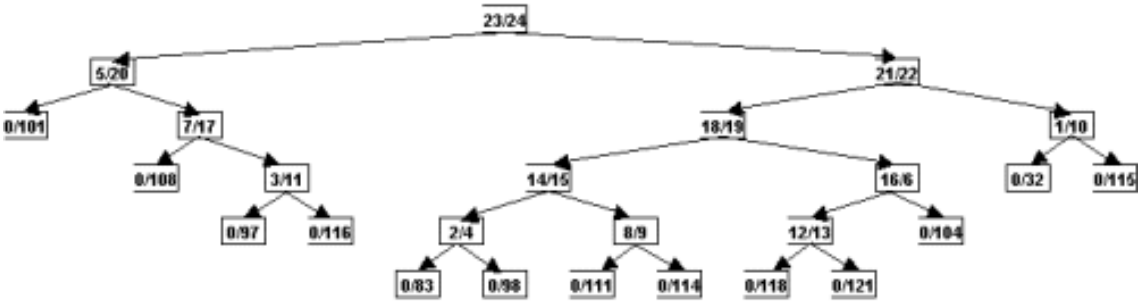


Figure 5.

Huffman Tree		
IDX	**Left**	**Right**
1	0	32
2	0	83
3	0	97
4	0	98
5	0	101
6	0	104
7	0	108
8	0	111
9	0	114
10	0	115
11	0	116
12	0	118
13	0	121
14	2	4
15	8	9
16	12	13
17	3	11
18	14	15
19	16	6
20	7	17
21	18	19
22	1	10

23	5	20
24	21	22
25	23	24

Figure 6.

Position 25 in the array is the root node of the tree, and it has a left and right sub tree. The left child is position 23 in the array and the right child is position 24 in the array. If I follow the left children from position 23, its left child is position 5 of the array, which is a leaf node because it has no left child, making 101 (‘e’) a valid character.

The code to build the Huffman Tree and Priority Queue is as follows:

```
huffTree.Init     PROCEDURE(*ULONG[] pFreq)
i  ULONG
T1 GROUP(T).
T2 GROUP(T).
   CODE
   SELF.n = 1
   p.init()
   LOOP i = 1 TO MAXIMUM(pFreq,1)
     IF pFreq[i]
       P.InsertQ(pFreq[i],SELF.newNode(0,i))
     END
   END
   IF P.isEmpty()
     MESSAGE('Error: Input File was Empty',|
           'System Error',ICON:Exclamation)
     RETURN TRUE
   ELSE
     LOOP
       P.remQ(t1.freq,t1.ptr)
       IF P.isEmpty() THEN BREAK.
       P.remQ(t2.freq,t2.ptr)
       P.insertQ(t1.freq+t2.freq,SELF.newNode(t1.ptr,t2.ptr))
     END
   END
   SELF.huffroot = t1.ptr
   P.kill()

   RETURN FALSE
```

The definition for the Group T is:

```
T       GROUP
freq    ULONG
ptr     LONG(0)
        END
```

The NewNode method is:

```
huffTree.NewNode        PROCEDURE(LONG pLeft, LONG x)
```

```
CODE
SELF.huffL[SELF.n] = pLeft
SELF.huffR[SELF.n] = x
SELF.n+=1
RETURN SELF.n - 1
```

So what's happening here? The `Node` count `n` is initialised to 1, and the Priority Queue is initialised. The code then loops through the frequency array. If the array frequency value is greater than 0, that is if the character occurred in the input file, it is added to the Priority Queue with a pointer (array index number) to the tree node that contains the value of that character.

The first character in the array is [space], index value 32, so when the `NewNode` method is called `pLeft` is 0 (zero) and `I` is 32, because this is the first value in the frequency array. The `NewNode` method then assigns these values to the arrays, and returns which `Node` this relates to, being 1. When the `insertQ` method for the Priority Queue is called it is passed 7, the frequency count (which acts as the priority), and the index number 1, which is where 32 is stored in the Huffman Tree.

As `HuffL[1]` contains 0 (zero) this node is a leaf node. If the left branch were not 0 (zero) then it would be a parent node that provides the index number of its left sub child.

The next element of the frequency array that isn't zero is 'S' value 83. So, a new node is added to the Huffman Tree, this time with the left value of 0 and a right value of 83. The frequency of 83 (1), and the index of the new node (2) are then added to the Priority Queue. Remember that the Priority Queue is keeping the nodes ordered lowest priority (frequency) first so now the Priority Queue will place this new one ahead of the entry for [space].

At the end of this first loop the Huffman Tree will look like Figure 7.

Huffman Tree		
IDX	**Left**	**Right**
1	0	32
2	0	83
3	0	97
4	0	98
5	0	101
6	0	104
7	0	108
8	0	111
9	0	114

10	0	115
11	0	116
12	0	118
13	0	121

Figure 7.

The Priority Queue will look like Figure 8.

PQ	
Freq	**Ptr**
1	2
1	4
1	8
1	9
1	12
1	13
2	3
2	11
3	6
4	7
7	1
7	10
8	5

Figure 8.

As the Priority Queue is not empty the second loop then starts building the parent nodes of the Huffman Tree. In this way the tree is built from the bottom up.

```
LOOP
   P.remQ(t1.freq,t1.ptr)
   IF P.isEmpty() THEN BREAK.
   P.remQ(t2.freq,t2.ptr)
   P.insertQ(t1.freq+t2.freq,SELF.newNode(t1.ptr,t2.ptr))
END
```

The first two nodes are removed from the Priority Queue. Note that the parameters of the `remQ` method are passed by reference, and are updated as a part of that method with the values

of the node being removed.

So `t1.freq` holds value 1, and `t1.ptr` holds index number 2. Therefore `t2.freq` holds value 1 and `t2.ptr` holds index number 4. At this point a new node is added to the Huffman tree which contains the values of the two `ptr` fields; 2 is assigned to `HuffL`, and 4 is assigned to `HuffR`. Thus the parent node is created and its corresponding node is added to the Priority Queue containing the combined frequency and the index value of the new parent node. (This use of the Priority Queue ensures that the most frequently occurring characters end up with the shortest codes, as they will end up higher in the tree than those nodes with lower frequencies, which can afford to have relatively lengthy codes.)

Note that the new Priority Queue node still only has a combined frequency of 2, which means that this new node will be placed behind the last node containing a frequency (priority) of two, which is well ahead of the node relating to ‘s’ with a frequency of 8, and as such will be built into the lower levels of the tree much earlier on then the highest priority node.

Next week I’ll continue with the explanation of the code, beginning with the assignment of binary codes.

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 (ClarioNET) 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

Data Structures and Algorithms Part XVI - The Huffman Compression Algorithm (Part 2)

by Alison Neal

Published 2003-02-25

Last week I introduced the Huffman compression algorithm, along with an implementation in Clarion. This week I'll conclude that discussion, beginning with the assignment of binary codes.

Step 3. Assign Binary Codes.

To assign the binary codes the Huffman Tree must be traversed:

```
huffTree.Traverse     PROCEDURE(*UNSIGNED[] pcde, *UNSIGNED[] pln)
  CODE
  SELF.Trav(pcde,pln,SELf.huffRoot,0,0)

huffTree.Trav  PROCEDURE(*UNSIGNED[] pcde, *UNSIGNED[] pln, |
                  LONG h, UNSIGNED cde, LONG ln)
ch  USHORT
  CODE
  IF SELF.HuffL[h] = 0
    ch = SELF.HuffR[h]
    pcde[ch] = cde
    pln[ch]= ln
  ELSE
    cde = BSHIFT(cde,1)
    ln += 1
    SELF.Trav(pcde,pln,SELF.HuffL[h],cde,ln)
    SELF.Trav(pcde,pln,SELF.HuffR[h],cde+1,ln)
  END
```

The `pcde` array and the `pln` array are declared to 255, which is the number of possible ASCII characters. The first call to `huffTree.Trav` passes both empty arrays, the root node, which

is the highest index value of the Huffman Tree (25) a code of 0, and a code length (Tree level) number of 0. Looking back at Figure 4 in last week's article, the root node of the Tree contains a left value of 23 and a right value of 24, which denotes it as a parent node. As this isn't a leaf, a zero bit is added to the cde, and the `ln` (length or level) value is incremented by 1 to 1.

A recursive call is then made to the left passing the still-empty arrays, the left sub child index value, a code of '0', and a line number of 1. The left sub child contains a left value of 5 and a right value of 20, making this node a parent as well, so another 0 bit is added to the code and the `ln` number is incremented to 2.

The next recursive call to the left passes the still-empty arrays, the left value of the current node (5), a code of '00', and a length of 2. Node 5 of the Huffman Tree contains a left value of 0 and a right value of 101. This node is therefore recognised as a leaf, which relates to a valid character 101 or 'e' from the input file. The code for position 101 in the `pcde` array is then assigned the now applicable code of '00'. The `pln` (code length) array is assigned the `ln` value of 2. The code length is used in recognising how many bits of the stored code are relevant in writing the compressed file.

The method returns at this stage and a call is made to the right sub child. This time the code passes the code and length arrays, the right value of 20, a code of '01' and a length value of 2. The left value of the current node is 7 and the right value is 17, which means that this node is a parent node. Another zero bit is added to the code giving '010' and ln is incremented to 3. Another recursive call is made to the left passing the value 7. Node 7 in the Huffman tree is a left node containing a left value of 0 and a right value of 108, which relates to the character 'l', so the `pln` array for index position 108 is updated with the value 3 and `pcde` array for index position 108 is updated with the valid code 010. The procedure continues allocating codes for all the leaf nodes.

Step 4. Write the frequency array.

Writing the frequency array as a header to the compressed file allows the decompression algorithm to rebuild the Huffman Tree, and therefore decipher the codes.

```
Write:FreqFile          ROUTINE
  DATA
MAX_SEQUENCE_LENGTH EQUATE(63) !Highest number holdable in 6 bits
SeqLen              BYTE(0)!6 bits=sequence length,2 bits = format
lenCode             BYTE(0)
!holds the format, based on the frequency count 00,01,10,11
UCHAR               EQUATE(BYTE)
jLower              UCHAR(1)
!Lower bound of range in frequency array
jUpper              UCHAR(255)
!Upper bound of range in frequency array
j                   ULONG(0)
```

```
j0                  ULONG(0)
f                   ULONG(0)!frequency & uLong place holder for output
u                   USHORT(0) !Ushort place holder for output
b                   BYTE(0)   !Byte place holder for output
  CODE
!Find the smallest range of the array that we have to deal
! with by finding the lowest and highest elements.
  LOOP jLower = 1 TO MAXIMUM(dat:freq,1)
    IF dat:freq[jLower] <> 0 THEN BREAK.
  END
  Cancel = ios_exp.WriteFile(ADDRESS(jLower),SIZE(jLower))
  IF ~Cancel
    LOOP jUpper = MAXIMUM(dat:freq,1) TO 1 BY -1
      IF dat:freq[jUpper] <> 0 THEN BREAK.
    END
    Cancel = ios_exp.WriteFile(ADDRESS(jUpper),SIZE(jUpper))
  END
!Output the frequency array in compact form by breaking the
! calculated range into subranges that meet the same format
! from FindLenCode format can be 00, 01, 10 or 11 = zero,
! 1 byte, unsigned short, unsigned long.
  j0 = jLower
  LOOP WHILE j0 <= jUpper
    IF Cancel THEN BREAK.
    LenCode = FindLenCode(dat:Freq[j0])
    j = j0 + 1
    LOOP WHILE j <= jUpper AND j - j0 < MAX_SEQUENCE_LENGTH |
          AND FindLenCode(dat:Freq[j]) = LenCode
      j += 1
    END
!2 bits contains format, 6 bits contains the sequence length
!MAX_SEQUENCE_LENGTH = 63 as that's the highest number
! possibly held in 6 bits.
    SeqLen = LenCode
    SeqLen = BSHIFT(SeqLen,6) + (j - j0)
    Cancel = ios_exp.WriteFile(ADDRESS(SeqLen),SIZE(SeqLen))
    LOOP WHILE j0 < j
      IF Cancel THEN BREAK.
      f = dat:freq[j0]
      j0+=1
      CASE LenCode
      !1 = BYTE, 2 = USHORT, 3 = ULONG
        OF 1
          b = f
          Cancel = ios_exp.WriteFile(ADDRESS(b),SIZE(b))
         OF 2
          u = f
          Cancel = ios_exp.WriteFile(ADDRESS(u),SIZE(u))
        OF 3
          Cancel = ios_exp.WriteFile(ADDRESS(f),SIZE(f))
      END
    END
  END
```

The first thing that this piece of code does is find the smallest range within the frequency array.

There is little point in writing out a series of zeroes if it isn't needed, so in the example `jLower` is 32 [space], and `jUpper` is 121 'y'. The loop then iterates through the remainder of the array from 32 to 121.

The `FindLenCode` function is used to identify how many bytes, or which data type, to write/read from the compressed file, when writing/reading the frequencies, without wasting space. It is 3 bytes if the value is greater than 65535, it is 2 Bytes if the value is greater than 255 but not greater than 65535, and it is 1 Byte if the value is less than 255 but not zero, and zero bytes if the value does not exist.

The source for `FindLenCode` is:

```
USHRT_MAX       EQUATE(65535)
UCHAR_MAX       EQUATE(255)
  CODE
  RETURN CHOOSE(pVal > USHRT_MAX, 3, |
    CHOOSE( pVal > UCHAR_MAX, 2, |
        CHOOSE(pVal <> 0,1,0)))
```

With my example the first `LenCode` is 1 as the frequency of 32 is only 7. The nested loop is to ascertain how many more in the series have the same `Lencode`. Unfortunately there are no more, as the 33 did not occur in the example. The first two bits of the `SeqLen` variable are made to equal the format '01', and the next six bits of the `SeqLen` variable are made to equal the sequence length of 1; `seqLen` equals 01000001. The frequency for 32 (7) is then output to the file in a one-byte space and j0 is incremented to position 33.

The `LenCode` of position 33 is 0, as there is no frequency for this, the same holds true up to position 83. This time `SeqLen` contains 00 in the first two bits, and 110010 (50) in the last six bits. The `SeqLen` variable is written out to the file, but when the second nested loop iterates, incrementing j0, no data is actually written as the `lenCode` is 0.

Returning to the main, loop `j0` now equals 83, which has a `LenCode` of 1, and is written on its own in the same way that 32 was written. Between 83 and 97 there is another series of zeros. The `seqLen` variable is written saying the format is '00', in the first two bits, and there are '001110' (14) zeros in the sequence.

Now `j0` equals 97 and there are two array elements in a row, which have a code length of 1, or code format of '01', so the `seqLen` variable is written to the file containing these details. The second nested loop sees that the frequency for both of these values is written within their own byte space, the frequency of 97 being 2 and the frequency of 98 being 1.

The process continues to the used upper limit of the frequency array, as stored in the value `jUpper`.

Step 5. Read the file again and write the relevant codes.

```
Write:File          ROUTINE
  DATA
j                       LONG
i                       BYTE(0) !Placekeeper
k                       BYTE(0)
t                       UNSIGNED(0)
b                       BYTE(0)

  CODE
  Cancel = ios_imp.SetPointer(0,0)
  IF ~Cancel
    LOOP
      IF ios_imp.ReadFile(ADDRESS(loc:impLine),1) THEN BREAK.

      t = cde[val(loc:impLine[1])]
      j = ln[val(loc:impLine[1])]
      LOOP i = j TO 1 BY - 1
        !last bit postion of b = bit position i of t.
        b = BSHIFT(b,1)
        b += BSHIFT(BSHIFT(t,31-(i-1)),-31)
        k += 1
        IF k = 8
          Cancel = ios_exp.WriteFile(ADDRESS(b),SIZE(b))
          k = 0
          b = 0
        END
      END
    END
    b = BSHIFT(b,8 - k)
    Cancel = ios_exp.WriteFile(ADDRESS(b),SIZE(b))
    Cancel = ios_exp.WriteFile(ADDRESS(k),SIZE(k))
  END
```

The code above loops through the input file, finds the relevant code for each character read and then outputs the resultant codes one byte at a time.

In the example *"Steve sells sea shells by the sea shore"*, 'S' is the first character read, which has a code t of '10000' and a code length j of 5. The nested loop iterates for the code length, appending each bit to that length on to the end of B. B starts by being '1', the next iteration has B as '10' until all five bit places have been appended.

Variable k keeps a count of how many bits have been appended to ensure that once a full byte is available it is written to the compressed file. When the code for 'S' is appended to B, k equals 5, so the next character is read, which is 't', which has a code t of '0111' and a length code j of 4. So bit positions 1 to 3 are appended to B giving '10000011', at which point the byte is full, k equals 8, and needs to be written to the compressed file. K is then reinitialised to zero, as is b, and the final 1 of the code for t is then appended to the now clear b variable.

Another letter is read from the file, this time ‘e’ which has a code t of ‘00’ and a code length `j` of 2. Both relevant bit positions are also appended to the `b` variable giving ‘100’ There are still five positions left in the byte, so it isn’t written to the compressed file and another letter is read, this time ‘v’ which has a code value t of ‘10100’ and a code length `j` of 5. The entire code is appended to the `B` variable giving ‘10010100’ before it is written to the compressed file.

The process continues until after the last character is read and its relevant code assigned. If at the end of the loop `k` is less than eight, then the remainder of the `B` variable is appended with zeros and written to the compressed file. Then the number of required zeros is also written to the file. This ensures that the decompression algorithm does not try to add additional characters e.g. characters with codes of all zeros to the end of the decompressed file.

Summary

The Huffman compression algorithm can be expanded to include more than one file, and to recognise recurring patterns rather than just single characters for very large files.

In my next article I will cover the somewhat simpler process of decompression.

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 (ClarioNET) 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

Debugging Queues With Excel

by Alan Telford

Published 2003-02-27

To the Clarion programmer, queues are the greatest thing since sliced bread (so what was the greatest thing *before* sliced bread?). They are one of my favorite features in Clarion. With queues I can easily read multiple files (or tables), consolidate, summarize, re-sort into any order, and then use the results. Most of my reports use queues. The main disadvantage I have with queues is the work involved to view their contents.

One report that uses queues a lot is my Stock Variation report. This report needs to calculate the actual stock movement (from delivery, waste and stock count information) and compare it with the expected stock movement (from product sold information and the quantity of stock items contained in each product). To do this I need to read about eight different files and summarize the results. This is a complex report, and recently I had to field a number of support calls for it. Some users were getting large stock variations which were incorrect, and it took a lot of work on my part to track down the problem.

My first approach was to use the Clarion debugger, trace the program, view the queue contents for each stock item and compare with what I expected it to be. But a report often has 2-300 stock items, and the queue is processed in code order, whereas the report displays in name order. This very quickly became error-prone and rather boring, but I persevered for the first couple of calls. Oh, if only there was an easy way to view the contents of a queue.

By the third support call I had modified my approach to traverse the queue, and write the contents out one record at a time into an ASCII file. I saved a lot of time by not having to use the debugger, but the code was specific to this report and this queue and couldn't be reused anywhere. Oh, if only there was an easy way to view the contents of a queue.

On other procedures I had used a special key combination to open up a window which uses a

listbox to display the queue. Again, this is highly specific in each case. Oh, if only there was an easy way … (I'm sure you get the point by now!)

The design

As I thought about the solution some more I decided to list my requirements for debugging queue contents. (yes, even the lazy programmers do occasionally do some design work). The requirements are as follows:

- View any queue
- Sort queue in any order
- Print the queue
- Add calculated columns based on queue contents
- Sum/average of any column
- Be transparent to the user (only the programmer needs to see the debug information)

and most of all

- Be really easy to implement – ideally only one line of code

Looking at the requirements gave me one of those magic Aha! moments. I already had a tool which provided those features: Microsoft Excel. Excel could already sort in any order, add calculated columns, and produce sums and averages and print results. All that I had to do was export my queue to Excel, and that can easily be done using a CSV file.

A CSV (comma separated values) file is an ASCII file which contains one line per queue record, where the line contains the contents of each queue field separated by a comma. And if the CSV file extension has a Windows association with Excel, then I can double-click on a CSV file and it's automatically opened inside my spreadsheet. So how do I do this?

The Clarion `WHAT(group, number)` statement is all that's required to retrieve the contents of a queue. The `WHAT` statement returns the field specified by *`number`* from the *`group`* structure. *`Group`* is the label of a `GROUP`, `RECORD`, `CLASS`, or `QUEUE` declaration. By assigning the `WHAT` to an `ANY` variable and testing for `NULL`, I can easily tell when I've passed the last field in the queue. The following code will create a string which contains the contents of each field in `myQueue`, separating each field with a comma.

```
Line = ''
Ndx = 0
loop
  Ndx += 1
  AnyVar &= WHAT( myQueue, Ndx)
  if AnyVar &= Null then break.
```

```
    Line = Line & choose(~Line,'', ',') & AnyVar
  End
```

All that's needed now is to create the file, read each record in the queue, and write to a file. For convenience I wrap this code in a generic procedure that can be called from anywhere.

```
ExportQtoCsv    Procedure(Queue p:Q)
!-----------------------------------------------------------------
ExportFile      File,Driver('Ascii'),create,Name('DebugQ.csv')
Record          Record
Line              String(4096)
                . .
AnyVar          Any
Line            Cstring(4097)
Ndx             Long
QueueNdx        Long
  code
  create(ExportFile)
  open(ExportFile)
  loop QueueNdx = 1 to records(p:Q)
    get(p:Q, QueueNdx)
    ! create string with queue contents
    Line = ''
    Ndx = 0
    loop
      Ndx += 1
      AnyVar &= WHAT( p:Q, Ndx)
      if AnyVar &= Null then break.
      Line = Line & choose(~Line,'', ',') & AnyVar
    end
    ExportFile.Line = Line
    Add(ExportFile)
  end
  close(ExportFile)
```

Here in 28 Clarion source code lines is a simple procedure which will take any queue, and export the contents into a CSV file called DEBUGQ.CSV, which can then be easily read by Excel. To use the procedure simply call it, passing the queue:

```
ExportQtoCsv( MyCustomQueue)
```

I immediately imported this procedure into a number of apps and put to use (you can do the same with the TXAs in the downloadable source – more about this at the end of the article).

Once I added this new tool to my toolbox, I wondered how I ever did without it. But of course, I soon noticed a whole lot of improvements that could be made:

1. Allow for a configurable filename rather than a fixed "DEBUGQ.CSV"
2. With larger queues, it would be nice to list the names of each column in the first row of the CSV (header record)

3. String values can contain commas (e.g. addresses, product names) and should be surrounded by quotes, so the CSV file doesn't split the value into two or more fields where the comma occurs
4. Date/Time fields by default display as the numeric value (e.g. 76351). An option is needed to specify a format picture for each field.

As this stage I like to change hats. Instead of being the programmer of the `ExportQtoCsv` procedure, I switch to being the user. This allows me to design the way `ExportQtoCSV` should be used and how it is called before I actually make the program changes. Here's what I added:

A **configurable filename** – I supported this by with an optional second parameter.

```
ExportQtoCsv PROCEDURE(Queue p:Q, <String p:Filename>)
```

This allows the procedure to be called multiple times for different queues without overwriting the file.

Listing column names – I do this always. If the column names are not required, then the first row can be deleted from within Excel after opening the CSV. The function gets the column names with the `WHO` statement.

Commas in data – I solved the problem of a comma in the middle of text by surrounding all fields in quotes. Optionally you could use the `ISSTRING` function to only surround string fields with quotes.

Date/Time pictures – I added a third omittable parameter specifies the format picture for each column. Use a pipe delimiter (|) between the format pictures so they can be combined into one long string.

```
ExportQtoCsv PROCEDURE(Queue p:Q, <String p:Filename>, <String p:Format>)
```

Then the procedure can be called as follows:

```
ExportQtoCsv( QueueName, 'debugQ.csv', '@s20|@n7.2|@d1|@t7|@s10')
```

This assumes the queue has five fields: a string, number, date, time, and a string.

Omit the picture (but keep the | delimiter) if you want to use the default picture for that column. E.g.

```
ExportQtoCsv(QueueName, 'debugQ.csv', '||@d1|@t7')
```

This assumes the queue has at least five queue fields, with the third field being a date, and the

fourth field being a time. The default format is used for all other queue fields.

Usage

To use the ExportQtoCSV procedure, simply import the TXA from the downloadable source (Export_Leg.txa for legacy applications, or Export_Abc.txa for ABC applications) into your application. Then in any embed point, add source code to call the procedure passing the queue name, and optionally a filename, and a pipe-delimited list of format pictures.

Note that for Legacy applications you will probably need to add the ASCII database driver to your application. From the project editor (**Project |Edit**) select the Database driver libraries icon, press **Add File**, and select the **ASCII (Text)** entry as shown in Figure 1.

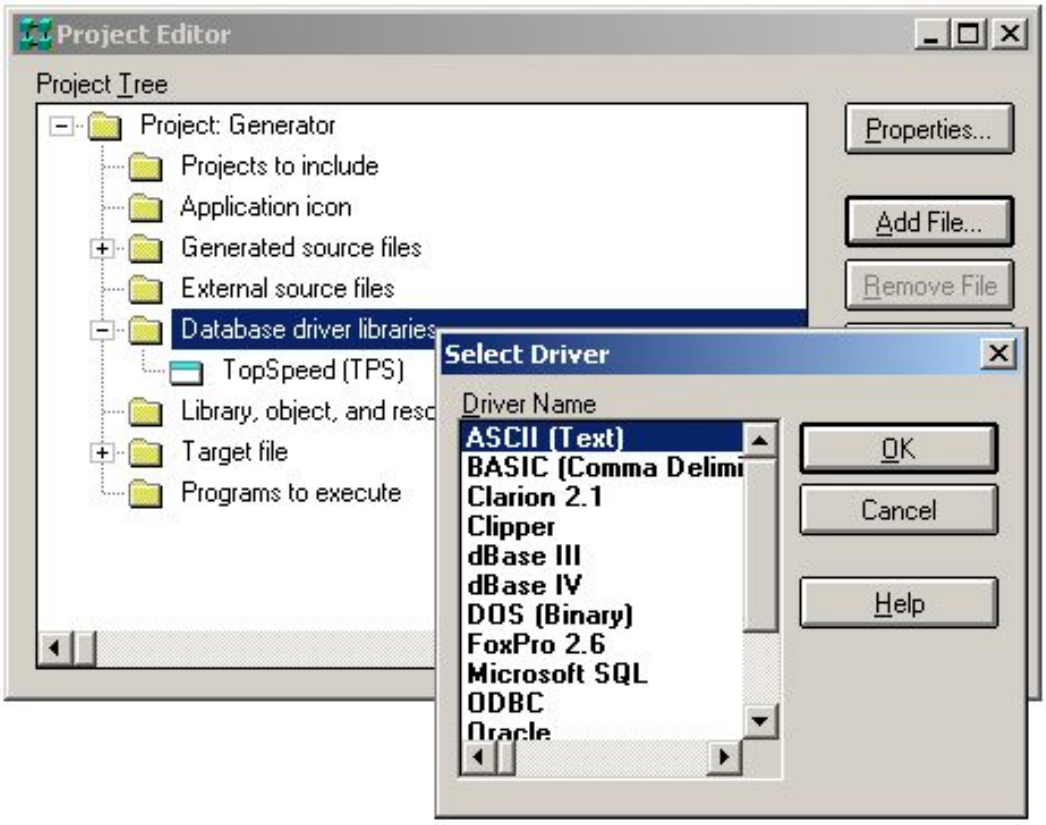


Figure 1. Adding the ASCII driver to the project

As a side note, you may be interested to know that the Legacy and ABC TXAs are identical apart from changing the fifth line from

```
FROM ABC Source
```

into

```
FROM Clarion Source
```

Summary

First I came up against the problem – I'm lazy and I don't like doing more work than required. Then I came up with the requirements – using one line of code, view the contents of any queue, in any order, in a flexible manner, including being able to add calculated columns and sums of columns.

Then in an Aha! moment, I realized I had a tool in my box which already did this, and it wasn't Clarion.

Now I solved the simpler problem of exporting the queue to a CSV file, abstracted the code to a procedure, and then added refinements to the procedure as they were needed.

I hope you enjoy using Clarion queues as much as I do, and if you ever need to quickly see what the contents of a queue are, now you have an easy way of doing this.

Download the source

Alan Telford has been programming in Clarion since 1994. He is the Chief Software Developer at Maxtel Software Ltd, a New Zealand software company specializing in writing back office computer solutions for McDonald's Family Restaurants and other similar markets.

Reader Comments

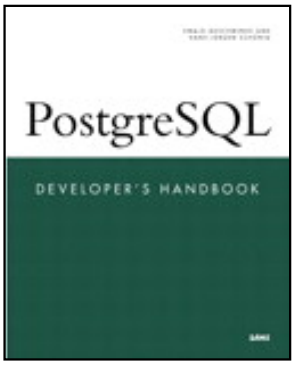
Add a comment

Alan, A wonderfull article. I'm already found an...
Very nice job Alan....Will be used extensively!
Hi Alan, If you use DRIVER('ASCII','/TAB=-100') and add...
Hi Patriek Thanks for that information. That's very...
There is an easier Way!!!!!!! You can view all Table...
There is an easier Way!!!!!!! You can view all Table...

Book Review: PostgreSQL Developer's Handbook

by David Harms

Published 2003-02-28



PostgreSQL Developer's Handbook

By Ewald Geschwinde and Hans-Jüergen Schöenig
Published by Sams, December 2001
ISBN: 0672322609
768 pages, $44.99 US, $67.95 CA, £32.99 UK

PostgreSQL Developer's Handbook, at some 750 pages, is one of the longer and more detailed books on PostgreSQL currently available. Not that there are a lot to choose from – a search on Amazon for PostgreSQL books yielded 11 hits to arch rival MySQL's 49.

The lesser part of this book is taken up with the nuts and bolts of common SQL operations as defined by PostgreSQL. Chapter one covers basic concepts, and chapter two installation on Unix/Linux and on Windows (the latter using CygWin – ugh! – happily a native Windows version is expected this summer, and a beta is available now). Chapter three is a longish one dealing with the basic SQL commands for creating databases, tables, and other components, adding and modifying data, and retrieving data. There's a lot of fairly detailed information here, including topics such as self joins, casting data types, and working with arrays, BLOBs, and other special data types. There's even a section on modeling databases.

After a very brief treatment of transactions, Handbook goes on to a topic that I've found of considerable personal interest lately: the PL/PGSQL programming language. Unfortunately this is also a short chapter – if you get into writing stored procedures in a big way, you'll

probably spend more time with the standard docs. Database administration, backup and recover, and performance tuning all get their own chapters, none extensive, but generally sufficient. The performance tuning chapter deals mainly with database design issues (when and how to create indexes, optimizing queries, using EXPLAIN and VACUUM) , and touches on system issues such as file systems and buffers.

There are some long, long chapters in this book. The granddaddy of them all is Programming Interfaces, which spans almost 200 pages (a book in itself!) and provides an overview of using PostgreSQL from a half-dozen or so languages, and with ODBC. Only the latter is likely to be of interest to most Clarion developers, so there's a good chunk of this book you probably won't care about. The last chapter in Part 1 summarizes software add-ons for PostgreSQL, such as a cube datatype (you can, for instance, select the union of two cubes), full text indexes, ISBN/ISSN numbers, a utility for dumping large objects, a soundex module, and more.

If you've read this far through the book you're about two-thirds done; the last third is devoted to real-world examples. I won't go into all the details, but the topics include: working with EBCDIC data; multidimensional data structures; classifying and aggregating data; generating Flash content with PHP and PostgreSQL; running the PostgreSQL regression tests (mainly useful to the PostgreSQL developers, but you may want to do this to ensure that the latest beta works on your system); extending PostgreSQL with C functions; creating custom datatypes; creating operators (for instance, overloading the + operator to allow for adding RGB color values to get a new color); writing new rules (PostgreSQL is a highly configurable database); handling date and time calculations; persistent database connections (using PHP); using ODBC; and finally, graphing PostgreSQL data using gnuplot.

The PostgreSQL Developers Handbook is a heavy volume with a lot of information on an incredible variety of subjects. It may not tell you everything you ever want to know about the SQL in PostgreSQL (although it will come reasonably close), but it will open your eyes to the great breadth of functionality available in this popular open source database.

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

I just thought that other ClarionMag readers might be...