CenturaPro™

Formerly Gupta Pro | Hot Ideas for Centura® Developers

Centura Code Warrior

R. J. David Burke

February 1997

Volume 2, Number 2

PRO Publishing

Every once in a while, it's useful to be reminded of some of the most basic programming rules, guidelines, practices, and disciplines that help us excel as software developers. This article deals specifically with Centura Team Developer and SQLWindows, but, at a higher level, many of the items discussed can be applied to most other software development environments. You'll also find warnings and advice for those who need to prepare for Tomahawk.

If you develop applications for a living, regardless of the tool you use, you are faced with a significant number of issues in addition to the technical issues associated with programming. There are pressures to "get the product out the door": provide a stable, (one hopes) bug-free product with a reduced time to market. There are the issues involving the development support environment such as repositories, version control, and configuration management, and the QA environment such as test plans and testing tools. A large part of all this comes down to time spent debugging applications. Therefore, much research has gone on in finding ways to reduce time spent in "debug mode."

This article doesn't present anything new or particularly innovative; it's more of a visit to those early days when we were taking courses in programming and system development. Also, you may not necessarily agree with all the ideas presented here, but at least it will start you thinking about these issues and considering how you want to deal with them.

Coding practices

We all develop our own programming style and preferences, and this is, indeed, part of reason there are many rich and innovative applications out there. So the last thing I would recommend are rigid programming rules that stifle our right-brain thinking. But there are many useful practices discovered over the years that don't intrude upon our creative nature. Let's explore a few of these.

Naming conventions

Effective naming conventions give developers significant details about a variable or window in just a glance. You can quickly discern the variable data type, where it's defined, how it's used, and what it abstracts with a well-developed set of conventions.

Most SAL developers start with the naming conventions recommended in the *Developer's Reference* manual, reproduced in **Table 1**. In **Table 2** I add some additional conventions that I've taken to. However, I

Continues on page 4

Branching Out

Mark Hunter

When Keith Lowery left Gupta Corp. after six years of good service, many developers regretted the loss of his talents. I lost touch of his activities after that, but it appears that Lowery has been quite busy. He is the founder and Chief Technical Officer of InfoSpinner, Inc., which has just agreed to be acquired by Centura Software Corp. in exchange for 4.5 million shares of Centura stock. In a conversation with *Centura Pro*, he related some of the reasons he decided to rejoin his old employer.

InfoSpinner, headquartered in Richardson, Texas, is in many ways a typical Internet-oriented startup company. At first staff meetings were held in coffee shops and source code was swapped on disk among engineers working out of their homes. Now the company has 14 employees, including eight engineers, and an actual office. Like other startups, InfoSpinner was formed to address obvious needs in the young Internet/intranet market, especially the needs of large enterprises. InfoSpinner created a product called ForeSite, which debuted in January 1996. Software AG was sufficiently impressed by the product to form a partnership with InfoSpinner and begin marketing the product under their name as iXpress. Advance royalty payments from Software AG allowed InfoSpinner to keep growing the company and the product.

What does InfoSpinner offer?

InfoSpinner's main product, ForeSite, is a deployment platform for Web-enabled applications. It offers several sophisticated features, including comprehensive site management, dynamic load balancing, fault tolerance, multi-platform compatibility, security, and the ability to present dynamic page content through links to relational databases and other applications.

What does Centura gain from

Cyber Editor Mark Hunter, **Techno Publisher** Dian Schaffhauser,
Business Manager-o-matic Shelley Doyle, **Production Editor Superhighway** Paul Gould

Centura Pro is published monthly (12 times per year) by Pro Publishing, PO Box 18288, Seattle, WA 98118–0288.

POSTMASTER: Send address changes to *Centura Pro*, PO Box 18288, Seattle, WA 98118–0288.

Subscription information: To order, call Pro Publishing at 206-722-0406. Cost of domestic subscriptions: 12 issues, $119; Canada: 12 issues, $129. Other countries: 12 issues, $139. Ask about source code disk pricing. Individual issues cost $15. All funds must be in U.S. currency.

Centura technical support: Call Centura Software Corp. at 415-321-4484.

If you have questions, ideas for bribing authors, or would just love to chat about what you're doing with Centura products, contact us via one of the means at right.

New Digs on the Web

Centura Pro on the Web
http://www.ProPublishing.com

Editorial Command Post
Phone: 818-249-1364
Fax: 818-246-0487
E-mail: 71460.3142@compuserve.com

Subscription Headquarters
Phone: 206-722-0406
Fax: 206-760-9026
E-mail: 71333.2142@compuserve.com

Mail
Pro Publishing
PO Box 18288
Seattle, WA 98118-0288

Source Code on CompuServe
GO CENTURA, Library 10

the acquisition of InfoSpinner?

First, they gain an entirely new revenue stream. ForeSite will be sold to companies that had not previously done business with Centura. Since the death of runtime license fees, it has been difficult for tool companies to get on-going revenue from their customers. Since ForeSite is based on deployment, not development, it represents a new form of revenue. It's also a product that's a natural choice for OEM deals like the one with Software AG.

Of course, ForeSite is also a near-perfect fit for Centura's plan for large-scale deployment of Web-enabled applications created with Web Developer. Without ForeSite, Centura would have needed to allocate lots of resources to enhancing the JAWS server with the features found in ForeSite— resources taken away from the ongoing tool development efforts. And, to industry analysts, ForeSite is more likely to be well-received than any deployment platform Centura might have created internally.

Centura management says ForeSite gives them an opportunity to visit accounts that hadn't previously chosen Centura tools. It is essential for Centura Software to increase its customer base, and ForeSite should open many doors.

Why did InfoSpinner choose to be acquired?

Lowery believes that there has been a fundamental change in management's vision at Centura recently. He says that they appear to have a solid plan for gaining market share, and their technical directions are aligned with that plan. Remember that InfoSpinner shareholders are accepting Centura stock, not cash. Lowery thinks that Centura stock, at its current price, is a good growth prospect. Considering that the stock price rose 25 percent on the news, in spite of the diluting effect of an additional 4.5 million shares, the market shares those sentiments.

And InfoSpinner encountered challenges in remaining independent. In the same way that Centura faced a lot of hard work to enhance Web Developer deployment, InfoSpinner faced a lot of hard work to add more and better connectivity and other basic features, to ForeSite. Much of Centura's existing technology can be put to work quickly to make ForeSite even more attractive to potential customers. Lowery noted the good fit between the two companies' products.

InfoSpinner's future

The InfoSpinner office in Richardson will become a branch office of Centura, and the staff will remain to work on ForeSite and related products. Lowery sees no sudden shift in direction for his engineers; Centura is interested in InfoSpinner's products and wishes to keep enhancing and marketing them. The products will remain open to all platforms and development tools. And Software AG, which kept InfoSpinner going in the early days, is happy to see increased stability and functionality for the product; their expectations for ForeSite (iXpress) remain strong.

Why does the InfoSpinner Web page look so funny?

ForeSite is becoming a Centura product, and Centura wants to create a coherent marketing plan for this important new part of the company. While that's going on, they have asked InfoSpinner to suspend its public marketing efforts. Thus, the empty look of www.webspinner.com. OEM vendors may still publish product information for the current version of ForeSite, but there really will be significant changes to ForeSite between now and the official Centura rollout later in the first quarter. At that time a revised set of product specs and other information will become available. You can also expect an in-depth look at ForeSite's features in *Centura Pro* at about the same time.

Reading between the lines

R.J. David Burke's "Code Warrior" article in this issue is full of good advice, but the most interesting tidbits are those which discuss how Tomahawk and its conversion tools will deal with your existing SQLWindows and Centura applications. It's not too early to start thinking about what you'll need to do to make your apps compatible with the future Centura Team Developer 2.0. At Pro Publishing we've seen a little of the future, and it's both daunting and exciting. Preparing for Tomahawk will be a major focus of the newsletter in the months to come. **CP**

Centura News

Real-time Goes Graphical

DataViews Corp. recently announced DV-Xpresso 2.0, a real-time graphics package that allows developers to create and customize cross-platform 2-D and 3-D dynamic data displays for decision support. The graphics can be built and deployed to the Web, Windows, or Unix. Developers can use the utility to control, monitor, and represent data for executive information systems in industrial and financial applications. The package, which consists of 120-plus standard charts and graphics, downloads drawings once; the dynamic data then modifies the drawing, based on a frequency determined by the programmer. Pricing starts at $1,000. The product is expected to ship this quarter. To learn more, call (800) 732-3200 or (413) 586-4144 or visit www.dvcorp.com. **CP**

Centura Code Warrior . . .

Continued from page 1

think there are still some conventions to work out.

Additionally, you can use a naming convention for constants that don't fit well into the referenced tables. This convention is to start with an upper case prefix, typically two or three characters, that identifies the constant group and then an underscore, followed by a descriptive, proper-cased phrase:

```
SAM_DoubleClick
```

Table 1. SAL Identifier naming conventions.

Item type	Object type	Name prefix	Example
Variables	Boolean	b	bDone
	Date/Time	dt	dtStart
	File Handle	hFile	hFileLog
	Long String	ls	lsImage
	Number	n	nProfit
	Sql Handle	hSql	hSqlQuery
	String	s or str	sLastName
	Window Handle	hWnd	hWndMain
	globals	g + type	gbDebug
	arrays	a + type	hWndaChildren[*]
Windows	Form Window	frm or fw	frmMain
	MDI Window	mdi	mdiFrame
	Table Window	tbl or tw	tblMain
	Dialog Box	dlg	dlgLogin
	Data Field	df	dfPassword
	Multiline Field	ml	mlComment
	Column	col	colROWID
	Pushbutton	pb	pbClose
	Radio Button	rb	rbFemale
	Check Box	cb	cbSaveInPreferences
	Option Button	ob	obTableView
	List Box	lb	lbDatabases
	Combo Box	cmb	cmbTitle
	Child Table	tbl	tblDetails
	Horizontal Scroll Bar	hsb	hsbPan
	Vertical Scroll Bar	vsb	vsbAspect
	Picture	pic	picLogo
	Custom Control	cc	ccMeter

Table 2. Additional popular naming conventions.

Item type	Object type	Name prefix	Example
Resources	Bitmap	bmp	bmpSplash
	Icon	ico	icoCustomer
	Cursor	cur	curDragCustomer
Functions and Top Level Windows	Parameters	p_ +	p_RepeatCount
Classes	Class Names	c or cls	cExplorerList
	Class Variables	c_ +	c_hWndInstances
	Instance Variables	i_ +	i_sUserName
Named Menus	Menus	menu	menuFile

Some developers have devised schemes for naming functions, but I haven't found one that suits me as far as prefixing a function; so I just use unprefixed, descriptive proper case names. One area where I deviate from the common SAL-naming conventions is in the naming of classes. Personally, I don't find it very useful to prefix class names with a c or cls. As with functions, I use an unprefixed, descriptive proper case phrase, as shown in **Listing 1**.

Listing 1. Declaring classes and custom controls.

```
Global Declarations
          .
          .
          .
          Class Definitions
            Custom Control Class: Slider
              Description: Slider control
              Derived From
              Class Variables
              Instance Variables
              Functions
              Message Actions
          .
          .
          .
        Form Window: frmMain
          Description: Main application
            window
          Named Menus
          Menu
          Tool Bar
          Contents
            Slider: sldrVolume
              Message Actions
          Functions
          Window Variables
          Window Parameters
          Message Actions
```

Listing 1 also shows another practice I like, which is to prefix custom control class instances in a similar manner as the standard controls. Thus, all instances of the Slider custom control class are prefixed with “sldr.”

Another useful technique is to organize your variables. I like to organize them alphabetically. This makes it fairly easy to locate variables when browsing code. I used to organize variables in the order they were referenced, like in my structured programming days; but this is harder to browse and often unknown in event-driven environments.

I would also like to point out that naming conventions are guidelines rather than rules. This makes it easier to allow useful exceptions. Perhaps the best example of such an exception is in variable names for numerics that are really handles. For these variables I generally choose to go with h as the prefix as in **Listing 2**.

Listing 2. Allowing exceptions where appropriate.

```
Set hSysMenu = GetSystemMenu( hWndMDI )
```

I now find myself moving toward using Java naming conventions for several reasons. First, the Tomahawk (2.0) release of Centura Team Developer (CTD) will be quite Java-centric (generated Java code, support for Java classes). Second, I think Java's style is more readable than the legacy SAL conventions. Consider isDone instead of bDone and numItems instead of nNumItems.

SAL supports up to 32 characters for identifiers. This should give you the freedom to come up with meaningful, descriptive names; but it must be balanced against using names that are too long and difficult to read. Remember, you want to get maximum expressiveness out of variable names, not maximum character length.

Comments

The application outline includes many predefined items for entering comments and descriptive text. However, I've seen source code for many production programs in which description items are mostly blank. Given that typically a program is read many times but written once, it only benefits us to document the code.

In the outline description items I like to use a format similar to the Windows SDK documentation for functions and a somewhat modified version for top-level windows and classes. In-code comments are really useful as well, especially for documenting non-obvious algorithms and such. However, software development veterans know that comments can never be trusted (typically, when the code is changed by a junior maintenance programmer, who doesn't bother updating the comments) and only trust the code itself. This reality reinforces the adage:

If you can say it in code or say it in comments, say it in code.

What this means is that if you can capture the essence of a code fragment with a comment vs. with descriptive variable names and programming constructs, go with the latter. In self-descriptive code, the code itself will explain how it works. In some cases it is helpful to augment the code with comments that explain why the code does it the way it does.

External function issues

One dilemma with respect to external functions is whether to use export ordinals. Generally, I prefer not to, unless required. If export ordinals are used, the compiler will compile external functions as long as any function has been exported with the specified ordinal, regardless of whether it's actually the function you intended to call. By not using export ordinals, the compiler is forced to search binaries for functions by name, which will cause the compiler to fail if the functions aren't defined and exported. I'm sure export ordinals are in the top three of migration problems when going from SQLWindows to Centura Team Developer. A typical situation (IMHO) that justifies using export ordinals occurs when you need to declare the same external function more than once with different parameters. A good example is the Windows API function SetWindowPos, where you might redefine the second parameter to accept either a Window Handle: HWND or a Number: DWORD (Number: HANDLE in CTD) depending on whether you need to actually position a window or make it topmost.

Another reason for using ordinals is to avoid compile errors for duplicate function names. For example, if you write a general-purpose APL that defines a Windows API function, and you want to include it in an existing application, you must determine whether the function is already defined in some other part of the app, and perhaps modify code. If your APL uses a different, unique name for the function and defines the ordinal, you can include the APL in the app without fear of duplicate symbol errors.

The second dilemma surrounding external functions is whether to map API functions that take a handle data type to a number or a window handle. Both have their advantages and disadvantages. Mapping an external handle data type to a window handle offers some protection from using the value in arithmetic operations, which typically aren't a valid way to use the handle value. However, the value is now susceptible to being used as a window handle argument. By using a number for the internal mapping, you get better expression in CTD because you can choose Number: HANDLE. Also, with the forthcoming changes in the Tomahawk release of CTD, window references will replace window handles, and this may make migration efforts more challenging.

More on functions

While talking about functions, here are some more tips. A useful practice to adopt is to always check and validate your parameters at the start of the function. While some developers prefer to do validation on the arguments before calling the function, from a modular and encapsulation perspective, the function shouldn't count on this.

Another guideline for coding functions is to have only one exit point (i.e. one Return). Most developers who made it through structured programming retain this practice, but the C community seems to have discouraged this style. The benefit is readability. If there's only one exit, then it needs to be at the end of the function (the exceptions are pathological) and can be quickly located.

Multiple Return statements make the code harder to read and typically introduce more code paths that need to covered during testing (discussed in Part 2 of this article).

Finally, get in the habit of explicitly coding a return data type for your functions. While the current releases of SQLWindows and Centura Team Developer default to a numeric return data type, this will change in the Tomahawk release of CTD, which will support functions that return nothing (VOID). (The converter will automatically change functions with no explicit return type to return a Number.)

- Application Description: SQLWindows version 5.0 starter application
  - Design-time Settings
    - Outline Window State: Normal
    - Outline Window Location and Size
    - Options Box Location
    - Class Editor Location
    - Tool Palette Location
    - Fully Qualified External References? No
    - Reject Multiple Window Instances? Yes
    - Enable Runtime Checks Of External References? Yes
    - Use Release 4.0 Scope Rules? No

Figure 1. Centura Team Developer defaults to Fully Qualified External References but SQLWindows does not.

For every open, a close

Whenever you open a file or connect a Sql Handle, ensure that you have a matching close or disconnect. While an open File Handle isn't dangerous (the runtime engine will close the file upon normal termination of the application), detecting an open handle may point you to code that isn't being executed, though you thought it was. Connected Sql Handles at application termination can present problems with some back ends. Suppose you build an application to work with SQLBase for development and forget to disconnect your Sql Handles at termination. You then switch to Oracle for production. A connected Sql Handle to Oracle at application termination can result in a GPF, depending on which version of SQL*Net you're using.

Application properties

In SQLWindows these are items indented under the "Design-time Settings" outline item (see **Figure 1**). In Centura Team Developer, right click on the Application node in the left pane and choose Properties from the menu. Then select the Runtime tab in the properties dialog. In either environment, consider the following options:

- *Use Release 4.0 Scoping Rules.* Always set this to No unless you have an application built with SQLWindows 4.0 and dependent on the scoping rules in that version. If you have such an application, I recommend you convert the app to use the present scoping rules, as it is anticipated that release 4.0 scoping rules won't be supported in the Tomahawk release of CTD.

- *Reject Multiple Window Instances.* My preference for this property is to set it to No (unchecked in CTD). If you leave it as Yes (or checked), then when the application attempts to create multiple instances, it terminates rather abruptly and in an unfriendly manner. If I need to limit window instances, I do it in code.

- *Fully Qualified External References.* I normally set this item to Yes (or checked). Unqualified external references to variables and controls complicate debugging. Of course, references to variables are best avoided altogether if possible, as they violate encapsulation. References are further discussed in the section *Scoping and Qualification*. Unfortunately, this setting has no effect on handle-qualified references (previously known as semi-qualified references).

- *Enable Runtime Checks of External References.* This item is useful to set to Yes (or checked) during development, but set to No for production releases. When set to Yes, the object referred to by a window handle used in a fully qualified reference is checked to ensure that it matches the identifier that follows the window handle in the fully qualified expression. If this is set to No and the check fails, then you're guaranteed to get a GPF and a fatal abnormal termination of your app. If it's set to Yes, you'll get a friendlier error dialog that explains why your app is about to terminate. Certainly helps with the debugging effort. After thoroughly testing your app, you typically set this to No for building the production EXE file to receive a modest performance improvement in code execution of these kinds of references.

See the on-line help or manuals for more information on these items.

Object-oriented programming

OOP isn't something that can be covered in any significant detail in an article like this, but there are important points to remember. In the current implementations of SAL, encapsulation is a discipline you have to maintain. Refrain from external references to variables or child windows. In the Tomahawk release we'll see support for encapsulation from the compiler.

A hot topic in OOP is whether to use multiple inheritance. I believe multiple inheritance has its place and can be useful in many situations. Others disagree and avoid using multiple inheritance.

Strive for reusability. Use classes and inheritance, and partition your library files to optimize the reuse of your work. Get over the not-invented-here-syndrome and look for other code that you can reuse. Consider and evaluate the available class libraries from third parties like FrontEnd Systems, METEX, ADC, and Ten Ham Informatiesystemen.

Scope and qualification

Understanding scope and how to formulate qualified references is one of the more complicated aspects of SAL. Fortunately, if you stick to the principles of object-oriented programming, you can minimize your need to use qualified references and simplify your code. The forthcoming Tomahawk release of CTD will include significant changes in this area, eliminating much of the complexity that exists in SAL today. However, for the present, it is useful to understand the various forms of qualification and how they are used (or avoided).

In the outline there are many and varied places where you can define the elements of an application. I use elements as a catch-all to refer classes, windows, variables, functions, and so forth. Each element has a name or identifier that you use to refer to or manipulate the element. The term used in the *Developers Reference* manual is *symbol*. Depending on where you are in the outline, and what element you want to refer to, you may have to qualify the element's symbol to help the compiler (or runtime engine) understand what element you are specifying.

The scope of a symbol is the part of the outline where you can refer to an element using just its symbol, without any additional qualifications. In SAL, scope is linked to containership, that is, the scope of a symbol is anywhere within the defining element, nested to any outline level. This means, for example, that the scope of a data field in a form window is anywhere inside the form window, including its functions, other child windows, message actions, and menu. The scope of global variables is anywhere else within the outline, because you can think of the outline itself as the highest-level of container.

To refer to an element of an application from outside its scope, you need to use a qualified reference. Generally, I see qualified references as either *external references* (referring to an element defined in a different top-level window or MDI window) or *child references* (referring to an element defined in a child element; this is my own term, not in the documentation). External references can be more specific by making them *fully qualified*, which means a window handle expression is prepended to the qualified reference. Another way to look at qualified references is that they are either *object-qualified* or *class-qualified*, depending on what kind of reference you're trying to make.

What follows here is look at all the forms of qualification, where and why you'd use them, and any other comments.

Object-qualified references

Object-qualified references can take on a variety of forms in SAL. Consider accessing a variable or data field in a form window outside of the current scope:

```
Set frmCustomer.sLastName = frmQuery.sLastName
Call SalHideWindow( frmMain.pbCancel )
```

Notice how both of these lines of code are invasive, violating the spirit of encapsulation. However, object-qualified references to functions are generally acceptable:

```
Set sLastName = frmCustomer.GetLastName(  )
```

Resolving references to duplicate symbols

One of the reasons for object-qualified references is to eliminate the ambiguity of duplicate symbols in different top level windows. For example:

```
Form Window: frm1
    Functions
        Function: fX

Form Window: frm2
    Functions
        Function: fX

Dialog Box: dlg1
    Window Variables
        String: s
    Message Actions
        On SAM_Create
            Set s = fX( )
```

The compiler won't be able to resolve the assignment to s. The solution is to qualify fX with the object name of the form whose fX function is to be invoked:

```
Set s = frm2.fX( )
```

Referring to child symbols in the context of a parent

A suitable example:

```
Form Window: frmMain
    Contents
        Child Table: tblDetails
            Contents
                FlashableColumn: col1
            Message Actions
                On SAM_Destroy
                    Call aTransaction.Active( FALSE )
    Window Variables
        Transaction: aTransaction
```

```
    Message Actions
        On SAM_Timer
            Call tblDetails.col1.Flash(  )
```

The first occurrence of object-qualification is the call to aTransaction.Active( ). The scope of the UDV aTransaction, defined in frmMain, extends to tblDetails. However, the *functions* defined in the Transaction class aren't in the scope of tblDetails, so an object-qualified reference is necessary.

The second occurrence of object-qualification is the call to tblDetails.col1.Flash( ). The context for this call is within frmMain and, since col1 is a child element of tblDetails, it needs to be qualified. Also notice col1 qualifies the Flash( ) function. In an object-qualified reference you can provide as many levels of nesting as necessary for your reference. Arguably, this second occurrence violates encapsulation. Perhaps a better approach would be to send the tblDetails a message or call one of its functions, which would execute the column-qualified function call.

Fully object-qualified reference

In applications that support multiple instances of a top-level window, it is necessary to specify a particular instance in an object-qualified reference. This is supported by prefixing the object-qualified reference with a window handle expression that provides the window handle of the instance you want to use.

Resolving references to multiple instances of a window

Recall the following form window:

```
Form Window: frm1
    Functions
        Function: fX
```

Now consider the following:

```
Set hWnd1 = SalCreateWindow( frm1, hWndNULL )
Set hWnd2 = SalCreateWindow( frm1, hWndNULL )
Set s = frm1.fX(  )
```

The object-qualified reference frm1.fX( ) is ambiguous in this case since there are two instances of frm1. The solution is to go to a fully object-qualified reference that specifies the instance to use:

```
Set s = hWnd2.frm1.fX(  )
```

Additional nesting is supported for fully object-qualified references as well. Here's an example:

```
Call hWndFrame.mdiMain.frmCustomer.
tblContacts.colLastName.Clear(  )
```

Another form of fully object-qualified references handles the situation where there are multiple instances of frmCustomer within mdiMain. An example:

```
Call hWndForm.(mdiMain.frmCustomer).
tblContacts.colLastName.Clear(  )
```

In this syntax, the window handle has to refer to the rightmost symbol inside the parentheses. As another example, if you had the window handle of an instance of colLastName, you would write:

```
Call hWndCol.(mdiMain.frmCustomer.
tblContacts.colLastName).Clear(  )
```

Fully object-qualified references are perhaps the most difficult to formulate in SAL because it's difficult to anticipate (during application design) exactly what is needed and how to structure the application. It is anticipated that the Tomahawk release of CTD will make this form of qualification obsolete.

Handle-qualified reference

Handle-qualified references (originally called semi-qualified references back around SQLWindows 3.x) were introduced to SAL to provide a "poor man's polymorphism" before all the OOP additions that went into SQLWindows 4.0. This form of qualification is now obsolete and shouldn't be used. It can make applications difficult to maintain and debug. Currently, handle-qualified references are still supported to facilitate migration of older applications. However, in the Tomahawk release of CTD, it is anticipated that this form of qualification will no longer be supported. The recommended migration strategy for legacy SAL code is to use late-found function calls. For example, consider a circa-SQLWindows 3.1 application with code like this:

```
Form Window: frmA
    Functions
        Function: Clear

Form Window: frmB
    Functions
        Function: Clear

Call hWnd.Clear(  )
```

where hWnd is the window handle of either an instance of frmA or an instance of frmB. A possible restructuring could look like this:

```
Form Window Class: cFrm
    Functions
        Function: Clear ! possibly a virtual function

cFrm: frmA
    Functions
        Function: Clear

cFrm: frmB
    Functions
        Function: Clear

Call hWnd.cFrm..Clear(  )
```

Continues on page 12

Save Effort and Resources with Forms on Tabs

Larry Stahl

One great change introduced in SQLWindows 5.0.2 (and temporarily left out of 5.0.3) was the capability of associating a form (or dialog) with one particular tab on a tab frame. This article shows how this feature can conserve resources and improve team programming.

Associating a form with a tab is a great improvement over the initial tab forms. Anybody who has worked with tabs would probably agree that the user interface (UI) is great but has contributed to some serious problems. The sole original method by which tabs worked was to have *all* of the children on all tabs belong to the one form that contained the tab. This often led to forms with hundreds of child windows, sometimes reducing resources to 10 percent or less available.

Another problem that crops up is managing the development of the tab form. Usually all the information presented on multiple tabs is related, but it often comes from different tables for different tabs. Because all of the tabs for a particular form are part of one source code module, only one person can work on the form at a time. Often projects are managed by checking in and out modules that control a form; but this doesn't work with individual tabs on tab forms.

Forms on tabs can solve these problems.

What are forms on tabs?

So what, exactly, are forms on tabs? If you've used tabs, you're familiar with the properties dialog. The first five properties were there in prior versions of SQLWindows. They allow specification of the attributes of the tab form such as the labels on the tabs, fonts, and colors of the labels and the internal tab name.

A sixth property was added in 5.0.2 that lets you associate a top-level form (form window or dialog box) with an individual tab. You'll notice in **Figure 1** that there's a window name, the tab label, and the internal tab name (which you can't see here). Also illustrated in the figure is a nice feature which, when you put the focus in the window name column, shows a list of all of the forms from which to choose.

Another feature has been added to ease working with forms on tabs, as shown in **Figure 2**. When a form is associated with a tab, the tab displays a form icon and a little message that a form is so associated. It is easy to see the design window associated with the form by double-clicking on the icon.

Once you've associated a form with a tab, SQLWindows handles the creation of the form when the appropriate tab is clicked. All of the standard SQLWindows messages (such as SAM_Create) occur for this form.

A fortunate combination of features

"OK," you say, "That's fine if a user is only going to use a subset of the total number of tabs each time the tab form is used. But if the form is created each time a tab is

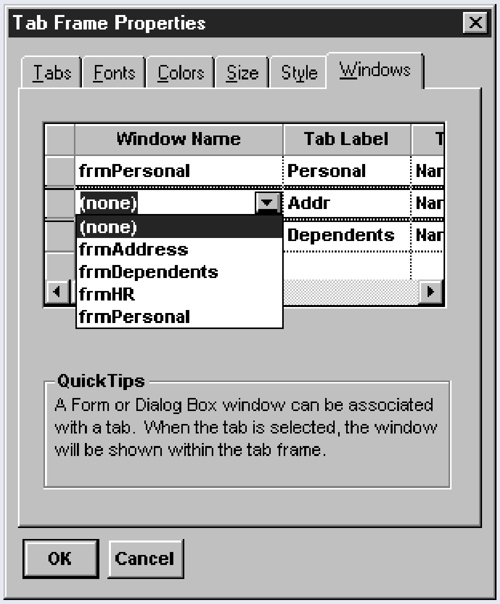


Figure 1. Associating a window name with a specific tab.

chosen, and all of the tabs are selected, we can very quickly get up to using 400 or 500 child windows." Here, a powerful combination of features in SQLWindows or Centura Team Developer proves useful.

To take advantage of these features, each form that is associated with a tab should contain a user-defined variable (UDV). A UDV is a variable that is an instance of a functional class that the developer creates. In the example I've provided, they are instances of classes cfcPersonal, cfcAddress, and cfcDependents. Each functional class has the data required for the particular form (in other words, cfcPersonal, shown in **Listing 1**, contains variables for employee name, ID, and birth date as well as functions to access those variables). Depending upon how the database is structured, the functional class might contain functions to insert, update, and delete the data to the relevant tables in the database. Many additional examples of UDV usage can be found in MUSQLB.APP in the SAMPLES subdirectory of your SQLWindows or Centura directory.

Listing 1. The functional class for Personal form UDV.

```
♦ Functional Class: cfcPersonal
    ◊ Description: Employee's personal information
    ◊ Derived From
    ♦ Class Variables
        ◊ String: siEmpId
        ◊ String: siName
        ◊ Date/Time: dtiBirthdate
    ◊ Instance Variables
    ♦ Functions
        ♦ Function: GetEmpId
        ♦ Function: PutEmpId
        ♦ Function: GetName
        ♦ Function: PutName
        ♦ Function: GetBirthDate
        ♦ Function: PutBirthDate
```

I have found it useful to have a UDV for the entire tab form (like a buffer for a database table or tables) and contribute the UDV from each "child form" to this overall UDV. Then, that UDV might be type cfcEmployee and contain variables of type cfcPersonal, cfcAddress, and

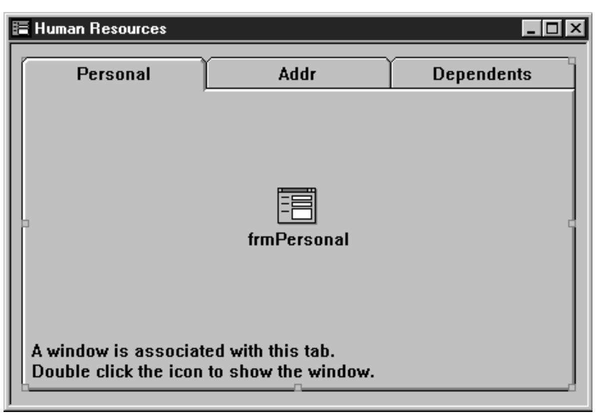


Figure 2. The first tab displays showing a form association.

cfcDependents, *along with others.* This functional class could have functions to manipulate the database for the employee, including such functions as creating a new employee or—ahem—terminating one. This really shows the power of object-orientation.

How do I use it?

When the form is created, read the data contained in the UDV into the appropriate objects (data fields, combo boxes, etc.) on the form. The value of data may be null, so make sure that situation is handled correctly. When the form is destroyed, write the data back into the UDV. The data in a functional class is persistent through this application session. (For example, class variables in a functional class retain their information even when instances of the functional class are terminated.) It's convenient to have one function to read the data from the UDV into the form and one to write the data from the form to the UDV.

When you're in the windows variables section of a form, the functional classes are shown as datatypes in the outline options. The variables section of each of the three forms we have associated with our tabs will have the following defined:

```
cfcPersonal: uPersInfo
cfcAddress: uAddress
cfcDependents: uDeps
```

Using a late-bound function

The last capability needed to make this work is some way of knowing when a user has "clicked off" the tab so that you can save the information from the form and destroy the form. As part of tab processing, the function TabUserRequest is called, late-bound, from the SQLWindows code. This means that if you have a function by the same name in your tab form (the form that contains the tabs), your version of the function will be called, instead of the default version that comes with QCKTABS.APL. The function takes two parameters: a window handle and a number. Centura has implemented tab functionality within the structure of a picture window, picTabs (by default).The window handle is that of the tab frame (often named picTabs), and the number is the index of the tab to which you are moving. If you're experienced with tabs, you'll recognize that this late-bound function simply takes the place of the message, TABSM UserRequest. Also, if for some reason program logic dictates that the user not change tabs, you may call function CancelMode( ), which will deny the user's request and leave the current tab displayed.

From within the TabUserRequest function (see **Listing 2**), call the function hWnd.cQuickTabs.GetTop( ), where hWnd is the window handle parameter originally passed to the TabUserRequest function. Set a numeric variable, say, nWasIndex, to that value. Then call:

```
hWnd.cQuickTabs.GetName( nWasIndex, sTag)
```

This takes the tab index and gets the “tag” or internal name of the tab (remember the unique value you specified when you set up the tabs). Given this tag, you may then call:

```
hWnd.cQuickTabs.DestroyPage(sTag)
```

which destroys the form on the tab but won’t eliminate the association between the tab and its “child” form. When you go back to that tab, the form will be automatically created.

Now you need to store the data that resides on the form into your UDV (that was created for that purpose). When the form is created, there must be code to read data from the UDV and write it on to the form. It works fine to write logic for those two tasks within the SAM_CreateComplete and the SAM_Destroy messages. I recommend creating a form window class that always calls a function to restore data on the create and save the data (to the UDV) on the destroy. By doing it all within the form, the capability works without regard to whether or not the form is associated with a tab. Remember to use class (not instance) variables in the functional classes, because you destroy the instance of the class every time you destroy the form. By writing a function to save the data for each form, you can easily handle special cases such as table windows or extra data stored in form variables.

Drawbacks and caveats

Data is in the UDV, not the form

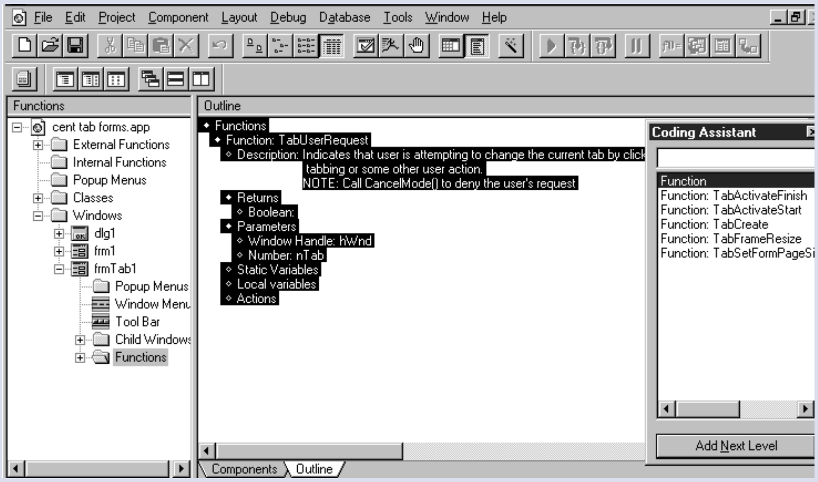
This is a new way of programming. The functional class is the primary location in the program, not the form. The data remains even after the form is destroyed. The reason the variables are class variables is so that even after the instance of the class is destroyed (the form variable), the class variables still exist. Any other instance of the functional class shares in the functions and data in that class. Therefore, the function to write to the employee table (the DB table that holds the employee’s personal information) might be in our cfcPersonal class and used from the main tab, not the personal tab.

No knowledge of parent

The most difficult issue is not really a difficulty but a

Late-bound Functions In CTD

Using late-bound functions is facilitated by Centura Team Developer. When you are in an instance of a class and in the function section of the outline, you can see all of the class functions available to you. If you double-click on the function you wish to implement, a template (name, description, variables) of that function is created in the outline for you.—*L.S.*



Outline view with coding assistant displaying class functions and specific template.

Listing 2. An example of the TabUserRequest function.

```
♦ Function: TabUserRequest
   ◊ Description: Indicates that user is attempting to change the current tab
       by clicking, tabbing, or some other user action.
       Note: Call CancelMode() to deny the user’s request
   ◊ Returns
   ♦ Parameters
       ◊ Window Handle: hWnd
       ◊ Number: nTab
   ◊ Static variables
   ♦ Local variables
       ◊ String: sTag
       ◊ Number: nWasIndex
   ♦ Actions
       ◊ Set nWasIndex=hWnd.cQuickTabs.GetTop()
       ♦ If nWasIndex !=nTab            ! only do something if tabs change
           ◊ Call hWnd.cQuickTabs.GetName( nWasIndex, sTag )      ! get the tag and number tab
           ◊ ! The DestroyPage function destroys the form associated with the tab
               but doesn’t destroy the association; so that when it is again selected
               the form appears.
           ◊ Call hWnd.cQuickTabs.DestroyPage(sTag)
```

different way of thinking about the situation. That is, the form has no direct knowledge of what is happening in the tab (or parent). In many applications the user isn't allowed to save until certain information is supplied. When all fields were on one form, that was fairly straightforward. Now we'll achieve this same capability using variables that share the same class.

Suppose an employee can't be saved to the database until a name, birth date, and address are supplied. If the main tab contains variables of type cfcPersonal and cfcAddress, the solution is (again) to use the functional class or classes that are instantiated by the UDVs used to save the form's data. If the main tab has UDVs that are of the same class as the tab form's UDVs, communication can happen through those same variables. A simple call to:

```
uVarPersonal.GetName()
```

reveals the user's input (or lack thereof). Even combining such queries into one function in the "employee" variable (say, cfcEmployee has a function IsAllReqd( )) would make sense. This entire concept of communications between forms and sharing data can be expanded (and possibly a topic for a later article).

The end of MDI?

An application or programming style is specific to an individual programmer. But in this case I think it's easy to imagine that, rather than using an MDI structure for many applications, using a tab form with individual forms on tabs will be as flexible, and perhaps more intuitive, for a user. There are many functions available to manipulate tabs. Examine the sample application QUIKTABS.APP (in the SAMPLES subdirectory) for information regarding those functions. **CP**

Download FORMTABS.ZIP from the Centura third-party forum on CompuServe or visit the Centura Pro Web site at www.ProPublishing.com.

Larry Stahl is a consultant and founder of InterAct Consulting and a Centura Software Training Partner. He provides business requirements analysis, project management, design, programming services, and training to a variety of businesses. He can be reached at larry.stahl@juno.com or on CompuServe at 71165,1012.

Centura Code Warrior . . .

Continued from page 8

Class-qualified reference

A class-qualified reference is similar to an object-qualified reference except that a class name is used to qualify a reference. Class-qualified references are typically used in derived classes or class objects to reference an element in a base class. Class-qualified references are used to resolve multiple inheritance collisions. An example:

```
Functional Class: A
    Functions
        Function: X

Functional Class: B
    Functions
        Function: X

Functional Class: C
    Derived From
        Class: A
        Class: B
    Functions
        Function: X
            Actions
                Call A.X(  )
                Call B.X(  )
```

An interesting use of class-qualified references is that they can be used to reference class variables even when there are no instances of a class. For example, you can do this:

```
Global Declarations
    Class Definitions
        Form Window Class: cFrm
            Contents
                Data Field: df1
                    Message Actions
                        On SAM_Create
                            Set MyValue = c_s
            Class Variables
                String: c_s
    Application Actions
        On SAM_AppStartup
            Set cFrm.c_s = 'Hello'
            Call SalCreateWindow( frm1, hWndNULL )

cFrm: frm1
```

and, upon execution, df1 in frm1 is initialized to "Hello".

Fully class-qualified reference

A fully class-qualified reference is similar to a fully object-qualified reference, except that a window handle expression is used to specify a particular instance of a class. After being evaluated, the window handle result must refer to an instance of the class, or an object of a class derived from the specified class. Fully class-qualified references provide a fair amount of flexibility and room for growth when designing class libraries or application frameworks in applications themselves. For example, in:

```
Call hWnd[ nIndex ].cFrm..Initialize(  )
```

the window handle result would have to refer to an instance of cFrm or an instance of a class derived from cFrm.

Like fully object-qualified references, it is anticipated that Tomahawk will make the need for, and use of, fully class-qualified references obsolete.

Next month: Test plans and testing. **CP**

R.J. David Burke is a Project Leader with Centura Software Corp. You can send him e-mail via David.Burke@CenturaSoft.com.

Reporting Tools:
The Answer is Crystal Clear

Doug Mitchell

This article reviews Seagate Software's Crystal Reports 5.0, a popular and award-winning reporting tool. This program can be used as a powerful, stand-alone reporting utility or as a report engine integrated into your favorite development environment.

Because of Crystal Reports' scope and complexity, this article focuses only on the features and user interface of the product, to provide you with a good understanding of Crystal's capabilities and limitations. A second article, which will appear in a subsequent issue of *Centura Pro*, will focus on the integration of Crystal with various development environments, particularly Centura's development products.

A review, not a comparison

To begin with, this first article is more of a review of Crystal, rather than a direct comparison of Crystal with Centura's reporting tool (ReportWindows/Report Builder). Therefore, you won't find any comparative feature checklist or performance benchmark graphs. However, since you're probably familiar with Centura's reporting tool, this article informally compares the two tools to each other frequently. While both of these tools are in the same category (client-side reporting tools) Crystal is in an entirely different class. Crystal is the Ferrari of reporting tools—powerful, fast, friendly, and slick looking. By comparison, Centura's reporting tool is more like a hand-me-down Yugo—outdated, slow, sparse on features, and not known for its reliability (but free!). I am not out to bash Centura, but just about the biggest change to ReportWindows in the last three years was its name change to Report Builder. (To be fair, the QuickReports feature has been added more recently, but it enhances integration, so it will be covered in the second article.)

Let's face it, there are only two reasons to use Centura's reporting tool when products such as Crystal are available: its price (free) and its tight integration with Centura's development products. This second reason is very important to users of Centura's development tools; therefore, I'll cover it thoroughly in the second article.

Go Professional

Crystal comes in two versions: the *Professional* edition which lists for $395 (upgrades $199) and the stripped-down *Standard* edition for $195 (upgrades $79). The features discussed in this review are based on the Professional edition. Besides a few add-ons, the main features missing from the Standard edition are native SQL support (ODBC access only to desktop data sources such as Access, Excel, and FoxPro) and some programmability (about half the number of API functions are exposed). Therefore, when using a client/server development environment, such as Centura's, the Professional edition is the only way to go.

Weighing in at over 87 tons, er, megabytes…

Crystal comes in both 16-bit and 32-bit versions, each of which installs easily from the CD. When fully installed, the 32-bit Professional edition consumes a whopping 87M. The recommended RAM is 8M, but 16M is more workable. It runs on Windows 3.1, Windows 95, and Windows NT 3.51 or higher.

Luckily, Seagate provides the necessary assistance for using its monster of an application. The CD includes a computer-based training course sampler, a user manual of more than 700 pages, and extensive on-line help. The manual is impressive, not only in its size and

Crystal Reports Professional 5.0

Suggested retail, $395; upgrade, $199

**Seagate Software, Inc.
Information Management Group
1095 West Pender Street
Vancouver BC Canada V6E 2M6
(800) 877-2340, (604) 681-3435
www.img.seagatesoftware.com**

detail, but also in its ability to anticipate common questions and issues. For example, I was interested in reproducing the greenbar paper effect that I often use with my Centura reports. Sure enough, the manual had a topic entitled "How to alternate background colors for rows." (See my tip if you're interested in how it's done.) Other nice touches with the user manual are much appreciated. For example, it includes an extensive "What's New" section and a clever section—"Suggested Learning Paths"—for different types of users. Moreover, Crystal's wizard-like "Experts" make the typical complex tasks easy.

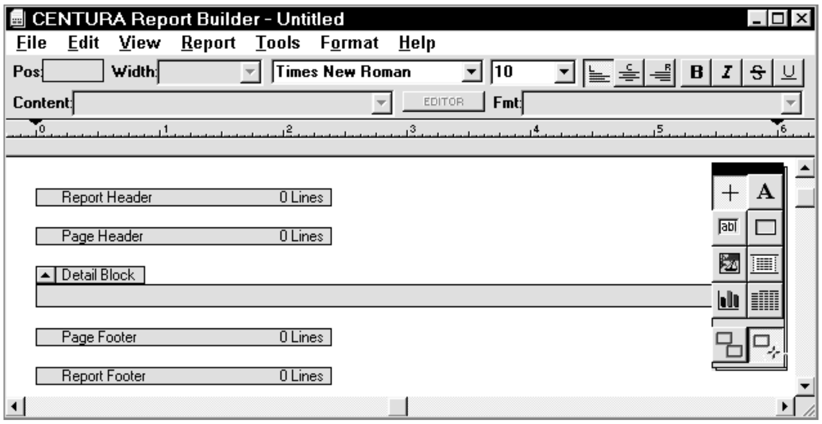


Figure 1. Centura's Report Builder interface.

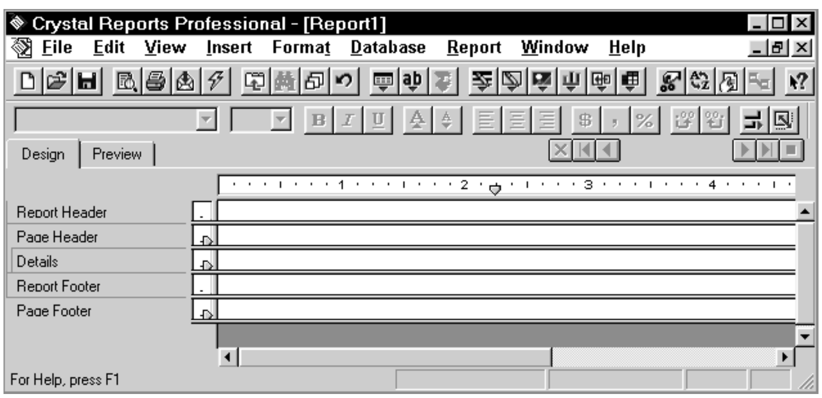


Figure 2. Seagate's Crystal Reports Pro 5.0 interface

You name it, Crystal Reports connects to it.

One of Crystal's many strengths is its ability to connect to virtually any data source. Besides the typical data sources, Crystal can connect to Arbor's Essbase (OLAP), Symantec's Act!, Computer Associates' Clipper, Microsoft's Exchange, Excel, NT event logs, ASCII files, Lotus Notes 3.x and 4.x, and a variety of Web server activity logs. In addition to the plethora of ODBC devices, the latest version of Crystal comes with a collection of native SQL routers for faster SQL access. Native router support is provided for Centura's SQLBase, Oracle, and Sybase databases.

Centura's reporting tool's data source has to be either a Centura application or a CSV file. Since Centura's development products can connect to most of the above data sources, indirectly, Centura's reporting tool can, too. However, this architecture, which relies on DDE to transfer the data from the Centura application to the report engine, adds another layer of overhead that hits performance. I have long suspected that this DDE interface has been Centura's Achilles' heel when it comes to robustness. Centura's reporting tool is adequate for short, simple reports; but I have often encountered problems when I have used it for larger batch reports. Other users of Crystal's previous versions have reported stability problems, so Crystal is not necessarily immune to this problem.

Features, features, everywhere

The sheer number of features that Crystal 5.0 has is almost overwhelming. The excellent documentation, on-line help, and Experts go a long way to prevent a novice user from becoming intimidated. Here are some of the more notable features.

A friendly (inter)face

If you use Centura's reporting tool, you'll feel at home with Crystal's user interface, which is similar. Both have report templates divided into report bands or sections (see **Figures 1** and **2**). These bands can be expanded, contracted, hidden, or duplicated to customize a report template. A tool bar and menus provide access to the various features. In Crystal, right-clicking invokes context menus for quick and easy access to key functionality.

Once I became familiar with Crystal's interface, it was no contest in terms of ease-of-use. However, not everything is intuitive, and it does have its limitations. One particular annoyance was the inability to undo after the use of certain Experts. While Crystal does warn you that this is the case, if you ignore the warning and don't save, you learn the hard way that Experts aren't always right.

Another annoying limitation is the inability to drag and drop a group of selected items between report bands. Fortunately, Crystal allows you to move one item at a time, a step above Centura's reporting tools, which force you to cut and paste between bands. It's also possible to open multiple report design windows at once and drag items from one report to another.

A Query tool allows you to take a graphical approach to developing the underlying SQL queries for reports. You may also enter SQL queries directly if you prefer.

Many Experts ready to assist you

Crystal provides over a dozen Experts, or wizards, to guide you with such tasks as creating a variety of reports, preparing graphs, setting up your overall report format, linking related databases, selecting records for inclusion in your report, and gathering the necessary files for report distribution. Centura's reporting tool at least provides a Wizard for cross-tab reports. This is necessary because cross-tab reports are one of the most complex routine reporting tasks.

Reporting types for all occasions

One of Crystal's most useful features is its tab-based wizard, Create Report Expert. This Expert leads you through step-by-step instructions for creating a number of different types of reports: standard, form letter, form, cross-tab, subreport, mail labels, top N, and drill-down. These reports can contain multiple detail sections, conditional sections, and columns. With version 5.0, reports can also be nested within each other. The Expert's interface is user-friendly and easily replaces Centura's quick report-generation capability, while providing a great deal more flexibility and power. For example, this report creation Expert can be re-entered to accommodate future changes.

Report distribution in many shapes and sizes

Seagate Software takes report distribution to new heights. In addition to the standard ability to print to a printer or a Preview window, Crystal can export a report file through e-mail, as a fax, directly to disk, to a Microsoft Exchange folder, to a Lotus Notes database, or to an ODBC data source. The report can be exported in any of several formats, including word processing, spreadsheet, database file, HTML, or standard data exchange (such as CSV, DIF, and RTF).

Crystal allows you to distribute a report to a user's desktop, complete with a stand-alone snapshot of data. Using the Report Distribution Expert, you can package a report with a current snapshot of data and any required components for stand-alone viewing/printing. A royalty-free runtime license is included.

Centura Tip!

How to Create a Greenbar-paper Effect

One effect I created extensively with Centura's reporting tool was to alternate the background color or shade of detail lines in order to improve readability. This effect mimics the green-barred paper used by antiquated line printers.

Producing this effect is simple using Centura's reporting tool. After selecting the Detail band, choose the Line menu item under the Format menu. Click on the Color button from the Format Line dialog box to access the Format Line Colors dialog box. In the background group, choose a color, such as green, and click on the Alternate Color check box. Now the detail lines will alternate between the specified background color and no color. Since I often print on a black and white printer, I choose a custom light gray shade as my alternate color. **Hint:** make the background color a light shade. Otherwise, when the report is copied or faxed, the shaded detail lines can become illegible.

Using Crystal, the process to produce the effect is not as simple nor intuitive. Fortunately, it's nicely documented within the user manual for version 5.0 on page 179, so I won't detail the exact steps here. The method takes advantage of the conditional formatting feature to conditionally suppress the background color formatting for alternating rows. On the downside, this method requires that you duplicate the entire detail band. As a result, this means that enhancements and maintenance of the report becomes more of a hassle.—*D.M.*

It all depends—conditional formatting

One of Crystal's key features is its flexibility provided through its conditional formatting. Using its If-Then-Else formula, you can allow a report's content to change depending upon its source data. While a programmer will be comfortable with Seagate's powerful Formula Editor used to control the conditional logic and more, it is not as intuitive nor user-friendly as the rest of the tool. On the positive side, users of Centura's reporting tool will be at home because Centura's Formula Editor has the same look and feel as Crystal's.

What Crystal Reports is not

It's easy to get overwhelmed by all of Crystal's features. However, Crystal does have its limitations. Although it's obviously a powerful client-side reporting tool, in trying to be all things to all people, Crystal falls short when compared to its competition once it steps out of its bailiwick.

Client/server reporting tool for large batch jobs

The limitations of client-side reporting tools become apparent when the client and server reside on physically different machines, and large batch reports are a requirement. In this case, any client-side reporting tool is going to fall short in terms of performance when compared to a server-based reporting tool such as MITI's SQR Workbench. Of course, Centura's reporting tool is also client-side and, therefore, suffers similar performance problems.

Note that Seagate has already recognized that Crystal can't be all things to all people, and has introduced another product called Crystal Info. This three-tier client/server reporting system allows users to schedule reports for immediate or future generation. It uses Crystal Reports as the design tool and features a robust server-based engine with good scalability. Check out their Web page for more information on this new product (www.img.seagatesoftware.com/cinfo/default.htm).

Ad-hoc query and reporting tool

Crystal isn't the best choice for an ad hoc query and reporting tool for end users. It can be simplified for users by creating an optional "Crystal Dictionary" metalayer. The dictionary metalayer provides a filter that simplifies and clarifies complex data forusers. In other words, the dictionary makes it easier for users to create their own reports. There are many other advantages to using the Crystal Dictionary feature, so if providing on-the-fly query and reporting capability is a requirement, that's added incentive to investigate this useful feature.

While its Dictionary metalayer feature makes Crystal more user-friendly for users, there are better tools on the market that provide greater functionality and ease of use, such as IQ Software Corp.'s IQ Objects. Centura's answer to end-user query tools is its aging Quest product, which can't compete with these newer products on the market.

Web publishing tool

Finally, I wouldn't recommend Crystal as a Web publishing tool. While it does allow for reports to be exported to a static HTML file, it doesn't allow any capability for publishing dynamic HTML database reports, such as Oracle's Web Request Broker or Netscape's LiveWire product.

Since Centura's reporting tool doesn't support HTML output at all, I guess something is better than nothing. Using Centura's tight integration between its development and reporting tool, it is possible to create HTML output with additional coding using QuickReports API, but this and other possibilities will be explored in my next article.

But, wait, there's more!

When all said and done, it's easy to see why Crystal is such a popular client-side reporting tool. From a stand-alone reporting tool perspective, Centura's reporting tool doesn't provide much competition. However, the critical question that remains from a Centura programmer's point of view is this: To what degree can Crystal be integrated into the Centura development environment? Obviously, Centura's own reporting tool is going to have better integration out of the box, but how much better? Stay tuned! **CP**

Doug Mitchell is a Principal with American Management Systems (AMS) in Fairfax, Virginia. He has been using Centura products to help his clients develop custom client/server business applications for the last six years. He volunteers as a Centura Team Assist member, serves as a Centura instructor, presides over the Washington, D.C.-area Centura user group and has presented at the last four Centura conferences. Contact him at doug_mitchell@mail.amsinc.com.

Centura News

Drawing Library Launched

TriLogic+ Automation BV, in Amsterdam, recently introduced its fourth utility for Centura developers. TriLogic Shape Composer is a figure library that allows data to be presented as graphical objects in a diagram or drawing. The advantage is that developers can use it to provide end users with real-world metaphors representing their data and the logic of their programs. Shape Composer includes a 16-bit SQLWindows version and a 32-bit Centura, shipped on the same disk. The company also publishes Class Explorer for SQLWindows, a class browser; Memorize, an in-memory database for optimizing application performance; and the Widget Library for SQLWindows, a library of user interface QuickObjects. A developer version of Shape Composer is priced at $495 during its introduction. To learn more, contact the company at +31-20-612-3344 or fax +31-20-685-5741. The company expects its Web site at www.trilogic.nl to be active in February. **CP**

If your company has news about new products, services, clients, or projects, send details to 71460.3142@compuserve.com, or fax (818) 246-0487.