CenturaPro™

Visit us at www.ProPublishing.com! Hot Ideas for Centura® Developers

The Secret's Out: Centura *is* Dynamic

Gianluca Pivato

June 1999

Volume 4, Number 6

Dynamic objects instantiation with object pointers is one of the most overdue improvements we could ever ask for in SQLWindows. It has been years now that developers have been asking for this feature—since the days of SQLWindows 4. Now it's possible—and it actually always has been!

For years developers have had to find creative ways to overcome a particular limitation in SQLWindows' implementation of object-oriented programming. We use window handles as object pointers, but it takes too many resources—plus Windows can't handle too many windows. We use "callback" functions to pass an object up to a base class as a parameter, but the code becomes less flexible. We also use global predefined arrays of objects with the index in the array as a pointer, but it has other obvious limitations and problems. I've always thought that if I only had dynamic object instantiation and pointers, my OOP code would be great!

SQLWindows

16 / 32

Wouldn't it be nice to be able to write something like this?

```
Object Handle: hObject
CMyClass: Test

!Create the object
Set hObject = SalUdvCreate( CMyClass )
!Using late bound
Call hObject.CMyClass.fMethod()
Set hObject.CMyClass.m_sStringMember = "Ciao"
! Using early bound
Set Test = hObject.CMyClass
Call Test.fMethod()
Set Test.m_sStringMember = "Ciao"
!Free the object
Call SalUdvFree( hObject )
```

Too good to be true, right? In fact, it can't be done. But we can get pretty close:

```
Window Handle: hObject
CMyClass: Test

!Create the object
Set hObject = XSalUdvCreate( CMyClass )
! Using early bound (late bound isn't possible)
Call XsalUdvLock( Test, hObject )
Call Test.fMethod()
Set Test.m_sStringMember = "Ciao"
Call XsalUdvRelease( Test, hObject )
!Free the object
Call XSalUdvFree( hObject )
```

Actually, you can also do much more than this. You can create an object using the class name as a string; and

Continues on page 3

Readers Speak

Mark Hunter

The results of the April *Centura Pro* survey are in. We got a response rate of about 10 percent, pretty good for these kinds of things. And what do you have to say?

The average length of time that readers have been using Centura tools or databases is 5.8 years. That's a lifetime in the world of I.T.! Let's see, how many different management teams is that?

What specific applications have you developed? If we posted all the responses, this editorial would be over. That's what we get for making it a fill-in-the-blank question instead of multiple-choice. How about just the big ones? Sales/marketing scored the highest, followed by inventory and materials management, then accounting, finance, order entry, ERP, healthcare, and human resources. Of course, there were a lot of individual answers that would fold into these, such as budget planning, pharmaceutical marketing, etc. It was interesting to see the variety of environments running CTD applications, from nuclear power plants to banks to environmental consultants.

Then you scored various reasons that you choose Centura tools. Important reasons included ease of use, customer satisfaction, and cost of development/return on investment. Least important were cost of acquisition, security, and out-of-the-box readiness.

Half of you are interested in joint marketing your apps with Centura. About 55 percent are interested in being featured in a Centura case study. You have subscribed to *Centura Pro* for an average of 2.4 years, and 78 percent of you have recommended it to friends and colleagues—thank you!

You visit the Centura Software Web site often: 60 percent of you are there every week or more frequently The *Centura Pro* Web site only draws fifteen percent of you that frequently. But things are happening fast lately, so consider checking in more often.

Who at Centura Software would you like to see as an author or interview subject in *Centura Pro*? Joe Falcone wins this one. Close behind are David Burke, Scott Broomfield, and Charity Silkebakken. Sometimes the choice was generic, such as "director of tools development" and "CTD product manager" (that's Charity). But I think my favorite response was: "Those responsible for the ReportBuilder!" Just sounds kind of ominous when you say it that way.

Forty-five percent of you are tackling distributed

Continues on page 9

Publisher Mark Kenobi Hunter, **Editor** Dian Skywalker Schaffhauser, **Business Manager** Shelley R2D2W2IRS Doyle, **Production Editor** Paul "Use the Force-Fit" Gould, **Centura Dog** Mocha Chewbarka

Centura Pro (ISSN: 1093-2100) is published monthly (12 times per year) by Pro Publishing, PO Box 2399, Nevada City, CA 95959.

POSTMASTER: Send address changes to *Centura Pro*, PO Box 2399, Nevada City, CA 95959.

Subscription information: To order, call Pro Publishing at 530-265-4082. 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.

Call Centura Software Corp. at 650-596-3400.

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.

Rebel Base

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

Editorial Department
Phone: 818-249-1364
Fax: 818-246-0487
E-mail: mhunter@sprintmail.com

Subscription Services
Phone: 530-265-4082
Fax: 530-265-0368
E-mail: shelley@propublishing.com

Mail
Pro Publishing
PO Box 2399
Nevada City, CA 95959

Centura *is* Dynamic . . .

Continued from page 1

you can test an object handle to see if it's directly created from a class or if the class appears anywhere in the inheritance tree. You can reference an existing object (object pointers) and even use the keyword "this" (or "self" for Delphi programmers) to get a pointer to the current object. How many times have you needed to pass the current object as a parameter? Now it's easy!

How does it work?

I've often pondered in my spare time about what it would take to add dynamic object instantiation and object pointers to SQLWindows. After all, I thought, the basic code to create an object from a class definition at runtime is already there. It happens each time a locally declared object has to be allocated and de-allocated and when an array of objects gets truncated or extended. It also happens when you create a window from a visual class derived from a functional class.

Using "SalArrayCreateUdv, SalArrayDestroy" in CDLLI11.DLL (or CDLLI15.DLL for CTD 1.5), you can actually create an array from a UDV class at runtime:

```
hArray = SalArrayCreateUdv( CClassName )
```

The declaration in the external functions is as follows:

```
Function: SalArrayCreateUdv
  Description:
  Export Ordinal: 0
  Returns
    Window Handle: HWND
  Parameters
    Template: TEMPLATE
```

So, now I can create an empty array from a UDV class at runtime. I can extend it and truncate it using SalArraySetUpperBound( )—all SalArray* functions accept a window handle as the array parameter—but I can't really use it unless I pass it to an external DLL . Even doing so, I would only be able to access the object's attributes. It's not much use for dynamic instantiation.

The solution is to use templates, which is similar to C++ casting. I used a check-out/check-in mechanism to switch the dynamic object's attributes with a "known" object in the outline acting as a template or casting variable. This is why I need the XsalUdvLock( )/ XsalUdvUnlock( ) pairs.

When you use XSalUdvLock( ), you "check out" the dynamic object into the template object, which is recognized by the SQLWindows engine, since it already existed at compile time. See the similarity with C++:

C++	**XSAL**
CMyClass* T;	Window Handle: H CMyClass: T
T = new CMyClass;	Set H = XSalUdvCreate(CMyClass) Call XSalUdvLock(T, H)
T->Method();	Call T.Method()
T->Property = 0;	Set T.Property = 0
Function(T);	Call Function(T)
delete T;	Call XSalUdvUnlock(T, H) Call XSalUdvFree(H)

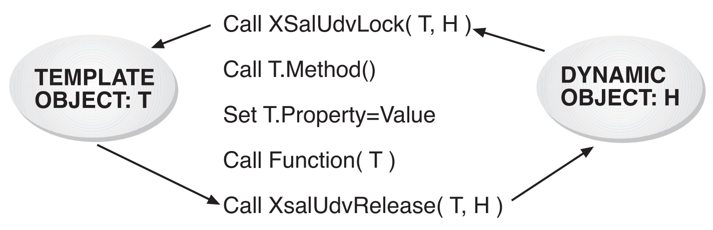


Figure 1. Lock/Unlock pairs.

The "template method" allows me to use an object not present in the outline at compile time during runtime. But what happens in XSalUdvCreate? The creation of the dynamic object can actually be accomplished in three ways, and I tried all three of them before finding the right one.

Creating dynamic objects

My first approach was to use SalArrayCreateUdv( ) to create a UDV array with only one item and use the array handle as the object handle. In the Lock/Unlock calls I could use the SWinArrayUdvAddress( ) call to get the memory address of the object to move its memory block to and from the template object.

This works, but it has some limits. After a certain number of arrays CTD crashes (I went up to 30,000), and there's no way to distinguish between an object pointer and a dynamic object.

My second approach was to use the array as a Class Factory (COM style). I created an internal list of dynamically created arrays for each class type, with only one item. One class, one array, and no more. Whenever I needed to create a new object, I allocated a block of memory for the object (the size is retrievable using SalOutlineClassSize( )), moved the new item memory block into the newly allocated memory block. To delete a dynamic object, I moved the object back into the array, truncated the item to free all the associate memory (strings and other arrays), and then re-extended the array of one item to be ready for the next new object request.

This approach allowed me to use my own structure for the object memory allocation in order to store extra information, such as the class handle (HITEM) and the type of handle. This way I could use the same handle for dynamic objects and object pointers. The object pointer is only some kind of "proxy" to an existing UDV. But this approach had a major flaw!

When SQLWindows creates a new object from a class, it initializes all its members to their default initial value. This is very important for numbers and dates. String handles are set to null and array members are created. And here's the problem. The new item in the dynamically created array (the class factory) always had the same array handles, which means that different instances created from the same class were sharing the same array members! It's a very bad case of data corruption.

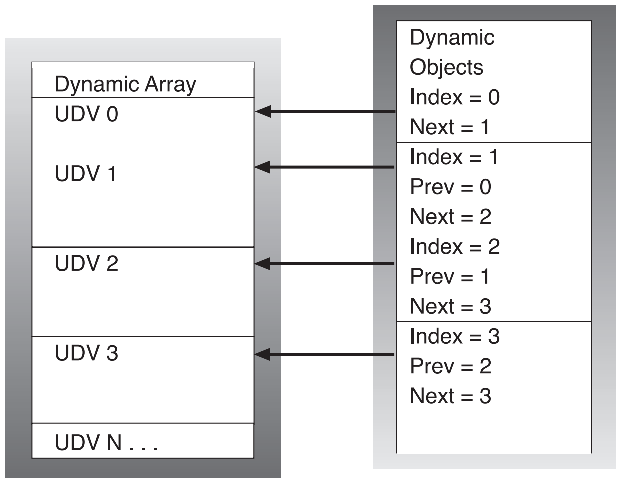


Figure 2. The memory structure.

I thought of creating the new item only when requested by XSalUdvCreate( ), resetting all its values and destroying it right after so that each new object would receive unique array handles. But there would be a problem in the destruction of the objects. To destroy the dynamic object, I need to move it back into the dynamically-created array. But the array doesn't have any item anymore, and if I extended the array by one again, a new set of embedded array handles would be created, and I would have memory leaks.

The final solution

The third and final approach, which worked, is actually a mix of the two previous approaches.

When XSalUdvCreate( ) is called, I look up the class HITEM in a table of existing dynamically created arrays. If the class was never used before, I create a new array using SalArrayCreateUdv( ) and add it to the memory table. The second step is to extend the array by one to create the object. The third step is to allocate a small memory block in which I store the handle type, the index in the array, the memory address of the UDV for object pointers, and other useful information. The last part consists of adding this memory block to a "daisy chain" (or linked list) of objects. The object pointer—what you declare as "Window Handle" in the outline—is the memory address of this "proxy" structure.

The fun comes with the destruction of the object. In memory we have two structures: the dynamically created array, which holds each instance of our dynamic objects; and the linked list of dynamically created objects. In **Figure 2** you can see a graphic representation.

When a dynamic object needs to be freed, I have to eliminate the corresponding item in the array. The only way to do that is to truncate the last item in the array using SalArraySetUpperBound( ); but the object that has to be released might not always be the last one created. My solution was to swap the last item in the array with the item to delete and then truncate the array.

To swap two items in a UDV array, I need only get their memory address using SWinArrayUdvAddress( ) and physically move the memory block using memmove( ). Then I have to update the object that was previously allocated to the last item in the array. Looking at Figure 2, if I need to delete UDV0, and I swap UDV3 (the last one), the dynamic object with Index = 3 would become unusable. The best approach is to also swap the two dynamic objects in the linked list; being a linked list, it's easy and fast.

The result is that the order in the linked list always reflects the order in the dynamically-created array.

Creating object pointers

An object pointer is the same as a dynamic object without the corresponding item in the dynamically created array. It's like an "empty shell" or "proxy." Object pointers are kept in a separate linked list to avoid interfering with the tight correspondence between the dynamic objects list and the array.

I have two functions for object pointers:

```
Window Handle: hObject
Set hObject = XSalUdvGetObject( OBJ )
Set hObject = XSalUdvGetObjectThis()
```

The first one creates an object pointer out of an existing object; the second creates the pointer out of the currently executing object.

The object pointer contains a pointer to the memory address of the UDV it's referring to. The memory address of a UDV is in the first four bytes (LONG) of the HUDV structure. To retrieve the HUDV of the currently executing UDV, I simply use SalUdvGetCurrentHandle( ).

Object pointers need to be freed using XSalUdvFree(

hObject). In case you forget, XSal2 will free everything you have allocated during runtime when the application stops. Plus, there a couple of diagnostic functions that allow you detect the number of currently allocated objects.

Using dynamic objects and object pointers

Now you know how the underlying memory management works. But the usage is a different story.

As I mentioned earlier, the only solution I could find was to use the "template method." This requires you to "check-out" the dynamic object into a known template and to check the object back in when you're done. An object can be checked out only once. After it has been checked out, you should only use the template to which the dynamic object was assigned.

The "check-out" procedure is composed of two steps: First, save the memory block of the template object into a temporary buffer associated with the dynamic object, and then copy the dynamic object into the template buffer. This way the runtime engine is working as usual with the known object; but in actuality it's working on a different set of values. The "check-in" procedure does exactly the opposite, ensuring that whatever has changed in the template goes back into the dynamic object, and that the template memory block gets restored to its original values. This way, when runtime frees the template (for example, exiting from a local code block), it doesn't destroy the content of the dynamic object. The persistence is assured.

The same technique works for object pointers; instead of copying the memory block out of the dynamically-created array, I copy it straight from the object. See what is possible now that was impossible before:

```
Functional Class: CParent
  Instance Variables
    String: m_sName
    ChildObject: CChild
  Functions
    Function: Test
      Actions
        Call ChildObject.SetParentTest(
          XSalUdvGetObjectThis( CParent ), "Luca" )
        Call SalMessageBox( m_sName, "", 0 )
Functional Class: CChild
  Functions
    Function: SetParentTest
      Parameters
        Window Handle: p_hObjectParent
        String: p_sValue
      Local variables
        Parent: CParent
      Actions
        Call XSalUdvLock( Parent, p_hObjectParent )
        Set Parent.m_sName = p_sValue
        Call XSalUdvRelease( Parent, p_hObjectParent )
        Call XsalUdvFree( p_hObjectParent )
```

A child object is able to know about its parent object.

More useful functions

You can use other very useful functions to achieve much better OOP than has ever been possible with SQLWindows:

```
XSalUdvCreateFromString( "ClassName" )
XSalUdvIsObjectOfType( hObject, ClassName )
XSalUdvIsDerivedFrom( hObject, ClassName )
```

Using XSalUdvCreateFromString( ), you can pass the class name as a string parameter, which means you can actually decide the class to use at runtime. The object handle datatype is always a window handle anyway.

XSalUdvIsObjectOfType( ) allows you to detect the class that was used to create the object. This function is of great help when you need to test the handle before casting it into a template (check-out):

```
Set hObject = XsalUdvCreateFromString( sClassName )
If XSalUdvIsObjectOfType( hObject, CClass1 )
  Call XSalUdvLock( T1, hObject )
  Call T1.Method1()
  Call XsalUdvRelease( T1, hObject )
Else If XSalUdvIsObjectOfType( hObject, CClass2 )
  Call XSalUdvLock( T2, hObject )
  Call T1.Method2()
  Call XsalUdvRelease( T2, hObject )
```

XSalUdvIsDerivedFrom( ) returns TRUE if the class is anywhere in the dynamic object's inheritance tree. This is useful if you need to pass the object as a parameter to a function that has a parameter declaration of a base class.

A sample application

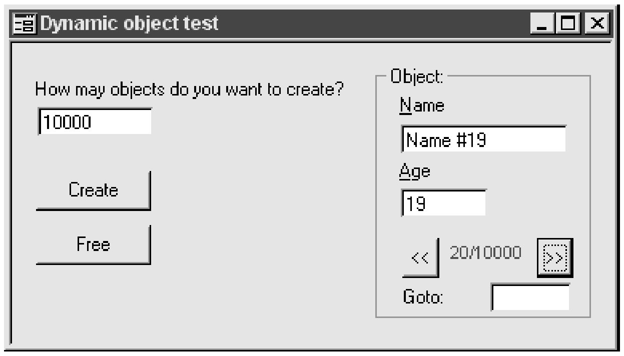
The sample code I've included is simple. It was intended to show the reliability and speed of dynamic objects.

The sample application creates a variable number of objects and initializes them with a string and number. Using the arrow buttons you can browse the collection of objects. You can change the value of each individual object by just typing in the datafields.

This sample application could have been written using dynamic arrays, but looking at my source code, you'll notice that the object collection isn't declared anywhere; it's created dynamically!

Can every dynamic object be a COM automation object?

One of the potential, relatively easy enhancements of the technique I've illustrated is to be able to attach any kind of code to the dynamic object structure, the one that holds



How may objects do you want to create? 10000

Name: Name #19

Age: 19

20/10000

Goto:

Figure 3. The sample application.

the index to the dynamically created array or the pointer to the existing object.

It's possible to implement the IDispatch interface and make the object immediately become an automation object. The IDispatch has to do two basic things: return an ID from a member name and dispatch requests for the member.

GetIdsOfNames

Since we can't read the definition of the class at runtime, we build one at runtime. Each time a client requests a name, we look it up in a table; if it exists, we return the ID; if it doesn't, we simply add it.

Invoke

This is a little more difficult. This function is used to read and write the properties and to execute the methods exposed from the automation object. To read and write the properties, we can simply use SalOutlineVarOffset( ) to retrieve the offset of the instance variable from the memory address of the UDV.

To execute a function call is a little more difficult. The technique is to build the call using the parameters list and "fool" SalCompileAndEvaluate( ) by manually changing the context string. Remember that we're talking about dynamically-created objects, which means that we don't have the object name in the outline at compile time. So we can't simply use "Call OBJ.Method(p1, p2)". We only have the function name, not the object name.

The context string (the last parameter in SalCompileAndEvaluate( )) holds, among many things, the memory address of the currently executing UDV and the class HITEM. We have both! The solution is to retrieve the context string and to change those two LONG numbers in there. After that SalCompileAndEvaluate( ) will think that it's running in the object scope.

What to do with the technique

The first use that comes to mind is the Microsoft Script Control. It's a COM class that runs on COM clients, implements VBScript and Jscript, and is freeware. You can use VBScript/Jscript to script your CTD applications, but you must register the objects that the script recognizes. One way to register them is to use the newly released XSalCOM, but that means that *all* your objects become published COM objects—they go into the Windows registry.

Using dynamically created objects, you can simply pass the object IDispatch* pointer (at this point it's the same as the object handle) to the Script controls and you're done. I'll write a more detailed example of this very useful feature for a future issue of *Centura Pro*.

How good is it?

I know that this new OOP enhancement, dynamic objects, is not the same as C++, Visual Basic, or Delphi. And I know that with access to the SQLWindows/32 engine source code, it would be possible to do much better than this. However, this is a pretty good implementation that enables a lot of previously impossible things—important things! Many of the sophisticated OOP features of a great 3GL like Java or C++ are now available in the world's best 4GL, SQLWindows. They always were "available"—they were just waiting for someone to expose them. **CP**

Download DynObj.ZIP from this issue's Table of Contents at www.ProPublishing.com or find it on this month's Companion Disk.

Gianluca Pivato designs LANs, WANs, databases, software, and anything that interacts with the system. Developer of XSal and XSal2, he recently developed XSalCOM, a C++ application that turns Centura applications into Automation Servers. His company is Pivato Consulting, inc., and he can be reached at gianluca@pivato.com.

XOR for SAL

Centura Tip!

Gerd Marinitsch—SAL provides the common AND, OR, and NOT Boolean operators and also provides bitwise AND (& - ampersand) and bitwise OR (| - vertical bar) operators. These can be used as the building blocks to implement an XOR (eXclusive OR) operator for SAL. The XOR operator produces results as shown in the following table:

Truth table for XOR operation

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

A Boolean XOR operator can be implemented with the following code:

```
◇ Set bResult = ( bA OR bB ) AND NOT ( bA AND bB )
```

A bitwise XOR operator is a little more complicated since SAL doesn't provide a bitwise NOT operator; but the following code shows how to do it:

SQLWindows
16 / 32

```
◇ Set nResult = ( nA | nB ) & ( -( nA & nB ) - 1 )
```

Gerd Marinitsch is Sales Engineer with Centura Software in Germany. Reach him at gerd.marinitsch@centurasoft.com.

Maximize your Documentation

David Brito

PRODUCT Review

One of the most neglected steps of programming is documentation. Programmers fall into one of two categories: those who comment while they program, and those who wait until the program is complete. Regardless of which method you prefer, the results tend to be the same. Once the project deadline nears, comments get pushed aside and ultimately never get entered. That's where systems | objects | resources' DocSal comes in. This tool allows you to comment your programs. You're given the option of selecting which parts of the program will be imported into DocSal. So, for example, if time is short and function descriptions are a must, then you would only import the internal functions. See **Figure 1** for an example of how this is done.

An interesting advantage to using this tool is that you don't have to have a programmer fill in the comments. The interface, which looks a lot like Windows Explorer, is so intuitive that you can have anyone add the comments for you, as long as you provide the content. Of course, it's more advisable to have the developer write comments as code is created.

SQLWindows 32

So, what exactly is it?

DocSal allows you to comment your code, but that's just the beginning. Its true purpose is to export key portions of your program, with their comments, into an HTML format. This allows technical documentation to be stored on your intranet. This is a powerful advantage when you have libraries that are shared throughout the company. All someone needs to do is browse the documentation over the Web when looking for an existing APL to fit his or her needs. However, the one portion of your program that it can't import or export is the Actions section of a function. This is unfortunate, since that's usually where the majority of your logic resides. So regardless of whether or not you decide to use this tool, you will still have to do some commenting within CTD (that is, if you're disciplined enough to do so). *[A version of DocSal released after this review was written adds the ability to read comments from functions and the Actions section.—Ed.]*

A definite plus of using the tool is that it forces you to adopt a standard for commenting your code. We programmers tend to have our own preferences for adding comments. Some people add them above a line of code, others below, and others on the right side of the code. The tool places the comment below the line of code and indents it. As you all know, standards can ease the pain of reading through someone else's code.

Customizing the HTML

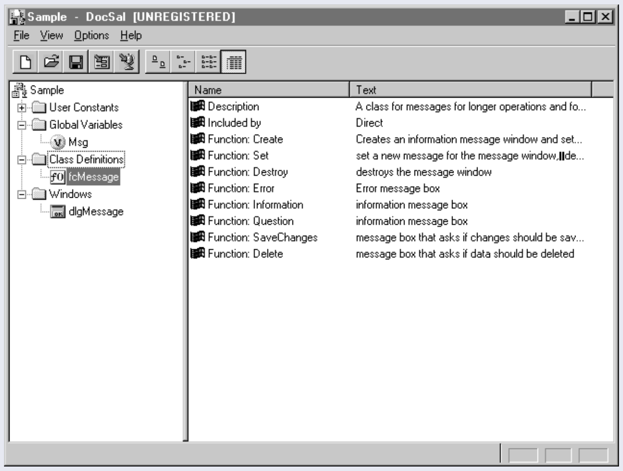
You can customize your page by choosing what font, color, and background color you would like to use. You can even elect to have frames or publish your page without frames just in case you have an outdated Web browser.

Continues on page 9

DocSal

systems | objects | resources
www.rimpi.de/sor
$249 US per desktop

You can download the demo at this site or find it on this month's *Centura Pro* Companion Disk.



Name	Text
Description	A class for messages for longer operations and fo...
Included by	Direct
Function: Create	Creates an information message window and set...
Function: Set	set a new message for the message window,‖de...
Function: Destroy	destroys the message window
Function: Error	Error message box
Function: Information	information message box
Function: Question	information message box
Function: SaveChanges	message box that asks if changes should be sav...
Function: Delete	message box that asks if data should be deleted

Figure 1. Choosing the outline elements you wish to work with.

Listing 1. A class definition from the DocSal sample application.

```
Functional Class: fcMessage
  Description: A class for messages for longer
    operations and for information and question dialogs
  Derived From
  Class Variables
    Window Handle: c_hWndMsg
    Boolean: c_bCancelMode
  Instance Variables
  Functions
    Function: Create
      Description: Creates an information message window and sets the
          message using the existing window if it was created before
      Returns
      Parameters
        String: p_sMsg
          ! The message to display
        Number: p_nMode
          ! the mode to use different icons, not used at this time
      Static Variables
      Local variables
      Actions
    Function: Set
      Description: set a new message for the message window,
          destroys the message window if cancel mode was
          active and the cancel button was clicked
          *Returns: Successful (TRUE) or not (FALSE)
      Returns
        Boolean:
      Parameters
        String: p_sMsg
          ! The message to display
      Static Variables
      Local variables
        Boolean: bRet
      Actions
    Function: Destroy
      Description: destroys the message window
      Returns
      Parameters
      Static Variables
      Local variables
      Actions
    Function: Error
      Description: Error message box
      Returns
      Parameters
        String: p_sMessageText
          ! The text to display
      Static Variables
      Local variables
      Actions
    Function: Information
      Description: information message box
      Returns
      Parameters
        String: p_sMessageText
          ! The text to display
      Static Variables
      Local variables
      Actions
    Function: Question
      Description: question message box
          *Returns: IDYES, IDNO or IDCANCEL
      Returns
        Number:
      Parameters
        String: p_sMessageText
          ! The text to display
      Static Variables
      Local variables
      Actions
    Function: SaveChanges
      Description: message box that asks if changes should be saved
          *Returns: IDYES, IDNO or IDCANCEL
      Returns
      Parameters
      Static Variables
      Local variables
      Actions
    Function: Delete
      Description: message box that asks if data should be deleted
          *Returns: IDYES, IDNO or IDCANCEL
      Returns
      Parameters
      Static Variables
      Local variables
      Actions
```

The Vendor Responds

Here we give S|O|R a chance to address this review . . .

There are some other important features of DocSal that should be mentioned. DocSal allows you to change and delete items. You can add comments, descriptions, and other information to the items: This means you can edit your comments in DocSal from the application file. And you can write the changes to your documentation back to your application source file; this work isn't lost!

You can also save the documentation in a separate file. In this way you can keep documentation and source code appropriately related (in other words, one file for version 1.0, another for version 1.1).

Some other helpful features include:

- Automatic index creation with links to existing documentation. As you can see in the samples, there is the "DocSal documentation center" where all documentation is indexed.
- Easy navigation in the created HTML; just choose your category, subcategory, item, and display all information you are looking for.
- An option for whether items from included libraries should be read or not.
- The choice to exclude libraries and external functions. You can decide which libraries and which DLLs shouldn't be read or documented

With regard to the pricing issue, remember this: Creating documentation with DocSal costs the time during development in which you add your comments (which you should do anyway), and the time DocSal needs to create the docs, which usually takes a few minutes.

Maximize Your Documentation

Continued from page 7

What about my existing applications?

DocSal comes with a tool called MoveComments. The purpose of this tool is to move comments that are on top or to the right side of a line of code to the format required by DocSal.

An example

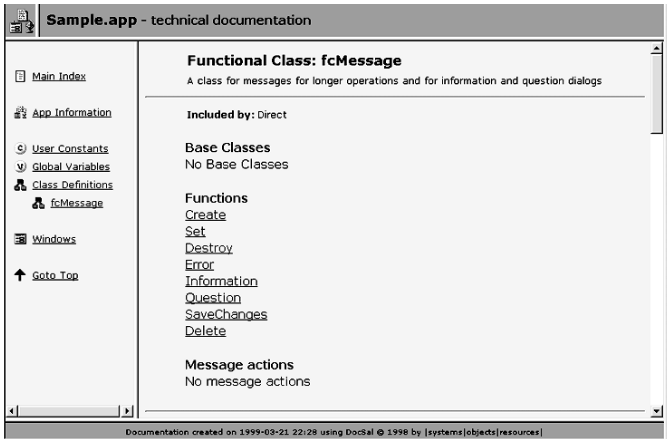
Look at **Listing 1** for an example of a class definition, and then look at **Figures 2** and **3** for the equivalent HTML. The tree-structured list in the frame on the left side of the browser window is an approximation of the SAL outline.

A matter of style

The value of DocSal depends partly on your current—and future—programming style. If you're careless about documentation already, DocSal won't help you much. If your documentation is well-organized, DocSal can be very helpful. And even if you're well-organized, there are some style issues. As noted above, comments within an action block are ignored by DocSal. This means that you should place increased emphasis on the "Description" section of functions, moving some of your comments there instead. If a function has parameters, put a description of each parameter inside the Parameters section itself, instead of explaining each parameter inside the action block, at the spot where it's used. These style issues would affect my company, and I'm sure they would affect other potential users.

The bottom line

The interface is easy to follow. The ability to browse an application by opening the documentation as HTML is very useful. The product reads all CTD file formats, including APT. I can't say that everyone would benefit from using this product, because of the style issues I've mentioned. At $249 US per desktop (to a maximum of $999 US per site) I find the price to be a little high. However, updates are free for the lifetime of the product. And it's worth looking into, especially if you have a large, well-organized development team. **CP**



Sample.app - technical documentation

Main Index
App Information
User Constants
Global Variables
Class Definitions
fcMessage
Windows
Goto Top

Functional Class: fcMessage
A class for messages for longer operations and for information and question dialogs

Included by: Direct

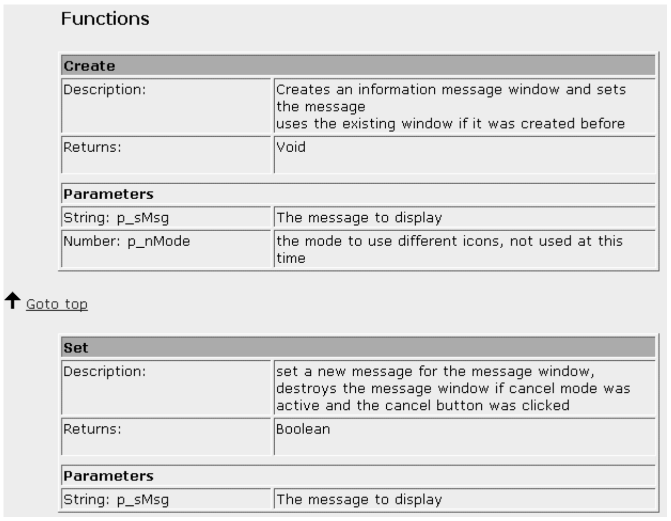
Base Classes
No Base Classes

Functions
Create
Set
Destroy
Error
Information
Question
SaveChanges
Delete

Message actions
No message actions

Documentation created on 1999-03-21 22:28 using DocSal © 1998 by |systems|objects|resources|

Figure 2. Part of the HTML generated from comments in fcMessage.



Functions

Create	
Description:	Creates an information message window and sets the message uses the existing window if it was created before
Returns:	Void
Parameters	
String: p_sMsg	The message to display
Number: p_nMode	the mode to use different icons, not used at this time

Goto top

Set	
Description:	set a new message for the message window, destroys the message window if cancel mode was active and the cancel button was clicked
Returns:	Boolean
Parameters	
String: p_sMsg	The message to display

Figure 3. HTML for a few of the functions in fcMessage.

David Brito is a senior application developer at Southern California Edison, a large utility company. He can be reached at britod@sce.com.

Readers Speak

Continued from page 2

applications within six months, 31 percent in 18 months, and 24 percent have no plans. Forty-five percent plan to use COM or DCOM, 26 percent will use Microsoft Transaction Server, 29 percent will use CORBA, and 17 percent will use Enterprise Java Beans.

And what about the contest? First place and a $50 amazon.com gift certificate goes to Lehi Clark. Victor Livshen and Claus Hanson each take away a $35 gift certificate. Congratulations—and thanks for your insights!

Pivato does it again

After turning the SQLWindows/32 upside down with XSalCOM last month, Gianluca Pivato turns it inside out this month with his treatise on dynamic object instantiation. It's extremely thought-provoking, especially the realization that this feature has been possible since 16-bit SQLWindows release 4.0. Read about it on our cover. **CP**

Being Resourceful

One undocumented SAL function is SalResLoad. This function copies (or loads) a resource (from the Resources section of Global Declarations) into a string variable. The syntax is:

```
bOk = SalResLoad( tResource, strResource )
```

where tResource is the template name of a resource in the application, and strResource is the string variable that will receive the resource data. A Boolean TRUE or FALSE is returned to indicate the success of the function.

Resources are bitmaps, icons, and cursors embedded within an application's .EXE file. One advantage of using resources is that you can reduce the number of files you have to deploy. Functions like SalPicSet and SalCursorSet can use the resources defined in an application.

What is the value in being able to copy a resource into a string? Well, one useful application is with reports. Suppose you're printing a credit report on customers. With each customer you want to print a bitmap that indicates good credit or bad credit. In your Report Builder .QRP file, you create an Input Item called CreditPicture with an Object data type, and assign it to a picture in the detail block.

In your application you'd have a couple of bitmap resources:

```
◆ Global Declarations
   .
   .
   .
  ◆ Resources
    ◆ Bitmap: bmpGoodCredit
      ◇ File Name: Good Credit.bmp
    ◆ Bitmap: bmpBadCredit
      ◇ File Name: Bad Credit.bmp
```

When you start the report in your application, using SalReportView or SalReportPrint, you match a string variable from your application with the CreditPicture object in the .QRP file:

```
◇ Call SalReportView( hWndForm, hWndNULL, sReport,
    "…, sCreditPicture, …",
    "…, CreditPicture, …", nFlags )
```

For this example we get data from a database, so the SAM_ReportFetchNext message handler will get the next row from a database query, where one of the columns describes the customer's credit rating. Use such a column to load the appropriate resource for the report:

SQLWindows 16 / 32

```
◆ On SAM_ReportFetchNext
  ◆ If SqlFetchNext( hSql, nFetchInd )
    ◆ If sCreditRating = "good"
      ◇ Call SalResLoad( bmpGoodCredit,
         sCreditPicture )
    ◆ Else If sCreditRating = "bad"
      ◇ Call SalResLoad( bmpBadCredit,
         sCreditPicture )
    ◇ Return TRUE
  ◆ Else
    ◇ Return FALSE
```

This way you can dynamically determine the picture image to send to the report and take advantage of their being implemented as resources in your application.

SAM User exposes undocumented interfaces and relentlessly pursues the undocumented in Centura's desktop development environments. Send comments and questions in care of the editor.

Just What are You?

Centura Tip!

R.J. David Burke—If you're maintaining both 16- and 32-bit versions of applications, it can be useful to determine, in SAL code, whether the application is a 16-bit application, developed with the original SQLWindows, or a 32-bit Centura Team Developer application built with SQLWindows/32.

There are a variety of ways to go about this, but one simple and reliable technique is the IsSQLWindows32 function, presented below.

```
◆ Function: IsSQLWindows32
  ◇ Description:
  ◆ Returns
    ◇ Boolean:
  ◇ Parameters
  ◇ Static Variables
  ◆ Local variables
    ◇ Number: nError
    ◇ Number: nErrorPos
    ◇ String: sReturn
    ◇ Number: nReturn
    ◇ Date/Time: dtReturn
    ◇ Window Handle: hReturn
  ◆ Actions
    ◆ If SalCompileAndEvaluate( 'PLATFORM_Current',
       nError, nErrorPos, nReturn, sReturn, dtReturn,
       hReturn, TRUE, SalContextCurrent(  ) )
       AND nError = 0
      ◇ Return TRUE
    ◇ Return FALSE
```

SQLWindows 16 / 32

The idea behind this technique is to use SalCompileAndEvaluate to evaluate a constant that is only present in Centura Team Developer, and not in the original SQLWindows. There are a few suitable constants; I went with PLATFORM_Current, which was originally introduced in CTD 1.0 as a means for an app to determine whether it was running under a Windows or Solaris environment.

In the IsSQLWindows32 function, if the call SalCompileAndEvaluate is successful, and there is no error, then the runtime must be 32-bit Centura Team Developer and TRUE is returned. Otherwise, the runtime is 16-bit SQLWindows, and FALSE is returned.

Real Cool Hotkeys

Thomas Wiedmann with Adrian Portugall

Here's how to include the functionality of Sidekick in Centura Team Developer—and offer any information with a single keystroke.

The use of Key Accelerators from Visual Toolchest or SQLWindows is only applicable to your active form. But the Win32 API easily facilitates the definition of a hotkey, which is valid for the whole desktop. **Figure 1** shows the starting form I use to set the hotkey.

Using the combo box and the three checkboxes, I can define a hotkey. The activation is effected with the Set button and the removal is carried out with the Destroy button. After the activation of the hotkey, the form is brought to the foreground, no matter what application had focus at the time of activation—even DOS boxes.

SQLWindows 32

The details

Message PAM_HotKey controls the display elements of the form window frmWinHotKey, shown in Figure 1. The mode Alt, Ctrl, or Shift, is defined by the following three constants, which can be combined:

```
Number: MOD_ALT = 1
Number: MOD_CONTROL = 2
Number: MOD_SHIFT = 4
```

I fill the combo box for the keycode with a loop to start. Plus, I convert the ASCII values from 65 (=A) to 91 (=Z) into CHAR before inserting them in the combo box.

For the definition of a hotkey, I need USER32.DLL and KERNEL32.DLL and the message WM_HOTKEY (0x0312)., as shown in **Figure 2**.

Listing 1. Class Function fCreateHotKey.

```
! First, Get free number
Set inAtom = GlobalAddAtomA('Centura')
!
If inAtom = 0
  Call SalMessageBox('GlobalAddAtomA','ERROR',MB_Ok)
  Return FALSE
!
! Second, register Hotkey
If RegisterHotKey(hWndForm, inAtom,
  p_nKey, p_nKeyCode)
  Return TRUE
Else
  Call SalMessageBox('RegisterHotKey','ERROR',MB_Ok)
  Return FALSE
```

If the return value is TRUE, the new hotkey is activated and can be tested.

Listing 2 shows how the hotkey is assigned. First, I need a unique value for the "atom." I may use any string to identify it in the Atom table—in my example I use "Centura." If the return value is valid, I register my hotkey with the API function RegisterHotKey. The parameters are the relevant window handle, inAtom, a value for Alt | Ctrl | Shift, and a keycode in the range "A" through "Z".

I use the Destroy button to remove the relevant hotkey again. In this case I need the functions UnregisterHotKey and DeleteAtom.

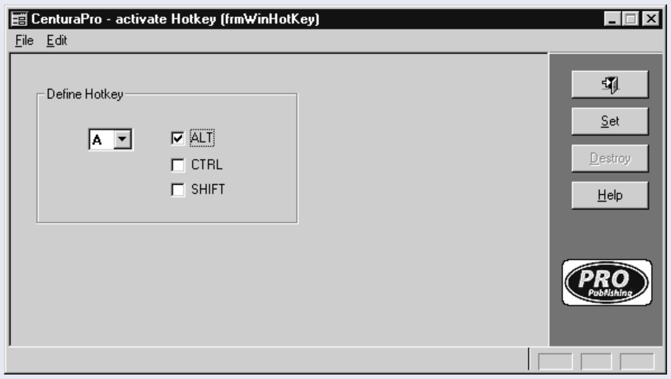
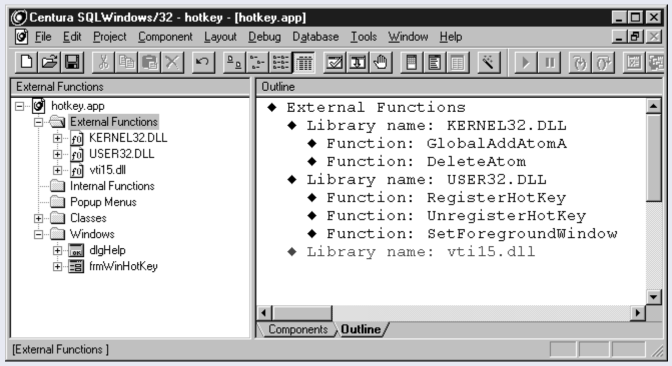


Figure 1. The starting form. "Define Hotkey" adjusts the hotkey.



```
◆ External Functions
  ◆ Library name: KERNEL32.DLL
    ◆ Function: GlobalAddAtomA
    ◆ Function: DeleteAtom
  ◆ Library name: USER32.DLL
    ◆ Function: RegisterHotKey
    ◆ Function: UnregisterHotKey
    ◆ Function: SetForegroundWindow
  ◆ Library name: vti15.dll
```

Figure 2. The necessary API functions.

Listing 2. Unregister the hotkey.

```
! First, Unregister HotKey
If NOT UnregisterHotKey(hWndForm,inAtom)
  Call SalMessageBox('UnregisterHotKey','ERROR',MB_Ok)
  Return FALSE
!
! Second, Delete Atom
Set inAtom = DeleteAtom(inAtom)
!
If inAtom = 0
  Return TRUE
Else
  Call SalMessageBox('DeleteAtom','ERROR',MB_Ok)
  Return FALSE
```

Now the hotkey is removed again and the unique number in the atom is released.

Action!

Which messages are set off? Very simple: as specified by the window handle parameter in Listing 2, the registered form window receives the message WM_HOTKEY.

Listing 3. WM_HOTKEY arrives at your form.

```
Message Actions
 On WM_HOTKEY
   Call VisWinShow(hWndForm,SHOW_Normal)
   Call SalBringWindowToTop(hWndForm)
   Call SetForegroundWindow(hWndForm)
   Call SalPostMsg(picPropub,PAM_HotKey,TRUE,0)
   Call SalSetFocus(pbHelp)
```

To activate and display my form window, I need the following code elements:

- VisWinShow( ) activates the form within the taskbar.
- SalBringWindowToTop( ) puts a window to the top of all overlapping SQLWindows forms.
- SetForegroundWindow( ) puts a window to the top of all overlapping desktop forms.
- SalSetFocus( ) is necessary if there are no enabled child window.

Centura Team Developer, with a little help from the Windows API, manages this requirement without any problems. **CP**

Download HotKey.zip from this issue's Table of Contents at www.ProPublishing.com or find it on this month's Companion Disk.

Thomas Wiedmann works as Developer and Database Manager at targetsoft gmbh in Weissach, Germany. He has been using Centura and IBM DB2 products for several years now. He can be reached at thomas.wiedmann@targetsoft.de.

Sneaky Sorting Dichotomy

Centura Tip!

SQLWindows

16 / 32

SQLBase

R.J. David Burke—When the Sorted attribute for list boxes and combo boxes is set to Yes, Windows will automatically sort the list items using a "telephone book" sort. With this kind of sort, apostrophes (') are ignored and the sort is case insensitive. For example, add the following names to a list box or combo box with Sorted set to Yes:

O'Brien
O'Malley
Oakes
Oldman

and they'll appear in this order:

Oakes
O'Brien
Oldman
O'Malley

Now consider putting this data in a SQL database, such as SQLBase. For example, execute the following SQL statements using SQLTalk:

```
CREATE TABLE PEOPLE
(
  LAST_NAME VARCHAR(20) NOT NULL
)
/

INSERT INTO PEOPLE (LAST_NAME) VALUES (:1)
\
Oakes
O'Brien
Oldman
O'Malley
/

SELECT LAST_NAME
  FROM PEOPLE
  ORDER BY LAST_NAME ASC
/
```

and the names are returned in an ASCII sorted order based on all characters including the apostrophe:

O'Brien
O'Malley
Oakes
Oldman

This sorting behavior was also confirmed in Oracle. Just another thing to be aware of when building applications. For consistency, consider doing all of your sorting on either the server or the client, but not both.