CenturaPro™
www.ProPublishing.com
Hot Ideas for Centura® Developers

Control the Calendar

Christian Astor

February 2000
Volume 5, Number 2

PRO Publishing

Here are samples of custom control classes, showing how to use two of the 22 Windows Common Controls: Date and Time Picker, and Month Calendar. You'll find heaps of useful declarations and examples in this article and the accompanying source code!

The Windows Common Control classes provide a lot of functionality to Centura applications, and it's not difficult to integrate them into your code. The class definitions in my sample application can be included directly, and the supporting external functions and constants should require only minor editing if your application is already defining some of them. Interacting with the controls at low level is done almost completely through messaging, using pointers to data structures. The data structures must be written and read using the CStruct family of functions. My class definitions encapsulate this tedious work in class functions, so your application interacts with the objects through ordinary function calls.

SQLWindows
32

To be able to handle Common Controls messages, I must redefine the function SendMessageTimeoutA( ), which allows the app to receive data in a LPVOID string pointed to by lParam after the message has returned. See an example in **Listing 1**, in which my cMonthCalendar class function sends a message to the month calendar control asking for the currently selected date, passing a string in the fourth parameter (lParam) to hold the answer, then extracts the numeric date components from the altered string using Cstruct functions.

The same technique is used to send data to the control. In **Listing 2**, I set the "today" date in the calendar. This normally defaults to the system date, but can be overridden using a low-level message, which has been encapsulated in a class function.

Listing 1. Retrieve data from the custom control using messaging.

```
◆ Function: GetCurDateSelected
  ◇ Description:
  ◆ Returns
    ◇ Date/Time:
  ◇ Parameters
  ◇ Static Variables
  ◆ Local variables
    ◇ String: sDate
    ◇ Number: nDateYear
    ◇ Number: nDateMonth
    ◇ Number: nDateDay
  ◆ Actions
    ◇ Call SalStrSetBufferLength( sDate, 16 )
    ◇ Call SendMessageTimeoutA( hWndCalendar,
      MCM_GETCURSEL, 0, sDate, 2,0, nReturn)
    ◇ Set nDateYear = CStructGetWord( sDate, 0 )
    ◇ Set nDateMonth = CStructGetWord( sDate, 2 )
    ◇ Set nDateDay = CStructGetWord( sDate, 6 )
    ◇ Return SalDateConstruct( nDateYear,
      nDateMonth, nDateDay, 0, 0, 0 )
```

Continues on page 3

And the Winner Is . . .

Mark Hunter

We recently finished counting the votes from the *Centura Pro* Mentor Awards. It was our intent to honor the most valuable contributors on the Centura Internet newsgroups, as chosen by the newsgroup participants themselves. *Centura Pro* is a great source of articles for explaining a topic at length, and the newsgroups are an active, speedy way of getting tips and answers to maddening questions (usually). No one pays much attention to the dedicated efforts of the newsgroup contributors, so Pro Publishing decided to provide some financial recognition.

The first place winner is Neil Rashbrook of Parkway Computer Consultants in the United Kingdom. Neil's voters were adamant that he is "the man." Some were particularly fond of his tool, Network-VTExplorer, which adds network exploration capability to the cDesktopListBox class from Visual tool chest. Neil gets a $100 gift certificate to amazon.com.

Second prize goes to Michael Vandine, the Australian SQLBase expert. The voters consistently praised the value of his posts. Michael responds, "I am very happy that I can help so many people with their questions. Thanks to everyone who voted for me! It really is nice to get some feedback showing that my efforts are appreciated!"

Indeed they are, Michael, and a $50 gift certificate to amazon.com is on its way to you.

Third prize goes to Christian Astor, a Frenchman active in the Centura Team Developer and Advanced Programming groups, among others. He also has many contributions in the Source Code forum. Christian receives a $30 gift certificate to amazon.com.

Each person got only one vote, and occasionally people expressed a wish to vote for more than one contributor. Several other contributors received votes, and all who sent in votes were enthusiastic in support of their favorites. I enjoyed reading the submitted vote messages, and enjoyed being able to provide some money and recognition to some truly valuable people. To all who voted, thank you! I hope we can do it again sometime.

He's baaaaaack . . .

Gianluca Pivato is at it again. Most of his previous work has been specifically related to Centura Team Developer. Now he's about to release more broadly focused tools that allow you to call any Java class as if it were a COM server. Jasper and JasperX should be coming to market around the time you read this. Says Gianluca, "You need JDK1.2 or JRE1.2 installed, of course. There is only one sample

Continues on page TK

Nails Mark Hunter, **Glue** Dian Schaffhauser, **Grease** Shelley Doyle, **Paint** Paul Gould, **Licensed but not bonded and insured** Mocha

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.

Contact Us

Centura Pro on the Web
www.ProPublishing.com

Editorial Department
Phone: 818-249-1364
Fax: 818-246-0487
E-mail: centurapro@visto.net

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

Control the Calendar . . .

Continued from page 1

Listing 2. Send data to the custom control, allocating and freeing a memory buffer in the process.

```
◆ Function: SetToday
  ◇ Description:
  ◇ Returns
  ◆ Parameters
    ◇ Date/Time: dDate
  ◇ Static Variables
  Local variables
    ◇ String: sDate
    ◇ Number: nDateYear
    ◇ Number: nDateMonth
    ◇ Number: nDateDay
    ◇ Number: nAddr
  ◆ Actions
    ◇ Call SalDateToStr( dDate,  sDate)
    ◆ If SalStrIsValidDateTime( sDate )
      ◇ Set nDateYear = SalDateYear( dDate )
      ◇ Set nDateMonth = SalDateMonth( dDate )
      ◇ Set nDateDay = SalDateDay( dDate )
      ◇ Set nAddr = CStructAllocFarMem( 16 )
      ◇ Call SalStrSetBufferLength( sDate, 16 )
      ◇ Call CStructPutWord( sDate, 0, nDateYear )
      ◇ Call CStructPutWord( sDate, 2,nDateMonth )
      ◇ Call CStructPutWord( sDate, 6,nDateDay )
      ◇ Call CStructPutWord( sDate, 8,1 )
      ◇ Call CStructPutWord( sDate, 10,1 )
      ◇ Call CStructPutWord( sDate, 12,1 )
      ◇ Call CStructCopyToFarMem( nAddr, sDate, 16 )
      ◇ Call SalSendMsg( hWndCalendar,
        MCM_SETTODAY,0,nAddr)
      ◇ Call CStructFreeFarMem( nAddr )
    ◆ Else
      ◇ Call SalMessageBox( 'Bad date format !',
        'Error', MB_Ok | MB_IconStop )
```

There's one exception to the messaging paradigm: The Common Control class itself must be registered using a function call to InitCommonControlsEx( ) before any other interaction can take place.

Month Calendar

In the sample application, frmCalendar contains the basic calendar controls, plus a large number of other controls that let you to change the appearance and behavior of the calendar. In **Figure 1**, for example, you can see that the MCS_MULTISELECT style has been turned on, allowing the user to drag across a range of days in the calendar rather than just a single day.

The Common Control class is MONTHCAL_CLASS (in C) or SysMonthCal32 (in CTD).

To initialize COMCTL32.DLL and register the class, the first function called must be InitCommonControlsEx() as shown in **Listing 3**, with flag ICC_DATE_CLASSES (the first parameter is 8: the size of the INITCOMMONCONTROLSEX structure).

Listing 3. You must initialize the Common Controls DLL before any other interaction.

```
On SAM_AppStartup
  Call InitCommonControlsEx( 8, ICC_DATE_CLASSES)
```

Styles

MCS_DAYSTATE—This style allows you to display days in bold (the calendar must be re-created to change it at runtime).

MCS_MULTISELECT—The user can select a range of dates, instead of only one (the calendar must be re-created to change this style at runtime).

MCS_WEEKNUMBERS—Week numbers will be displayed to the left of calendar.

MCS_NOTODAYCIRCLE—No circle will be drawn around the "today" date.

MCS_NOTODAY—The "today" date won't be displayed at the bottom of calendar.

Messages

MCM_GETCURSEL—Retrieves the currently selected date. The lParam parameter receives the date in a SYSTEMTIME structure. This structure has the following form:

```
{
  WORD wYear;
  WORD wMonth;
  WORD wDayOfWeek;
  WORD wDay;
  WORD wHour;
  WORD wMinute;
  WORD wSecond;
  WORD wMilliseconds;
}
```

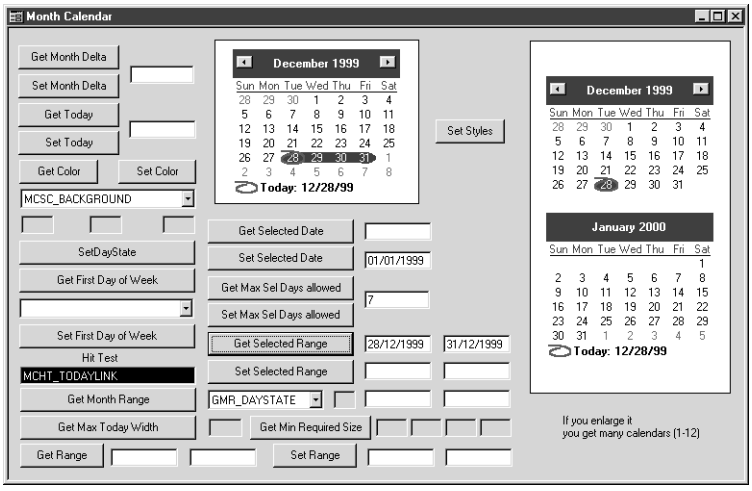

Figure 1. Some of the many ways to manipulate the calendar control.

I call the function SendMessageTimeoutA( ) to get a string pointer in lParam.

The parameters I want to get are wYear, wMonth, and wDay; so I call CstructGetWord( ) with offsets 0, 2, and 6, then call SalDateConstruct( ) to build the date.

MCM_SETCURSEL—It's the reverse of MCM_GETCURSEL: I pass a SYSTEMTIME structure in lParam. It must first be initialized with CstructPutWord( ).

MCM_GETMAXSELCOUNT—It gets the maximum date range that can be selected. It's simple; the returned value from the message gives me this value.

MCM_SETMAXSELCOUNT—It sets the maximum date range that can be selected. The value must be passed in wParam.

MCM_GETSELRANGE—It receives two dates that represent the range actually selected by the user. It's like MCM_GETCURSEL but with two SYSTEMTIME structures returned. So I call CstructGetWord( ) with offsets 0, 2, 6 for the first one and 16, 18, 22 for the second one.

MCM_SETSELRANGE—It selects a range between two dates. I must pass two SYSTEMTIME structures in lParam; they must be initialized with CstructPutWord( ).

MCM_GETMONTHRANGE—It retrieves the range of visible months. The wParam parameter indicates which scope to be retrieved: GMR_DAYSTATE for preceding and trailing months or GMR_VISIBLE for the current month entirely displayed. The lParam parameter returns two SYSTEMTIME structures. The value returned by the message represents the number of visible months displayed between the two dates of lParam.

MCM_SETDAYSTATE—Displays some visible days in bold. The wParam parameter is the value returned by MCM_GETMONTHRANGE with wParam.

GMR_DAYSTATE—The lParam parameter is a pointer to MONTHDAYSTATE arrays of bits (1-31). When a bit is set to on, the corresponding day will be in bold (the number of arrays is given by wParam).

MCM_GETMINREQRECT—It gets the minimum size (RECT structure) required to display a full month. The RECT structure is returned in lParam. I call CstructGetLong( ) with offsets 0, 4, 8, 12.

MCM_SETCOLOR—This message sets the color for a given part of the calendar. The wParam parameter contains the part (Background, text, title, etc). The lParam parameter contains the RGB color to be set.

MCM_GETCOLOR—Retrieves the color for a given part of the calendar. The wParam parameter contains the part (Background, text, title, etc). The returned value is the RGB color of the given part.

MCM_SETTODAY—Sets the "today" date. The lParam parameter must contain a SYSTEMTIME structure.

MCM_GETTODAY—Retrieves the date for the "today" date. The date is returned in lParam (SYSTEMTIME structure).

MCM_HITTEST—This messages allows you to determine which part of the calendar is under the mouse cursor. The lParam parameter points to a MCHITTESTINFO structure which has the following form:

```
{
UINT cbSize;
POINT pt;
UINT uHit;
SYSTEMTIME st;
}
```

So I must pass 32 for the cbSize (4+8+4+16) in offset 0 and X and Y coordinates in offsets 4 and 8. The Hit Test is the returned value.

MCM_SETFIRSTDAYOFWEEK—Sets the first day of the week (the default day is Monday) where a week begins. The lParam parameter contains the new first day (Monday = 0, Tuesday = 1,..., Sunday = 6).

MCM_GETFIRSTDAYOFWEEK—Retrieves the first day of the week. The low word of the value returned is the current first day of the week.

MCM_GETRANGE—It receives two dates that represent the minimum and maximum dates allowed. The lParam parameter receives two SYSTEMTIME structures.

MCM_SETRANGE—It sets the minimum and/or the maximum dates allowed. The wParam contains a flag indicating which dates are set (GDTR_MIN and/or GDTR_MAX). I must pass two SYSTEMTIME structures in lParam: they must be initialized with CstructPutWord( ).

MCM_GETMONTHDELTA—Retrieves the scroll rate for a month (when the user clicks on arrows Left or Right). The returned value contains the month delta.

MCM_SETMONTHDELTA—Sets the month delta. The month delta must be set in wParam parameter.

MCM_GETMAXTODAYWIDTH—Retrieves the maximum width of the "today" string. The returned value contains this width in pixels.

Date and Time Picker

In the sample application, the Date Time Picker form contains many child controls demonstrating how to manipulate the appearance and behavior of the custom control. See **Figure 2**.

The Common Control class is DATETIMEPICK_CLASS (in C) or SysDateTimePick32 (in CTD). To initialize COMCTL32.DLL and register the class, the first function called must be InitCommonControlsEx( ) with the same flag ICC_DATE_CLASSES as for Calendar control.

Styles

In the sample application, a dialog box (**Figure 3**) is used to select multiple styles.

DTS_APPCANPARSE—This style allows the owner to parse user input and take necessary action (the user can use F2 to edit the client part of the control).

DTS_LONGDATEFORMAT—Displays the date in long format.

DTS_RIGHTALIGN—The drop-down Month Calendar will be right-aligned with the control (left-aligned by default).

DTS_SHOWNONE—Add a check box that user can check once he has selected a date (Date and Time Picker must be re-created to change this style at runtime).

DTS_SHORTDATEFORMAT—Displays the date in short format.

DTS_TIMEFORMAT—Displays a time instead of a date (Date and Time Picker must be re-created to change this style at runtime).

DTS_UPDOWN—Uses an Up-Down control instead of a drop-down Month Calendar (by default). (Date and Time Picker must be re-created to change this style at runtime).

Messages

DTM_GETSYSTEMTIME—Retrieves the current selected date and time of the control. The lParam parameter receives the date and time in a SYSTEMTIME structure. So I call CstructGetWord( ) with offsets 0, 2, 6, 8, 10, 12.

DTM_SETSYSTEMTIME—Sets the date and time for the control. The wParam parameter must be GDT_VALID or GDT_NONE (which unchecks the eventual check box if the style DTS_SHOWNONE is set and ignores lParam). The lParam must contain a SYSTEMTIME structure.

DTM_GETRANGE—It receives two dates that represent the minimum and maximum dates allowed. The lParam parameter receives two SYSTEMTIME structures.

DTM_SETRANGE—It sets the minimum and/or the maximum dates allowed. The wParam contains a flag indicating which dates are set

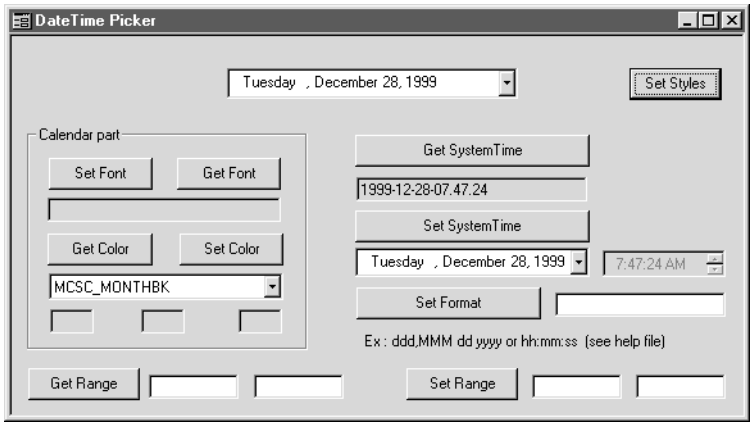

Figure 2. Use the code attached to the child controls as the basis for your own logic.

Figure 3. Multiple styles may be selected. If none are selected, DTS_SHORTDATEFORMAT becomes the default.

(GDTR_MIN and/or GDTR_MAX). I must pass two SYSTEMTIME structures in lParam: they must be initialized with CstructPutWord( ).

DTM_SETFORMAT—Sets the string format for the control. The string is passed in the lParam with the function SendMessageString( ). Valid formats can be found in the Win32 Help file at functions GetDateFormat( ) and GetTimeFormat( ).

DTM_SETMCCOLOR—This message sets the color for a given part of the drop-down Month Calendar. The wParam parameter contains the part (Background, text, title, etc). The lParam parameter contains the RGB color to be set.

DTM_GETMCCOLOR—Retrieves the color for a given part of the of the drop-down Month Calendar. The wParam parameter contains the part (Background, text, title, etc). The returned value is the RGB color of the given part.

DTM_GETMONTHCAL—Retrieves the handle of the drop-down Month Calendar. The return value is this handle, which is only valid between DTN_DROPDOWN and DTN_CLOSEUP notifications.

DTM_SETMCFONT—Sets the font for dates displayed in the drop-down Month Calendar. The handle of the font is passed in the wParam parameter. The lParam parameter is a Boolean to indicate if the Calendar must be redrawn immediately. (If you set a big font… you'll have an enormous Calendar).

DTM_GETMCFONT—Retrieves the font used by the drop-down Month Calendar. The returned value contains the handle of this font.

Sample source code

The sample application, CALENDAR.APT, is saved as text, but it assumes CTD version 1.5. If you're using CTD 1.1, simply edit WIN32APISMALL.APL and change the external function library declaration STRC15.DLL to end in "11" rather than "15".

Naturally, you may find that your applications already define some of the external functions used by the sample applications. The definitions for the Cstruct functions, and for the Windows API functions needed here, are contained in WIN32APISMALL.APL, along with a few Windows constants. The calendar-related system constants defined in this article are contained in COMMCTRL.APL. **CP**

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

Christian Astor has been a SQLWindows/Centura developer since 1993. He's also a Visual C++ programmer (C and API Win32). He's currently working at Maguid (SSII) for Alcatel CIT. Christian is addicted to graphics, animation, 3D. He can be reached at castorix@club-internet.fr or christian.astor@alcatel.fr.

Return Call

Christian Schubert

He's back! The author takes his theoretical work with callbacks and makes it practical!

In "I'll Call You Back" in the December 1999 issue of *Centura Pro*, I showed a way to use callbacks with CTD. Now I'll offer a couple of examples of callback functions that could be useful for developers. This month we'll look at the Microsoft Telephone API (TAPI).

What TAPI's good for

A detailed description of TAPI would be far beyond the scope of this article, so let this short introduction begin with some words from Chris Sells, author of *Windows Telephony Programming*: "TAPI was designed to contain the collective features of all of the telephony equipment and all of the telephony communications protocols on the planet."

Unfortunately, the result isn't easy to understand. There are at least 120 different functions with lots of flags and return codes. Using TAPI you can control devices directly attached to your computer or attached via network, make calls or receive them, answer calls automatically, record voice messages, redirect calls to a different phone number, distinguish between different services (voice, fax, or data), and lots more.

SQLWindows 32

This article, however, will pick out only a few of those features. I'm going to show how to trigger a call from within Centura Team Developer, how to be notified of an incoming call, and how to extract information about the caller (such as the telephone number). In most cases you encounter, this will be sufficient. If you require more information, you'll find additional resources in the sidebar, "Call Center," accompanying this article.

How TAPI works

If you want to control some piece of hardware, you'll need a suitable driver for it (that's probably nothing new). TAPI drivers for more common communication devices like standard modems are already included in Windows 95, 98, and NT 4.0.

So if you put some less common device into your PC (ISDN card, or cable modem) or connect your phone to the COM port, you'll have to install a manufacturer-supplied driver (TSPI) that "knows" how to handle this device. Programming for different telephony devices is quite similar because the device-specific functions are mapped into a common set of functions: TAPI. **Figure 1** shows the different layers between a piece of communications hardware and an application.

The TSPI driver will be able to tell a TAPI application which features are available for this specific device. For example, a modem will surely be able to dial a given number using touch tones or pulse dial, but it won't necessarily be able to detect a caller's phone number (at least not every telephone provider supports the transmission of caller information via analog lines). So an application can determine which features are supported and can prevent the user from selecting unsupported options.

Remarkable facts about TAPI programming

Some of the functions we encounter in TAPI programming work asynchronously. That means that the effects of a functions won't necessarily be visible when the

Figure 1. TAPI architecture.

function returns. This is also the main reason for using a callback function (remember, this article was meant to deal with callback functions). For example, when dialing a number using the function *lineMakeCall*, the function returns right after it has transmitted the necessary command to the communications hardware. If it would wait until there were a response from the device, the application could be blocked for several seconds (go off hook, wait for dialtone, dial, wait for response, and so on). Instead a callback is sent when the response from the communication device is available. The response could be a positive one (call was successful) or a negative one.

There's one more thing to have a look at: Some of the asynchronous functions take pointers to numbers (LPHANDLE) as parameters that only point to valid data *after* a decent callback has been received. However, this is a problem for CTD. Receive parameters are evaluated after calling an external function, right before the control is returned to the application (but in most cases before a callback was received). At this time, the values aren't yet valid. Fortunately, there's a way to get the desired values. Instead of defining such parameters as Receive Number, we define them as Receive String: LPVOID. Obviously, when using this, a real pointer to a string is submitted, which remains valid even after the external function call has returned and until the callback is received. Then the numeric value can be extracted using CStructGetLong (see "The Sample Applications" section).

Steps to making a call

I won't explain the TAPI functions used in the samples in detail; only enough to understand how it works. For a further explanation see below.

The following functions are required to make a call, in this exact sequence:

1. ***lineInitialize.*** Initializes the whole API and takes the pointer to the callback mentioned above. You should use lineInitializeEx instead when your target platform supports TAPI 2.0 and above.

2. ***lineGetDevCaps* (optional).** Elicits useful information about the available lines and capabilities of each.

3. ***lineOpen.*** Opens one of the existing lines and returns a line handle. Line events will be sent to the callback function if privilege parameter is LINECALLPRIVILEGE_MONITOR or LINECALLPRIVILEGE_OWNER.

4. ***lineMakeCall.*** Starts a call using the line returned by the lineOpen.

5. ***lineHold* (optional).** After successfully establishing a connection put the call on hold; call handle required.

6. ***lineUnhold* (optional).** Unhold the call.

7. ***lineDrop.*** Hang up; call handle required.

8. ***lineDeallocateCall.*** Clean up all remains of a call.

9. ***lineClose.*** Close the current line.

10. ***lineShutdown.*** Shut down TAPI.

Caller detection

One of the really "fine" features of TAPI is the capability of retrieving information about the calling party before answering a call. This can be used to establish a list of incoming calls when the call can't be answered immediately, to retrieve further information about the caller from a database (think of a call-center), or just to detect unwanted callers.

Caller information retrieval isn't supported by every device. For those that support it, programming is quite easy. Just make sure the line privilege flag of lineOpen is LINECALLPRIVILEGE_MONITOR or LINECALLPRIVILEGE_OWNER. The callback function bound to the TAPI when calling lineInitialize will be notified with a LINECALLSTATE_OFFERING message when a call arrives. lineGetCallInfo can be used to extract some information.

Figure 2. A demo of tapiRequestMakeCall - DEMO.

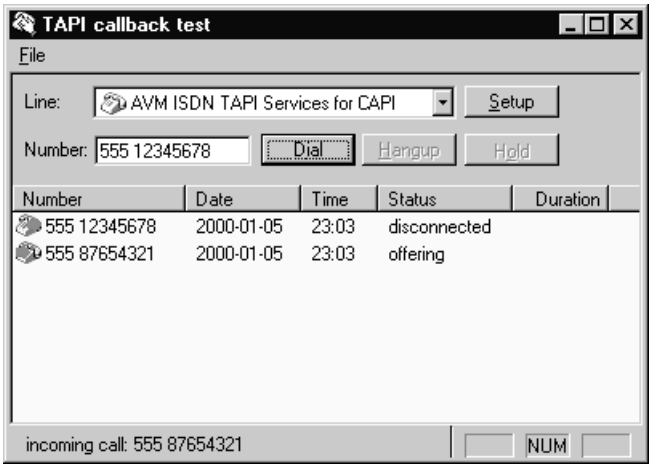

Figure 3. TAPI_Callback.APT.

The sample applications

Assisted telephony: Tapi_small.APT

I include Tapi_small.apt for reasons of completeness. It shows the use of the function tapiRequestMakeCall, which can be used to trigger a call in conjunction with an existing dialer application (most Windows versions come with a dialer application). Usage is simple—just one call:

```
◇Call tapiRequestMakeCall
( dfNumber, sAppTitle, STRING_Null, STRING_Null )
```

However, this call might bring up additional windows used by the dialer application, which might not be closed automatically after the call has finished. This function doesn't require a callback. If you need a quick solution, look at it.

TAPI_Callback.APT

The second application uses a handful of basic TAPI functions to give you an idea of how it can be done.

The upper half contains a combo box to select the line that should be used for dialing. The line's configuration dialog can be called using the Setup button. Not every device supports a configuration dialog. Lines capable of making a voice call show a small telephone symbol (not every TAPI-compatible device supports voice telephony—think of faxes or data modems). There's a data field to enter the phone number and three buttons to dial, hang up, and put the call on hold (not supported by all devices).

The lower half contains a list view (from Visual Toolchest) used to show outgoing and incoming calls. The upper and lower halves are put together with an invisible splitter.

The callback function

Remember, this article is about callback functions. TAPI uses one with six parameters defined like this (in C):

```
typedef void (CALLBACK * LINECALLBACK)(
    DWORD               hDevice,
    DWORD               dwMessage,
    DWORD               dwInstance,
    DWORD               dwParam1,
    DWORD               dwParam2,
    DWORD               dwParam3
    );
```

Using the CALLBACK.APL described in my previous article, we can define this function like so:

```
◇Call SetCallback ("Callback", 6,
FALSE, hTapiProcAddr, hTapiProcLib )
```

What's This?

Centura Tip!

Sven O. Rimmelspacher—In his January article, "HTML-style Help for CTD," Joachim Meyer described the function TextPopup( ), which shows a little help pop-up window for a control, usually used for the so-called "What's This?" help. Here's how you can implement this little button with a question mark in your dialogs and call the context help.

The button can be added to a dialog calling the following function:

```
◆ On WM_NCCREATE
  ◇ Call SetWindowLongA( hWndForm, GWL_EXSTYLE,
      WS_EX_CONTEXTHELP |
      GetWindowLongA( hWndForm, GWL_EXSTYLE ) )
```

Now when you start your application and click on that new button in the upper right corner of the dialog, the cursor changes into an arrow with a question mark. Your next click on a child control should then show the context help. But be aware that SAM_Help doesn't work—therefore we have to use WM_HELP (0x0053) for each control that should provide a What's This dialog.

Another problem is that you can't decide whether the user used the F1 button for calling the help or used the What's This button, because both send the same WM_HELP message (and even the HELPINFO structure that's pointed to by the lParam of this message doesn't show a difference). The solution for this could be that you trap WM_SYSCOMMAND (0x0112) in the parent window and check if wParam has the value SC_CONTEXTHELP (0xF180), which is used when the user clicks on the What's This button. Now you can set a flag that you're in "What's This mode" and check this flag on the WM_HELP messages described above.

SQLWindows

32

Download WHAT_THIS.ZIP from this issue's table of contents at www.propublishing.com or find it on this month's Companion Disk.

Sven O. Rimmelspacher is a senior developer and project manager at Pickert & Partner GmbH, a company with a quality management system. He's also the author of IntelliDoc, a documentation tool for SAL code. He can be reached at sven@rimpi.de.

The return values hTapiProcAddr and hTapiProcLib are the address of the callback function and the module handle of the DLL, which are needed for the lineInitialize function.

The application defines two classes. The call clsTapiFunc contains basic functionality for initialization and wrappers for some of the TAPI functions. clsFrmTapi is a form window class derived from clsTapiFunc. It defines the callback function, which is intended to be used as a late-bound function. Therefore, it doesn't do anything; it just shows the function signature you could use in your replacement:

```
◆Function: Callback
...
  ◆Actions
    ◆Select Case nMessage
      ◇! status of current call
      ◆Case LINE_CALLSTATE
        ◆Select Case nParam1
          ◇! no current connection
          ◆Case LINECALLSTATE_IDLE
            ◇Break
          ◇! it's ringing
          ◆Case LINECALLSTATE_OFFERING
            ◇Break
          ◇! we are dialing
          ◆Case LINECALLSTATE_DIALING
            ◇Break
          ◇! we are connected
          ◆Case LINECALLSTATE_CONNECTED
            ◇Break
          ◇! we are on hold
          ◆Case LINECALLSTATE_ONHOLD
            ◇Break
          ◇! other party is busy
          ◆Case LINECALLSTATE_BUSY
            ◇Break
          ◇! call has been disconnected
          ◆Case LINECALLSTATE_DISCONNECTED
            ◇Break
      ◆Case LINE_REPLY
        ◇Break
```

The LINE_CALLSTATE message can be nicely used to inform the user about the current status of the call. This is done in the overriding callback function found in *frmControls*. Here's one small part, for instance:

```
◆Case LINECALLSTATE_DIALING
  ◇! Set status text with number
  ◇Call SalStatusSetText ( hWndMessage,
    "dialing " || sPhoneNumber || " ..." )
  ◇! Add row to the list view
  ◇Set nItemIndex = AddOutgoing ( sPhoneNumber,
    "dialing" )
  ◇! Enable hangup button
  ◇Call SalEnableWindow ( pbHangUp )
  ◇Break
```

The LINE_REPLY message is needed to extract the call handle after a call has been triggered (remember what I said about asynchronous functions):

```
 ◇! we use a string buffer „sBuffer"
    ! instead of a number here
◇Set nRequestID = lineMakeCall( hOpenLine,
    sBuffer, sPhoneNumber, nCountryCode, 0 )
....
◆Case LINE_REPLY
  ◆If nParam1 = nRequestID
    ◆If nParam2 = 0
      ◇! Extract the numeric
        ! handle from the buffer
      ◇Set hCall = CStructGetLong ( sBuffer, 0 )
  ◇Break
```

There are certainly more messages sent to the callback function. I have shown only the ones we use in the sample.

A little troubleshooting

Programming with TAPI can give you trouble for several reasons:

- TAPI isn't installed properly or is the wrong version. Though this isn't likely, you should check out the possibility. Also, a bug in the TAPI that comes with Windows 95 prevents you from using the TAPI functions once you shut it down.

- Not every TSP supports all functions. Call lineGetDevCaps *or* lineGetAddressCaps (not part of the sample) to determine whether your driver supports the function.

Continues on page 12

Call Center

To learn more about the use of TAPI, I recommend:

- *Windows Telephony Programming* by Chris Sells, Addison-Wesley
  This book is intended for C/C++ programmers, but the samples are easy to understand. The author introduces a class framework to encapsulate the TAPI. There are samples for different programming problems (dialing, answering, etc.). Sells even shows how to get an answering machine to play and record WAV files.

- *MSDN Online:* http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/psdk/tapi21/tapilgl_8mw9.htm offers complete documentation of all available functions.

- ftp://ftp.microsoft.com/developr/TAPI offers samples and tools in C/C++.

- Usenet:
  microsoft.public.win32.programmer.tapi
  microsoft.public.win32.programmer.tapi.beta

—Christian Schubert

Sorting Tables List View Style

Sven O. Rimmelspacher

Have you ever thought about sorting a table list view-style? List view style means that you click on a column header and the table is sorted increasing by that column. If you click again on the same column header, the table is sorted descending, and so on. The function used for sorting a table is SalTblSortRows( ), and if you want to use this function you need to know some things about the current table.

Put a little flash in your table sorting!

What you need

The SalTblSortRows( ) function takes the following three parameters:

```
Window handle: hWndTbl ! The handle of a table window.
Number: nColumnID      ! The column Identifier of the column by which to sort.
Number: nOrder         ! The direction of the sort. Specify either:

TBL_SortDecreasing     ! which is 1, we need that later
TBL_SortIncreasing     ! which is 0, we need that later
```

The handle of the table window is usually easy; just use hWndItem. The column ID can be retrieved using SalTblQueryColumnID( ), if you've clicked column's window handle. But how do you get this? Certainly, you can use some Windows messages, but then you have to find out which part of the table was clicked and evaluate that click. If you browse through the list of SAM_ messages, you find one called SAM_ColumnSelectClick, which seems to be the solution. But before you can use it, you have to set TBL_Flag_SelectableCols to TRUE using the SalTblSetTableFlags( ) function.

One important thing about SalTblSortRows( ) is that that table must be non-discardable and the Max Rows in Memory property must be large enough to hold all the data.

Once the message is fired, the wParam contains the window handle of the clicked column as a number. This number has to be converted to a window handle using SalNumberToWindowHandle( ). Now you've got everything you need. Retrieve the column ID as stated above and call SalTblSortRows( ).

Nice, but we want more!

With that functionality you can now sort a table by clicking on the column header. But there are still some things to do. First, I don't like that the column I've clicked on is now selected. I want to switch that selection off. Use SalTblSetColumnFlags(hWndCol, COL_Selected, FALSE).

The feature we're still missing is the ability to sort ascending and descending alternately. Since we've created a class with that sorting function and this should work for all instances of this class, use an array to store the information about the last sort. This array gets a flag for every column you click on and can remember all click actions that are taken. It's just a Boolean array that either contains the TBL_SortDecreasing or TBL_SortIncreasing flag, which is toggled each time you click on the header. The only special case we have here is that every first time you click on a column (after clicking on another), the sort direction is always ascending, so you have to remember which column was clicked last.

For toggling I discovered that the TBL_Sort flags are 0 and 1, respectively, which leads to this easy toggling mechanism:

```
◇ Set m_baColumnSort[nColumnId] =
1 - m_baColumnSort[nColumnId]
```

instead of the more complex:

```
◆ If m_baColumnSort[nColumnId] =
  TBL_SortAscending
  ◇ Set m_baColumnSort[nColumnId] =
    TBL_SortDescending
◆ Else
  ◇ Set m_baColumnSort[nColumnId] =
    TBL_SortAscending
```

Now it works as you'd expect. I created a simple test application that uses this mechanism and contains the complete code for the table window class.

Listing 1. The sort function of the table window class.

```
◆ Function: __Sort
  ◇ Description: Sort the table contents by
    the clicked column, called from SAM_ColumnSelectClick
    IMPORTANT: If you want to use this feature,
    the table must be non-discardable and
    Max Rows in Memory must be large enough to hold all
the data
  ◇ Returns
  ◆ Parameters
    ◇ Window Handle: p_hWndCol   ! the column
            ! that was clicked
  ◆ Local variables
    ◇ Number: nColumnId
  ◆ Actions
    ◇ ! deselect the column
      if you like to show that this column was
      clicked, just remove this line
    ◇ Call SalTblSetColumnFlags( p_hWndCol,
      COL_Selected, FALSE )
    ◇ ! get the column ID
    ◇ Set nColumnId =
      SalTblQueryColumnID( p_hWndCol )
    ◇ ! last column clicked once again?
    ◆ If nColumnId = m_nLastColumnId
      ◇ ! a little dirty but works
        nice TBL_SortDecreasing is 0,
          TBL_SortIncreasing is 1
      ◇ ! toggle sort direction
      ◇ Set m_baColumnSort[nColumnId] =
        1 - m_baColumnSort[nColumnId]
    ◆ Else
      ◇ ! on first click sort always increasing
      ◇ Set m_baColumnSort[nColumnId] =
        TBL_SortIncreasing
      ◇ ! remember last column
      ◇ Set m_nLastColumnId = nColumnId
    ◇ ! sort by this column
    ◇ Call SalTblSortRows( hWndItem, nColumnId,
      m_baColumnSort[nColumnId] )
```

CP

Download SORTROWS.ZIP from this issue's table of contents at www.propublishing.com, or find it on this month's companion disk.

Sven O. Rimmelspacher is a senior developer and project manager at Pickert & Partner GmbH, a company with a quality management system. He's also the author of DocSal, a documentation tool for SAL code. He can be reached at sven@rimpi.de.

Return Call . . .

Continued from page 10

- Not every TSP driver sends all callback messages. The one I used in my tests was quite complete (AVM Fritz Card) but the generic Microsoft Modem driver isn't. It omits the LINECALLSTATE_DISCONNECTED message and sends the LINECALLSTATE_IDLE right after disconnecting. **CP**

Download Callback2.zip from this issue's table of contents at www.propublishing.com or find it on this month's Companion Disk.

Christian Schubert works as Senior Developer at GODEsys GmbH, Mainz, Germany. He has been using Centura products since 1993 and develops sales management applications. Reach him at christian.schubert@mainz.netsurf.de.

And the Winner Is . . .

Continued from page 2

app for now, but in Visual Basic or Active Server Pages you can go wild with the imagination—simply pick any Java class and use it as if it was COM. The wizard for CTD will be ready soon. VB, ASP, Delphi, Word, Excel, FoxPro and other automation clients don't need wizards."

With a lot of Pivato's products, you need to think about them for a few minutes before the full impact sinks in. I asked him for some additional information, and he said, "Jasper lets you use any Java class without registering it or converting it. It supports polymorphic calls, single dimension arrays of any type, and exceptions. Used in VB or ASP, it gives you the impression of actually working in Java. With Matterhorn it will be quite impressive, too, thanks to the new features with objects. JasperX is quite remarkable also; it embeds a Java frame that can contain any Java visual class, including windowless lightweight components such as Swing classes. JasperX also is able to 'listen' to most Java events."

Among his examples is a JasperX application that contains five Swing components in VB. Using those same components in an ActiveX created in VB, IE displays that ActiveX, aggregating the five components into a single one.

According to Pivato, "It's extremely fast and it supports multithreading. So far we've tested it with ASP, VB, VBA, Delphi and CTD. I haven't tried it, but since any Java class can be used, this should open the doors to RMI and CORBA as well as XML and many other free and ready-made classes."

I've tried the early sample, and it's very intriguing. See it at http://www.iceteagroup.com/download/aspj/aspj10.exe.

Matterhorn?

As Pivato noted above, Matterhorn is designed to offer many OOP improvements over the present SAL language. The ability to assign object references, constructors and destructors, and other features that have been standard in many OOP languages for years are coming to the next release of Centura Team Developer (or whatever they decide to call it!). Beta testers are pleased to have these tools available. We'll bring you detailed reviews and information at the earliest opportunity. **CP**

Centura Pro Classics on CD!

Searchable! Comprehensive! Long Overdue!

Tired of racking your brain, sifting through issues, trying to find that one programming technique you remember reading that'll save you two days of programming effort? Reach for *Centura Pro Classics*.

We've archived every issue of *Centura Pro* since 1996 in Adobe Acrobat format—completely searchable by keyword, author, date, and program. What could be easier? Order your copy before March 30, 2000 and save $50!

- **300** articles, editorials, and tips
- **47** complete issues from 1996, 1997 1998, and 1999
- **132** sample applications and utilities
- Uncle Fred and *How to Use OLE and ActiveX with Centura*
- Latest version of Adobe Acrobat Reader

Save $50

I've been waiting for this day!

Send me ***Centura Pro Classics on CD*** immediately. Since I'm a subscriber and I'm ordering by March 30, 2000, I'll pay only $99 for each! That's a $50 savings for our subscribers—plus free shipping worldwide!

		Quantity	Sub-total
Non-subscribers	**$149**	_____	$_____
Subscribers	**$99**	_____	$_____
		Order total	$_____
		7.375% tax (CA state residents only)	$_____
		Shipping	$ **FREE!**
		Total	$_____

Payment Method

❑ Visa ❑ MasterCard ❑ AmEx

❑ Check enclosed

Name on credit card _____

Credit card number _____

Expiration date _____

Signature _____

_____
Name

_____
Company

_____
Mailing Address Line 1

_____
Mailing Address Line 2

_____
City

_____
State/Provence Zip/Postal Code

_____
Country

_____
Email address

_____
Phone

Order Today!

PRO Publishing

Fax 530-265-0368

Phone 530-265-4082

Mail Pro Publishing
PO Box 2399
Nevada City, CA 95959

Web www.ProPublishing.com