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

Hosting a Parameter Party

Thomas Althammer

July 1999
Volume 4, Number 7

PRO Publishing

Thomas' article, "De-Sqribe Your Reports," in the May issue covered several aspects of the SQR reporting server. As promised, he now goes hands-on to show you the integration of Centura Team Developer with SQR server. Here he demonstrates a generic technique for constructing parameter entry masks dynamically—pretty nifty for similar products, such as Crystal Reports & Co. If you are still not interested in SQR, go ahead and jump to the paragraph titled "Parameter Party".

As mentioned in my previous article, the SQR server is just an executable. Actually, there are two additional program files, one for printing SPF files (the portable output format of SQR server) and one for executing precompiled reports. These two EXEs represent a subset of the complete functionality—they don't add anything not already in the main program, SQRW.EXE (or SQR.EXE on all non-Windows platforms). One SQR directory for each database brand contains all the necessary components.

To execute a report from the command line, enter the following:

```
SQRW.EXE [report-file]
[database]/[user]/[password] [flags] parameters...
```

Available command line flags include:

- Limiting the number of pages generated, disregard database access.
- Debug information, such as debug level.
- Path and file name of the error file and log file.
- Information on the output, such as a printer-specific file, the portable SQR file format, the Windows printer, HTML, etc.
- For SQLBase, isolation level and other DB-specific information.
- The path for additional include files.

Flags that are needed each time a report gets executed with SQR can be placed in the SQR.INI. One of these standard settings in the SQLBase-specific section of my SQR configuration file is "-LOCKRL", which causes SQR to use the Release Locks isolation level when connecting to the database.

Compilation and runtime parameters in SQR

You can pass arguments to an SQR report by using the ASK or INPUT commands. The value of an INPUT

Continues on page 3

Parameter Party …

Continued from page 1

statement is queried on each report run, whereas ASK is only processed on non-compiled reports. The ASK function can only be placed in the setup section of a report file, beneath the tag, "Declare-Report." The INPUT command can appear anywhere within the report.

Just place the command-line data in the same order as ASK and INPUT commands are executed throughout a report run, one after the other with a space in between. If SQR doesn't find a corresponding value and you haven't specified the "quiet mode," a dialog box pops up and lets the user enter the value. This only works with locally executed reports.

SQR include files are similar to SQLWindows code libraries. They have the same file format as regular SQR programs, and their content is merged with the file they're referenced from during compilation. Include files usually contain commonly used procedures, heading and footing functions, and so on.

So, if you set up different include files, you can use the appropriate one depending on the value entered at the ASK command to compile them for different company letterhead designs, for example. Variable substitution scanning takes place before the #INCLUDE command is processed. This allows you to substitute all or part of the include file name at runtime, adding flexibility to controlling which file is included for the run.

The C interface

In addition to the normal command line interface of the SQR server executable, you can use an external function interface provided in SQRW.DLL. The supported functions are:

```
sqr
sqrcancel
sqrend
```

The command line arguments match the ones mentioned above for the direct call of the server executable.

Although the sqr( ) function works synchronously, you can still cancel the running SQR program with sqrcancel( ) using another thread. (Soon you'll be able to do that with the help of the upcoming Ice Tea Foundation Classes.)

sqrend( ) frees memory and closes cursors left open from previous report executions. The minimum needed for running a report through this interface are five DLLs, a very small footprint, indeed.

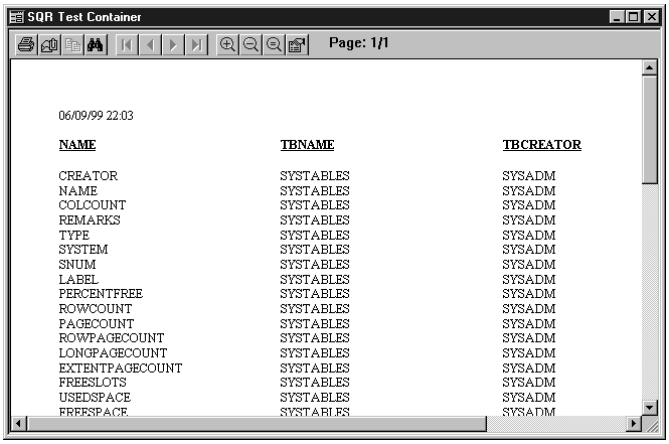

Figure 1. The SQR Viewer ActiveX Control.

ActiveX interface

Sqribe ships a product called "InSQRIBE," a set of three ActiveX components, to integrate with the customer's development environment (for example, Centura Team Developer).

SQR ActiveX Control

In addition to the asynchronous LocalRun( ) function that accepts the same command line flags as the sqr( ) function in the DLL, LocalRunNoWait( ) executes a report asynchronously.

As de-sqribed in my previous article, it's possible to scale up to the third tier with remote execution on an SQR server. The functionality, including TCP/IP and FTP communication handling, is provided by a set of functions in the ActiveX control as shown in **Listing 1**.

Listing 1. Functions exposed by the SQR ActiveX control for remote execution and retrieval of reports.

```
RemoteConnect
RemoteDisconnect
RemoteCancel
RemoteIsConnected
RemotePut
RemoteGet
RemoteDeleteFile
RemoteExec
RemoteExecNoWait
RemoteGetResults
RemoteCheckFile
RemoteGetShellType
RemoteGetServerType
RemoteGetUniqueSqrFileName
RemoteGetUniqueSqtFileName
RemoteSetDebug
```

SQR Viewer ActiveX Control

The SQR viewer is similar to other viewer controls. It lets you display a SPF, optionally with a toolbar.

The functions supported by the control are shown in **Listing 2**.

Listing 2. Functions exposed by the SQR Viewer ActiveX control.

```
Open
Close
SizeTo
ZoomIn
ZoomOut
Copy
Find
PrintSpf
AboutBox
SetCurrentPage
GetCurrentPage
SetCurrentScale
GetCurrentScale
GetPageCount
ShowToolBar
IsToolbarVisible
IsOpen
CanCopy
CanFind
```

As you can see in the listing, there's pretty much everything you need. If you look for a function to save a report, you won't find one. Instead, make a call to the SQR engine via one of the three different techniques described above specifying "–PRINTER:EH" for HTML and CSV and "-PRINTER:PD" for PDF output.

A very nifty feature of the SQR viewer control is the Copy and Find functions. You can search through an SPF file and copy parts of it. Text will be pasted as tab delimited in Excel, for example; graphics and graphs are copied as bitmaps and placed in the Clipboard.

Figure 2 shows the integrated ActiveX viewer within an application.

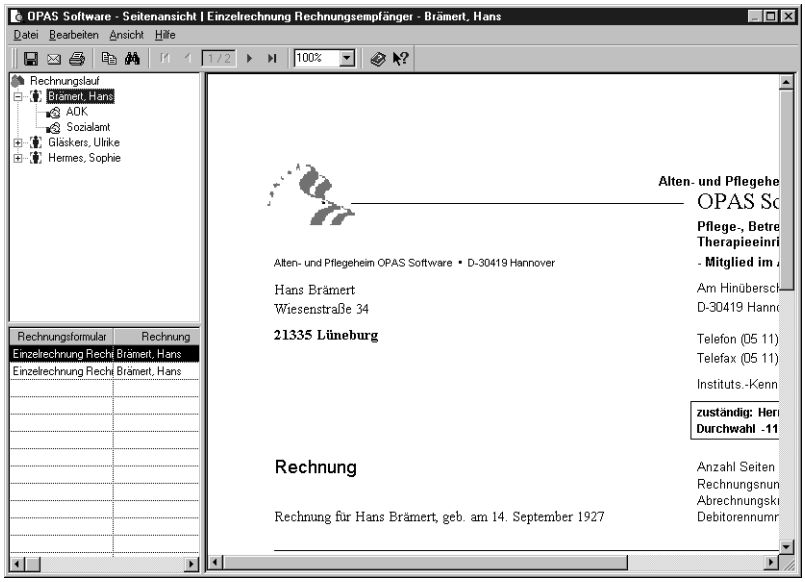

Figure 2. The SQR Viewer ActiveX Control used to display invoices.

SQR Printer ActiveX Control

This is a wrapper around the printing function of the SQR engine. Command line flags and usage work as I've already explained. For printing SPF files displayed in the viewer control, use the PrintSpf( ) function instead.

Parameter Party

Feeding reports with parameters is a pretty easy thing to do as long as you know the necessary data at design time. However, if the user can design and extend reports and use them at different places within the program, certain information might be required to execute a report successfully or decrease the amount of data presented, for example, by passing begin/end dates for a certain range of dates.

As long as there's only one value to be entered, this doesn't really cause any problems—even SQR prompts you for the value in the local set up if not put into "quiet mode." Entering one value after the other in different pop up windows can be quite annoying, though.

Developer and author Gianluca Pivato once presented a technique for creating Windows controls during runtime. You entered the types of parameter requested in his sample application, and the parameter window got generated. Agreed, this is a nice and clean technique. But in my case it's just not suitable, because I have several "business controls," such as tree views and combo boxes with small shortcut fields in front of them. These are controls and groups of controls known to the user from the application, and my parameter window should look similar to the main application window. I was looking for a way to have a dynamically created parameter interface and still use instances of my predefined classes. Dynamic object instantiation of visual controls isn't yet possible with SQLWindows, but there's a work-around . . .

Good old SalCreateWindowEx( )

Does this function ring a bell? Well, it's fairly new (existing since SQLWindows 5.0.2) and got covered several times in earlier issues of *Centura Pro*. Easy trick: set up a class cParameterForm and create one instance for each parameter type/business object you have, as demonstrated in **Listing 3**.

Listing 3. Several possible elements of a parameter mask containing parameter types/business controls.

```
cParameterForm: __pFormText
cParameterForm: __pFormTextLong
cParameterForm: __pFormNumber
cParameterForm: __pFormMoney
cParameterForm: __pFormDate
cParameterForm: __pFormBoolean
cParameterForm: __pFormDrive
```

Each form window is very "short," usually less than two centimeters tall. It contains one or more controls and represents one parameter value. __pFormText, for example, contains a string data field, __pFormDrive a

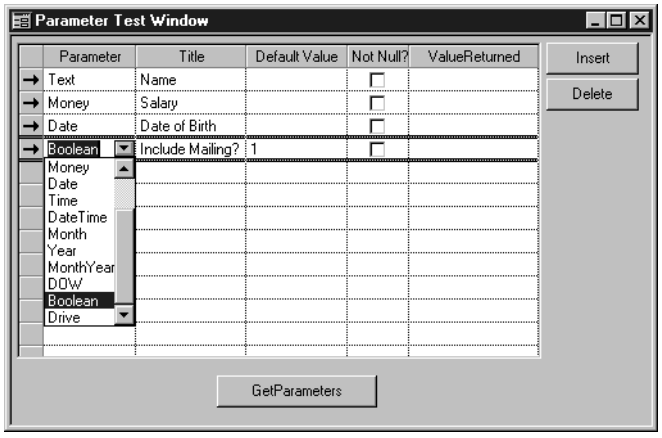

Figure 3. A sample application for defining the dynamically-created parameters. This information could come from the database or a report definition INI file.

combo box for selecting a drive letter. These forms will never be instantiated independently—they're always painted onto a container window.

__dlgParameterSelection is the container and parent of the dynamic parameters created.

Add your own

All you have to do to set up your own custom parameter type is the following:

1. Derive a new form window from the cParameterForm class.

2. Place your control(s) in there, such as table windows, data fields, and pictures.

3. Overwrite the late-bound functions getValue( ) and setValue( ).

4. Add an appropriate PAR_ constant to the user constants section with a unique ID.

5. Set up the corresponding windows name and height in the __dlgParameterSelection.createParameter( ) function—it has to be part of the Select Case statement.

Has the penny dropped? Right, top-level windows can be created at runtime in your empty parameter dialog; you can paint several top-level windows in the client area, one beneath another beneath another. Each contains the controls necessary to handle one of the parameters. Now it's fairly easy to generate your own parameter masks dynamically. However, there are still a couple of obstacles to overcome.

Under the hood

One problem I had to face was the fact that each parameter type isn't necessarily of datatype string. Instead of specifying variants or using different parameter classes, I just generalized the functions getValue( ) and setValue( ) to be of type string. Why? Because in most cases this parameter mask will be used to pass data on the command line or a file to an interface, such as a report. There you need strings anyway, so I didn't bother about all the conversion routines except in the actual cParameterForm instance itself.

The next issue was the tab order. Just create one parameter form after the other as a child of the dialog, I thought. Well, this doesn't work, because the controls would be in reverse order when using the tab key to jump from one to the other. (By the way, all controls in SQLWindows are created in reverse order at runtime.)

So I need to know the height of all previous parameter windows in order to start painting the client area of the dialog from the bottom up, starting with the last parameter. That's why I started to use recursion in the __dlgParameterSelection.createParameter( ) function. **Listing 4** shows the corresponding code.

Listing 4. Recursive call to "createParameter" ensures the correct tab order in the parameter mask.

```
Function: createParameter
  Set bOk = TRUE
  Select Case Parameter[p_nCounter].ParameterType_ID
    Case PAR_Text
      Set sWindow = '__pFormText'
      Set nHeight = 0.3
      Break
    Case ...
  If p_nCounter < p_nParameterCount - 1
    Set bOk = bOk AND createParameter( p_nPosition +
    nHeight, p_nCounter + 1 )
```

Setup parsing

The dialog box used for creating the parameters, __dlgParameterSelection, expects an array of rParameter UDVs as the first parameter and passes the return values back in a receive string array. The structure of rParameter is shown in **Listing 5**.

Listing 5. UDV for the parameter setup passed to __dlgParameterSelection.

```
Functional Class: rParameter
  Description:
  Derived From
  Class Variables
  Instance Variables
    String: Text
    Number: ParameterType_ID
    Boolean: NotNull
    String: DefaultValue
  Functions
```

Depending on the usage and the concept of your application, parameters are retrieved from a text file, from the database, etc. You have to take care of the necessary parsing yourself, by converting a comma-separated list of

Continues on page 12

Parameter Party …

Continued from page 5

parameter definitions to the rParameter array. Glance at the sample application and you'll see that it's a pretty easy thing to do.

Hands-on

This article's accompanying source code, ParamParty.APP and ParamParty.APL, isn't a complete library; it's a template to get you going. In the APL I defined a couple of business controls that can be extended and adapted to your class library. The application file contains a form window for selecting the needed parameters and shows the corresponding dialog.

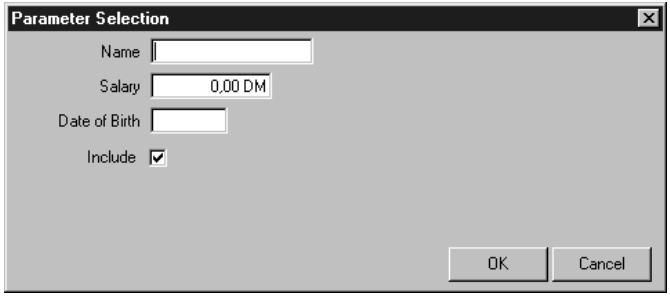

Figure 4. The corresponding parameter mask. The available types of parameters can easily be extended with your own "business controls."

Where to go from here

The technique demonstrated above is pretty simple and straightforward, but still useful. It nicely integrates your application with Crystal Reports, ReportWindows, or SQR reporting server and others. Adapt it for your own work. **CP**

You can download ParamParty.zip from this issue's table of contents at www.ProPublishing.com or find it on this month's Companion Disk.

Based in Hannover, Germany, Thomas Althammer works as an independent IT Consultant and project manager on client/server and Web integration projects. In one of his main projects he develops solutions for social institutions and healthcare associations using Centura products. In addition to being part of the Centura TeamAssist Organisation, he is a founding member of the Ice Tea Group (www.iceteagroup.com), doing consulting for their worldwide clientele. For more information, visit www.althammer.com or send him an email at thomas@althammer.com.